Zunaida Sitorus, S.Si., M.Si.

BUKU REFERENSI

PEMROGRAMAN DAN KOMPUTASI NUMERIK

DARI TEORI KE APLIKASI

IMPI
PT MEDIA PENERBIT INDONESIA

BUKU REFERENSI

PEMROGRAMAN DAN KOMPUTASI NUMERIK DARI TEORI KE APLIKASI

Zunaida Sitorus, S.Si., M.Si.

PT MEDIA PENERBIT INDONESIA

PEMROGRAMAN DAN KOMPUTASI NUMERIK
DARI TEORI KE APLIKASI

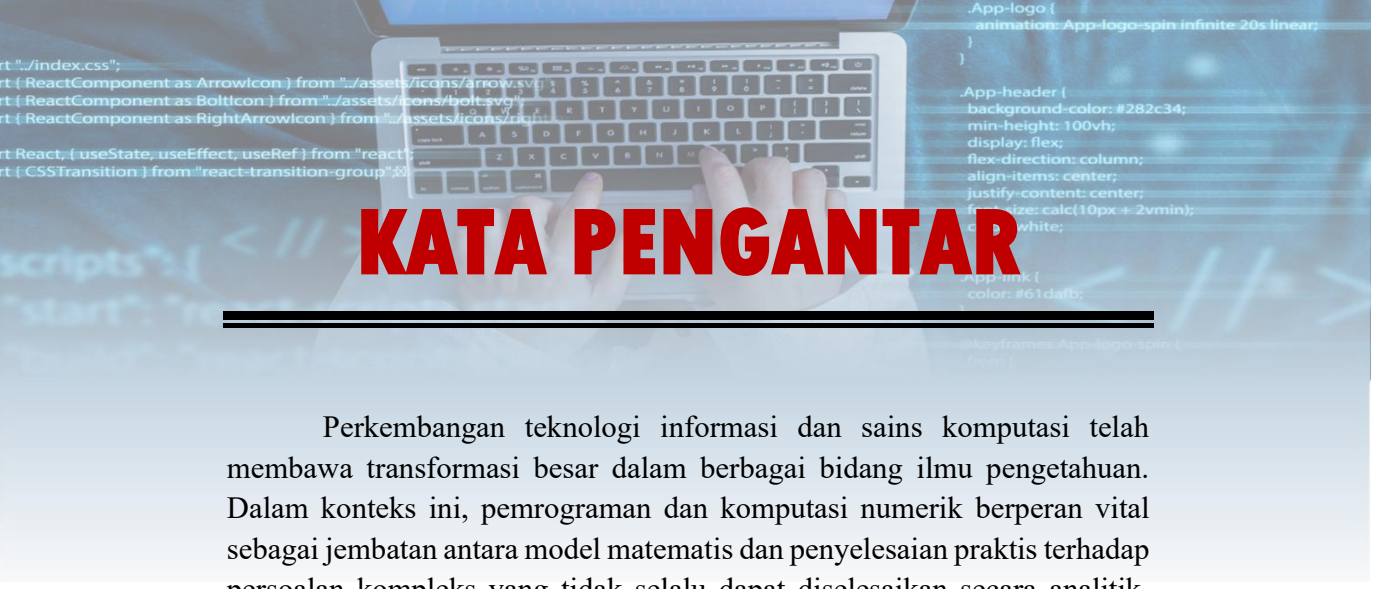
Ditulis oleh:

Zunaida Sitorus, S.Si., M.Si.

ISBN: 978-634-7305-84-8
IV + 219 hlm; 18,2 x 25,7 cm.
Cetakan I, Oktober 2025

Desain Cover dan Tata Letak:
Ajrina Putri Hawari, S.AB.

Diterbitkan, dicetak, dan didistribusikan oleh
PT Media Penerbit Indonesia
Royal Suite No. 6C, Jalan Sedap Malam IX, Sempakata
Kecamatan Medan Selayang, Kota Medan 20131
Telp: 081362150605
Email: ptmediapenerbitindonesia@gmail.com
Web: https://mediapenerbitindonesia.com
Anggota IKAPI No.088/SUT/2024



KATA PENGANTAR

Perkembangan teknologi informasi dan sains komputasi telah membawa transformasi besar dalam berbagai bidang ilmu pengetahuan. Dalam konteks ini, pemrograman dan komputasi numerik berperan vital sebagai jembatan antara model matematis dan penyelesaian praktis terhadap persoalan kompleks yang tidak selalu dapat diselesaikan secara analitik. Mulai dari simulasi teknik, pemodelan ekonomi, hingga analisis data berskala besar, kemampuan untuk menerapkan algoritma numerik dalam bentuk program komputer menjadi keterampilan yang semakin dibutuhkan.

Buku referensi "Pemrograman dan Komputasi Numerik: Dari Teori ke Aplikasi" membahas berbagai konsep dasar dan lanjutan dalam komputasi numerik, mulai dari representasi bilangan dan analisis kesalahan, hingga penyelesaian persamaan aljabar linear, interpolasi, integrasi, dan diferensial numerik. Selain itu, buku referensi ini membahas bahasa pemrograman seperti Python dan MATLAB sebagai alat implementasi algoritma numerik. Buku referensi ini juga membahas seperti optimisasi, komputasi matriks, serta aplikasi dalam bidang teknik, sains, dan keuangan juga disertakan, dilengkapi studi kasus dan latihan untuk memperkuat pemahaman dan keterampilan praktis pembaca.

Semoga buku referensi ini dapat menjadi sumber pengetahuan yang bermanfaat bagi para pembaca dalam memahami dan menguasai konsep serta aplikasi pemrograman dan komputasi numerik.

Salam Hangat

Penulis

DAFTAR ISI

BAB I
PENGANTAR PEMROGRAMAN DAN KOMPUTASI NUMERIK

Pemrograman bukan hanya tentang menulis baris-baris kode, tetapi tentang bagaimana menyusun logika dan algoritma yang efisien untuk menyelesaikan permasalahan nyata secara sistematis. Sementara itu, komputasi numerik hadir sebagai jawaban atas keterbatasan metode analitik dalam menyelesaikan persoalan matematika yang rumit dan tidak memiliki solusi eksak. Dalam bab ini, membahas konsep dasar komputasi numerik, sejarah perkembangannya, serta peran strategisnya dalam berbagai bidang seperti teknik, sains, ekonomi, hingga data science. Pendahuluan ini juga memberikan gambaran mengenai bagaimana komputer merepresentasikan angka dan bagaimana kesalahan dalam perhitungan numerik dapat muncul serta memengaruhi hasil akhir. Dengan pendekatan yang sederhana namun mendalam, bab ini bertujuan untuk membuka wawasan pembaca bahwa komputasi numerik bukan sekadar teori matematis, melainkan alat yang sangat berguna dalam menyelesaikan masalah-masalah nyata yang menuntut ketelitian, kecepatan, dan efisiensi dalam perhitungannya.

A. Definisi dan Ruang Lingkup Komputasi Numerik

Menurut Chapra dan Canale (2010) dalam Numerical Methods for Engineers, komputasi numerik didefinisikan sebagai bidang ilmu yang berkaitan dengan formulasi, pengembangan, dan implementasi

algoritma numerik untuk menyelesaikan permasalahan matematis melalui pendekatan komputasi. Komputasi numerik menggabungkan prinsip-prinsip matematika, ilmu komputer, dan teknik rekayasa dalam menyusun metode-metode yang memungkinkan pemecahan masalah matematika kompleks secara mendekati (aproksimasi), khususnya ketika solusi eksak secara analitik sulit diperoleh. Dalam dunia nyata, banyak permasalahan yang melibatkan data besar, model non-linear, atau bentuk fungsi yang tidak dapat diselesaikan dengan metode analitik konvensional. Di sinilah peran komputasi numerik menjadi sangat vital.

Tujuan utama dari komputasi numerik adalah memperoleh solusi numerik yang mendekati kebenaran aktual dengan tingkat kesalahan yang dapat diterima, serta memastikan metode tersebut dapat diterapkan dalam waktu dan sumber daya komputasi yang efisien. Dengan kata lain, komputasi numerik tidak berusaha menggantikan metode eksak, tetapi melengkapi dan memperluas cakupan penyelesaian masalah matematis yang realistis dan kompleks. Seperti dijelaskan oleh Atkinson (1989) dalam *An Introduction to Numerical Analysis*, pendekatan numerik bertumpu pada keterbatasan sistem digital dalam merepresentasikan bilangan real dan fungsi kontinu. Oleh karena itu, seluruh proses numerik mencakup pengubahan bentuk matematis ke bentuk diskret dan operasional, yang selanjutnya dapat dihitung oleh komputer menggunakan algoritma tertentu. Komputasi numerik mencakup berbagai ruang lingkup yang luas dan beragam. Secara umum, ruang lingkup tersebut dapat dikelompokkan ke dalam beberapa kategori besar berikut:

1. Representasi Bilangan dan Analisis Kesalahan

Pada komputasi numerik, representasi bilangan dan analisis kesalahan merupakan aspek fundamental yang memengaruhi akurasi dan keandalan hasil perhitungan. Komputer tidak dapat merepresentasikan semua bilangan real secara presisi karena keterbatasan dalam sistem bilangan biner dan panjang bit. Sebagaimana dijelaskan oleh Chapra dan Canale (2010), komputer menggunakan sistem *floating point* untuk merepresentasikan bilangan real, yang terdiri dari mantissa dan eksponen. Representasi ini menyebabkan munculnya *round-off error*, yaitu kesalahan akibat pembulatan bilangan yang tidak dapat ditulis secara tepat dalam sistem biner. Sebagai contoh, bilangan desimal seperti 0.1 tidak dapat direpresentasikan secara akurat dalam biner, sehingga

terjadi deviasi kecil yang bisa terakumulasi dalam operasi numerik berulang.

Terdapat pula *truncation error*, yaitu kesalahan yang muncul karena pendekatan suatu metode numerik terhadap bentuk matematis yang sebenarnya. Menurut Burden dan Faires (2011), kesalahan ini sering muncul dalam metode diferensiasi dan integrasi numerik ketika fungsi kontinu diganti dengan aproksimasi diskrit. Pentingnya analisis kesalahan terletak pada kemampuannya untuk memprediksi dan mengendalikan dampak dari ketidakakuratan dalam algoritma numerik. Oleh karena itu, metode numerik yang baik harus memperhitungkan stabilitas numerik yakni kemampuan algoritma untuk membatasi propagasi kesalahan kecil agar tidak menjadi besar secara eksponensial selama proses komputasi.

2. Penyelesaian Persamaan Aljabar

Penyelesaian persamaan aljabar merupakan salah satu fokus utama dalam komputasi numerik karena banyak persoalan dalam sains dan teknik dapat dimodelkan dalam bentuk sistem persamaan, baik linier maupun non-linier. Sistem persamaan linier, seperti Ax=b, sering muncul dalam simulasi struktur, aliran fluida, maupun pemodelan ekonomi. Untuk menyelesaikan sistem ini secara numerik, digunakan berbagai metode seperti eliminasi Gauss, dekomposisi LU, dan metode iteratif seperti Jacobi dan Gauss-Seidel. Menurut Chapra dan Canale (2010), metode eliminasi Gauss merupakan pendekatan langsung (*direct method*) yang efisien untuk sistem ukuran kecil hingga menengah, namun kurang cocok untuk sistem sangat besar karena kompleksitas komputasi dan kebutuhan memori yang tinggi.

Pada kasus sistem non-linier, penyelesaian persamaan semacam f(x)=0 memerlukan pendekatan iteratif, karena bentuk analitiknya sering kali tidak tersedia. Metode numerik yang umum digunakan meliputi metode bisection, secant, dan Newton-Raphson. Menurut Burden dan Faires (2011), metode Newton-Raphson sangat populer karena konvergensinya yang cepat, tetapi memerlukan turunan fungsi dan tebakan awal yang cukup dekat dengan akar sebenarnya agar hasilnya akurat. Di sisi lain, metode bisection lebih stabil tetapi konvergensinya lambat.

Pentingnya penyelesaian persamaan aljabar dalam komputasi numerik terletak pada aplikasinya yang luas di berbagai bidang.

Misalnya, dalam simulasi mekanika struktur, gaya dan respons sistem dirumuskan dalam bentuk sistem persamaan linier. Dalam pemodelan non-linier, seperti perambatan panas atau reaksi kimia, persamaan non-linier menjadi dasar dari model numeriknya. Oleh karena itu, pemahaman tentang metode-metode ini dan perilakunya sangat penting untuk memastikan hasil komputasi yang akurat, stabil, dan efisien.

3. Interpolasi dan Aproksimasi Fungsi

Interpolasi dan aproksimasi fungsi merupakan dua teknik penting dalam komputasi numerik yang digunakan untuk mendekati fungsi-fungsi matematis berdasarkan sejumlah titik data terbatas. Interpolasi bertujuan untuk mencari fungsi yang melewati seluruh titik data yang diberikan secara tepat, sedangkan aproksimasi berusaha mencari fungsi yang "mendekati" data secara keseluruhan, meskipun tidak harus melalui semua titik tersebut. Teknik ini sangat bermanfaat ketika fungsi eksak tidak diketahui, namun tersedia data hasil pengukuran atau simulasi.

Metode interpolasi yang umum digunakan antara lain interpolasi polinomial (seperti interpolasi Lagrange dan Newton) dan interpolasi spline. Interpolasi polinomial bekerja dengan membangun satu polinomial derajat tinggi yang melewati seluruh titik data, tetapi metode ini rentan terhadap fenomena Runge, yaitu osilasi ekstrem pada tepi interval ketika jumlah titik meningkat. Sebagai solusi, interpolasi spline kubik menawarkan alternatif dengan membagi interval menjadi segmen kecil dan menggunakan polinomial derajat rendah pada tiap segmen, sehingga hasilnya lebih halus dan stabil.

Pada aproksimasi, metode least squares sering digunakan untuk mencari fungsi yang meminimalkan selisih kuadrat antara nilai fungsi dan data yang tersedia. Pendekatan ini sangat berguna dalam analisis regresi dan pemodelan data eksperimental. Interpolasi dan aproksimasi tidak hanya digunakan dalam matematika murni, tetapi juga dalam berbagai aplikasi praktis seperti rekonstruksi sinyal digital, pemetaan geografis, grafika komputer, dan pengolahan citra. Keduanya menjadi alat penting dalam menghubungkan data diskrit menjadi representasi fungsi kontinu yang dapat dianalisis lebih lanjut atau digunakan dalam simulasi numerik yang lebih kompleks.

4. Penyelesaian Persamaan Diferensial

Penyelesaian persamaan diferensial secara numerik merupakan komponen penting dalam komputasi ilmiah, karena banyak fenomena alam dan teknik yang dimodelkan menggunakan persamaan diferensial. Persamaan diferensial menggambarkan hubungan antara suatu fungsi dengan turunannya, dan digunakan untuk merepresentasikan perubahan dinamis dalam sistem fisik seperti gerak, panas, pertumbuhan populasi, hingga sirkuit listrik. Dalam praktiknya, persamaan ini terbagi menjadi dua jenis utama: persamaan diferensial biasa (ODE) dan persamaan diferensial parsial (PDE).

Untuk ODE, yaitu persamaan diferensial yang melibatkan satu variabel bebas, metode numerik seperti metode Euler, Runge-Kutta, dan metode Adams-Bashforth digunakan secara luas. Metode Euler, yang paling sederhana, menghitung nilai fungsi ke titik berikutnya menggunakan turunan lokal, namun memiliki tingkat akurasi yang rendah. Sebaliknya, metode Runge-Kutta orde keempat (RK4) menawarkan akurasi yang jauh lebih tinggi dengan tetap menjaga kestabilan komputasi, sehingga lebih banyak digunakan dalam simulasi sistem dinamis.

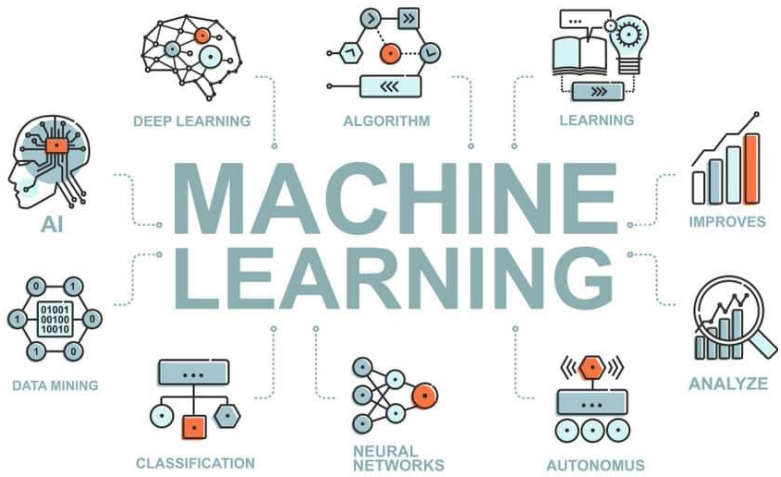
PDE melibatkan lebih dari satu variabel bebas dan sering digunakan untuk memodelkan fenomena dua atau tiga dimensi, seperti perpindahan panas dalam ruang atau perambatan gelombang. Penyelesaiannya memerlukan pendekatan numerik yang lebih kompleks seperti metode beda hingga (*finite difference method*), elemen hingga (*finite element method*), dan volume hingga (*finite volume method*). Metode-metode ini mengubah persamaan diferensial menjadi sistem persamaan aljabar yang dapat diselesaikan secara iteratif oleh komputer.

5. Optimisasi Numerik

Optimisasi numerik merupakan cabang penting dalam komputasi numerik yang fokus pada pencarian nilai minimum atau maksimum dari suatu fungsi, baik dalam ruang satu variabel maupun multivariat. Optimisasi ini sangat relevan dalam berbagai bidang, seperti teknik, ekonomi, sains data, dan *machine learning*, di mana banyak permasalahan nyata yang memerlukan solusi optimal dari suatu model matematis. Misalnya, dalam perencanaan produksi, kita ingin meminimalkan biaya dengan tetap memenuhi permintaan; dalam

machine learning, algoritma pelatihan bertujuan meminimalkan *loss function.*

Gambar 1. *Machine Learning*



Sumber: *Codepolitan*

Optimisasi numerik terbagi menjadi dua kelompok utama: *unconstrained optimization* (tanpa kendala) dan *constrained optimization* (dengan kendala). Untuk kasus tanpa kendala, metode seperti gradient descent, Newton-Raphson, dan *conjugate gradient* banyak digunakan. Metode *gradient descent* bekerja dengan mengikuti arah turunan (gradien) fungsi secara iteratif menuju titik minimum, dan sering digunakan dalam pelatihan model AI. Di sisi lain, metode Newton lebih cepat karena menggunakan informasi turunan kedua (Hessian), tetapi lebih mahal secara komputasi.

Untuk optimisasi dengan kendala, seperti pembatasan sumber daya atau batas nilai variabel, metode seperti Lagrange multipliers dan metode pemrograman kuadrat digunakan. Dalam pendekatan numerik, solusi sering kali tidak diperoleh secara eksak melainkan secara iteratif, sehingga penting untuk memperhatikan aspek konvergensi dan kestabilan algoritma. Penerapan optimisasi numerik sangat luas dan berkembang seiring kemajuan teknologi. Dalam perencanaan kota, optimisasi digunakan untuk mengatur lalu lintas; dalam keuangan, untuk portofolio optimal; dan dalam robotika, untuk menentukan jalur

pergerakan paling efisien. Dengan kemampuan komputer modern, optimisasi numerik telah menjadi alat yang sangat kuat dalam pengambilan keputusan berbasis data dan perancangan sistem yang kompleks dan adaptif.

6. Aplikasi Interdisipliner

Aplikasi interdisipliner dalam komputasi numerik mencerminkan peran vital metode numerik dalam menjembatani berbagai bidang ilmu untuk menyelesaikan persoalan kompleks yang tidak dapat dipecahkan secara analitik. Karena banyak sistem dalam dunia nyata bersifat dinamis, non-linear, dan berbasis data, maka komputasi numerik menjadi kunci dalam membangun model, melakukan simulasi, serta mengevaluasi hasil dalam beragam konteks ilmiah dan praktis.

Pada teknik sipil dan mesin, misalnya, metode numerik digunakan untuk menganalisis tegangan dan deformasi struktur bangunan dengan pendekatan elemen hingga (*finite element method*), serta simulasi aliran fluida dengan metode volume hingga (*finite volume method*). Di bidang fisika dan kimia, komputasi numerik digunakan untuk mensimulasikan dinamika partikel, reaksi kimia, atau perambatan gelombang elektromagnetik. Dalam biologi dan kedokteran, pendekatan numerik diterapkan pada pemodelan penyebaran penyakit, analisis jaringan biologis, hingga simulasi organ virtual untuk keperluan bedah presisi.

Di ranah ekonomi dan keuangan, komputasi numerik sangat berperan dalam pemodelan harga opsi (seperti model *Black-Scholes*), optimisasi portofolio investasi, serta analisis sensitivitas terhadap perubahan pasar. Bahkan di bidang lingkungan dan geografi, metode numerik dimanfaatkan untuk mensimulasikan pola perubahan iklim, pergerakan tanah, atau aliran air dalam sistem hidrologi. Selain itu, dengan kemunculan bidang *data science* dan kecerdasan buatan, metode numerik menjadi tulang punggung dalam pelatihan model pembelajaran mesin melalui optimisasi, regresi, dan aproksimasi fungsi. Kolaborasi lintas disiplin inilah yang menjadikan komputasi numerik tidak hanya sebagai alat matematis, tetapi juga sebagai fondasi teknologi modern yang mengintegrasikan sains, teknik, dan kebijakan dalam pengambilan keputusan yang berbasis data dan simulasi.

B. Perbedaan Metode Analitik vs. Numerik

Menurut Chapra dan Canale (2010) dalam *Numerical Methods for Engineers,* metode analitik dan metode numerik merupakan dua pendekatan utama dalam penyelesaian persoalan matematika dan rekayasa. Keduanya memiliki karakteristik, keunggulan, dan keterbatasan masing-masing. Pemahaman akan perbedaan mendasar antara keduanya sangat penting, terutama dalam memilih pendekatan yang paling sesuai untuk suatu jenis permasalahan dalam konteks akademik maupun praktis.

Metode analitik (*analytical methods*) atau dikenal juga sebagai metode eksak, merupakan pendekatan penyelesaian yang menghasilkan solusi dalam bentuk tertutup (*closed-form solution*). Artinya, solusi diperoleh melalui manipulasi simbolik menggunakan kaidah-kaidah matematika yang telah terdefinisi secara formal. Misalnya, untuk menyelesaikan integral atau turunan, kita dapat menggunakan rumus kalkulus klasik, seperti

$$\int x^2 dx = \frac{1}{3}x^3 + C$$

Pada konteks persamaan diferensial, metode analitik mencakup teknik seperti pemisahan variabel, transformasi Laplace, dan integrasi faktor. Solusi yang diperoleh biasanya dalam bentuk fungsi eksplisit yang dapat dievaluasi untuk nilai tertentu dengan presisi sempurna. Burden dan Faires (2011) dalam *Numerical Analysis* menyatakan bahwa metode analitik cocok untuk sistem yang relatif sederhana dan linier, di mana model matematis dapat dinyatakan dalam bentuk fungsi-fungsi dasar (eksponensial, trigonometri, logaritma, dll.). Sebaliknya, metode numerik (*numerical methods*) adalah pendekatan aproksimatif yang mencari solusi mendekati (*approximate solution*) dari suatu permasalahan matematika, dengan memanfaatkan algoritma dan perhitungan numerik berbasis komputer. Pendekatan ini digunakan ketika solusi analitik sulit atau bahkan mustahil untuk diperoleh.

Contoh klasiknya adalah menyelesaikan persamaan non-linier seperti $e^{-x} = x$, yang tidak memiliki solusi analitik dalam bentuk fungsi eksplisit. Dalam kasus seperti ini, digunakan metode numerik seperti Newton-Raphson atau bisection method untuk mencari nilai x yang mendekati solusi sejati. Menurut Atkinson (1989) dalam *An Introduction to Numerical Analysis,* metode numerik sangat berguna dalam konteks

perhitungan numerik yang kompleks, besar skala, atau tidak dapat dipecahkan secara simbolik. Komputasi numerik memanfaatkan algoritma rekursif, iterasi, dan teknik pendekatan diskrit untuk menggantikan analisis simbolik.

1. Perbandingan Karakteristik Utama

Perbandingan karakteristik utama antara metode analitik dan metode numerik mencerminkan dua pendekatan yang berbeda dalam menyelesaikan permasalahan matematika dan ilmiah, baik dari segi prinsip dasar, teknik eksekusi, hasil yang diperoleh, hingga tingkat fleksibilitasnya. Metode analitik dikenal sebagai pendekatan matematis yang menghasilkan solusi eksak melalui manipulasi simbolik terhadap persamaan yang ada. Misalnya, dalam menyelesaikan turunan suatu fungsi, metode analitik akan menghasilkan bentuk fungsi turunan secara langsung, seperti $\frac{\mathrm{d}}{\mathrm{dx}}(\mathrm{x}^2) = 2\mathrm{x}$. Sebaliknya, metode numerik menghasilkan solusi aproksimasi melalui pendekatan diskrit dan perhitungan iteratif yang dapat dijalankan menggunakan komputer, misalnya dengan memanfaatkan metode *finite difference* untuk menghampiri nilai turunan suatu fungsi berdasarkan data numerik yang terbatas.

Salah satu karakteristik pembeda utama terletak pada jenis solusi yang dihasilkan. Solusi analitik berbentuk tertutup (*closed-form*) dan eksak, sementara solusi numerik bersifat pendekatan (*approximate*) dan bergantung pada nilai awal, parameter langkah, serta struktur algoritma. Oleh karena itu, dalam hal akurasi, metode analitik secara teori lebih unggul karena tidak mengandung kesalahan pembulatan maupun pemotongan, selama manipulasi simbolik dilakukan dengan benar. Namun demikian, metode numerik memungkinkan pengendalian tingkat kesalahan melalui pemilihan ukuran langkah (*step size*), jumlah iterasi, atau tingkat presisi *floating point*.

Dari sisi fleksibilitas dan skalabilitas, metode numerik jauh lebih unggul. Metode analitik hanya dapat diterapkan pada sistem yang bentuk matematikanya relatif sederhana, linier, dan terdefinisi secara simbolik. Ketika berhadapan dengan sistem yang sangat besar, non-linier, atau mengandung data empiris yang tidak berbentuk fungsi eksplisit, metode analitik sering kali gagal. Di sisi lain, metode numerik dapat menangani sistem non-linier, multidimensi, bahkan yang berbasis data diskrit,

seperti yang sering dijumpai dalam pemodelan iklim, rekayasa struktur, atau sistem keuangan.

Pada sumber daya yang dibutuhkan, metode analitik lebih ringan karena hanya membutuhkan keterampilan matematis dan alat tulis, sedangkan metode numerik membutuhkan dukungan komputasi, baik perangkat lunak seperti MATLAB, Python, atau C++, maupun perangkat keras dengan kapasitas pemrosesan tinggi. Hal ini menjadikan metode numerik lebih bergantung pada perkembangan teknologi dan algoritma komputasi.

Kestabilan solusi juga menjadi faktor penting yang membedakan keduanya. Solusi numerik rentan terhadap instabilitas numerik, yaitu situasi di mana kesalahan kecil yang terjadi dalam perhitungan dapat berkembang secara signifikan, menyebabkan hasil yang menyimpang. Oleh karena itu, dalam metode numerik, analisis kestabilan dan konvergensi sangat penting, sedangkan dalam metode analitik, hal ini relatif tidak menjadi isu utama.

Secara umum, metode analitik lebih cocok untuk persoalan sederhana dan sebagai dasar pemahaman matematis, sedangkan metode numerik unggul dalam menangani permasalahan kompleks yang melibatkan banyak variabel, bentuk non-linier, dan pengolahan data besar. Dalam praktik modern, keduanya tidak saling menggantikan tetapi justru saling melengkapi, di mana metode analitik digunakan untuk validasi atau pembuktian konsep, sementara metode numerik digunakan untuk eksplorasi dan simulasi dalam skala besar serta berorientasi pada hasil praktis.

2. Analisis Kesalahan dan Akurasi

Pada komputasi numerik, analisis kesalahan dan akurasi merupakan aspek fundamental yang menentukan seberapa dapat dipercaya hasil perhitungan numerik yang diperoleh. Tidak seperti metode analitik yang menghasilkan solusi eksak, metode numerik hanya memberikan solusi pendekatan yang rentan terhadap berbagai jenis kesalahan. Oleh karena itu, pemahaman mendalam terhadap sumber-sumber kesalahan, cara mengukurnya, serta strategi untuk meminimalkan dampaknya sangat penting dalam praktik komputasi numerik.

Secara umum, kesalahan dalam komputasi numerik dapat diklasifikasikan ke dalam dua jenis utama, yaitu kesalahan pembulatan

(*round-off error*) dan kesalahan pemotongan (*truncation error*). Kesalahan pembulatan terjadi karena komputer hanya mampu merepresentasikan bilangan dalam presisi terbatas, biasanya dalam format *floating point*. Misalnya, bilangan desimal seperti 0.1 tidak dapat direpresentasikan secara tepat dalam biner, sehingga setiap operasi aritmetika dapat menimbulkan deviasi kecil yang terakumulasi selama proses komputasi. Dalam kasus iterasi yang panjang, akumulasi kesalahan pembulatan ini dapat memengaruhi hasil akhir secara signifikan. Sebaliknya, kesalahan pemotongan terjadi ketika suatu proses matematis yang seharusnya berlanjut tanpa batas seperti deret tak hingga atau proses diferensiasi/integrasi, dihentikan pada titik tertentu demi kepraktisan perhitungan. Sebagai contoh, metode numerik seperti metode Euler untuk penyelesaian persamaan diferensial menghampiri solusi dengan interval diskrit, yang pasti berbeda dari solusi kontinu yang sebenarnya.

Untuk mengevaluasi sejauh mana hasil numerik mendekati solusi yang benar, digunakan ukuran seperti galat absolut (*absolute error*) dan galat relatif (*relative error*). Galat absolut adalah selisih antara nilai eksak dan nilai numerik, sedangkan galat relatif menunjukkan proporsi kesalahan terhadap nilai eksaknya. Analisis ini membantu menentukan apakah suatu metode numerik menghasilkan solusi yang cukup akurat untuk tujuan tertentu. Selain itu, konsep kondisioning dan stabilitas numerik menjadi bagian penting dalam analisis kesalahan. Kondisioning mengacu pada sensitivitas masalah terhadap perubahan kecil pada data input, sedangkan stabilitas berkaitan dengan bagaimana kesalahan input atau pembulatan memengaruhi hasil dalam proses algoritma. Sebuah metode disebut stabil jika tidak memperbesar kesalahan kecil menjadi besar selama iterasi.

Pada desain algoritma numerik, perhatian terhadap orde akurasi juga krusial. Metode dengan orde yang lebih tinggi umumnya memberikan hasil yang lebih akurat dengan langkah yang lebih kecil. Misalnya, metode Runge-Kutta orde keempat dalam penyelesaian ODE memberikan akurasi yang lebih tinggi daripada metode Euler dengan langkah yang sama. Namun, akurasi yang tinggi tidak selalu berarti efisien, karena sering kali memerlukan komputasi lebih banyak. Oleh karena itu, dalam praktik, analis numerik harus menyeimbangkan antara akurasi, efisiensi komputasi, dan stabilitas.

C. Jenis Kesalahan: Trunkasi, Pembulatan, dan Presisi

Di dunia komputasi numerik, kesalahan (error) adalah hal yang tidak dapat dihindari. Komputer sebagai alat komputasi digital memiliki keterbatasan dalam merepresentasikan bilangan real dan melakukan operasi matematika yang kompleks secara presisi. Oleh karena itu, hasil dari metode numerik umumnya merupakan aproksimasi terhadap solusi eksak, dan mengandung berbagai jenis kesalahan. Menurut Chapra dan Canale (2010) dalam *Numerical Methods for Engineers*, kesalahan dalam komputasi numerik dapat dikelompokkan ke dalam tiga jenis utama, yaitu kesalahan trunkasi (*truncation error*), kesalahan pembulatan (*round-off error*), dan kesalahan presisi (*precision error*). Masing-masing jenis kesalahan ini memiliki sumber, sifat, dan dampak yang berbeda terhadap hasil akhir komputasi.

1. Kesalahan Trunkasi (*Truncation Error*)

Kesalahan trunkasi (*truncation error*) adalah jenis kesalahan numerik yang muncul akibat pemotongan atau penyederhanaan dari proses matematis yang seharusnya dilakukan secara lengkap atau tak hingga. Dalam konteks komputasi numerik, kesalahan ini terjadi ketika metode analitik yang kompleks, seperti deret tak hingga atau proses kalkulus kontinu, diubah menjadi bentuk diskrit atau dipangkas untuk membuatnya lebih mudah dihitung oleh komputer. Menurut Burden dan Faires (2011) dalam *Numerical Analysis*, kesalahan trunkasi bukan berasal dari representasi bilangan seperti pada kesalahan pembulatan, melainkan dari penggunaan aproksimasi terhadap ekspresi matematis, seperti menghentikan deret Taylor pada suku tertentu atau mengganti integral eksak dengan metode pendekatan numerik seperti trapezoid atau Simpson.

Sebagai contoh, pendekatan turunan pertama dari suatu fungsi f(x) menggunakan metode beda hingga (*finite difference*):

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

Mengandung kesalahan trunkasi karena hanya menggunakan sebagian informasi dari ekspansi Taylor, tanpa memperhitungkan suku-suku berikutnya yang juga memengaruhi nilai turunan sebenarnya. Semakin besar nilai h (*interval diskret*), semakin besar kesalahan trunkasi yang terjadi. Oleh karena itu, salah satu cara untuk mengurangi

kesalahan ini adalah dengan memperkecil nilai h, atau menggunakan metode orde lebih tinggi yang mempertimbangkan lebih banyak suku dalam deret Taylor.

Kesalahan trunkasi juga muncul dalam metode numerik penyelesaian persamaan diferensial, seperti metode Euler atau Runge-Kutta. Misalnya, metode Euler hanya menggunakan gradien pada titik awal untuk memperkirakan nilai berikutnya, dan mengabaikan perubahan gradien yang terjadi sepanjang langkah tersebut. Ini menimbulkan perbedaan antara solusi eksak dan hasil numerik. Analisis terhadap kesalahan trunkasi penting dilakukan untuk menilai seberapa akurat suatu metode numerik dalam menghampiri solusi sebenarnya. Dalam praktiknya, terdapat kompromi antara akurasi dan efisiensi komputasi: semakin kecil kesalahan trunkasi yang diinginkan, semakin kompleks atau banyak komputasi yang dibutuhkan. Oleh karena itu, pemilihan metode dan parameter numerik yang tepat sangat penting untuk menjaga akurasi sambil tetap efisien secara komputasi.

2. Kesalahan Pembulatan (*Round-off Error*)

Kesalahan pembulatan (*round-off error*) adalah kesalahan numerik yang terjadi akibat keterbatasan komputer dalam merepresentasikan bilangan real secara presisi penuh. Komputer modern menggunakan sistem bilangan biner dan format *floating point* (misalnya, IEEE 754) untuk menyimpan dan memproses angka. Karena hanya tersedia sejumlah digit terbatas untuk menyimpan bilangan, maka nilai-nilai desimal yang tidak dapat diwakili secara tepat dalam bentuk biner harus dibulatkan ke angka terdekat yang masih dapat ditampung. Hal ini menyebabkan terjadinya deviasi kecil dari nilai sebenarnya, yang dapat terakumulasi dan berdampak signifikan dalam proses komputasi berulang atau kompleks (Overton, 2001).

Sebagai contoh klasik, bilangan desimal seperti 0.1 tidak dapat dinyatakan secara eksak dalam sistem biner, sehingga saat digunakan dalam perhitungan komputer, nilainya hanya mendekati 0.1 tetapi tidak persis sama. Kesalahan kecil ini mungkin terlihat sepele dalam satu operasi, namun dalam algoritma numerik yang melibatkan ribuan atau bahkan jutaan iterasi, seperti simulasi numerik atau pemrosesan citra digital, akumulasi dari kesalahan pembulatan ini bisa menghasilkan hasil yang keliru secara signifikan. Terutama dalam operasi aritmatika yang sensitif, seperti pengurangan antara dua bilangan yang hampir sama,

fenomena yang disebut loss of significance dapat terjadi, di mana sebagian besar digit penting menghilang akibat pembulatan.

Kesalahan pembulatan juga dipengaruhi oleh jenis presisi yang digunakan: single precision (biasanya 32 bit) dan double precision (64 bit). Double precision memungkinkan representasi angka dengan lebih banyak digit signifikan, sehingga mengurangi kesalahan pembulatan. Namun, penggunaan presisi lebih tinggi juga memerlukan lebih banyak sumber daya komputasi. Untuk memitigasi dampak kesalahan pembulatan, perancang algoritma numerik dapat melakukan berbagai strategi, seperti mengatur ulang urutan operasi (menghindari penjumlahan antara angka besar dan kecil secara langsung), menggunakan teknik compensated summation, atau memilih algoritma yang stabil secara numerik. Dengan memahami dan mengelola kesalahan pembulatan secara cermat, hasil komputasi numerik dapat dibuat lebih andal dan akurat dalam berbagai aplikasi ilmiah dan rekayasa.

3. Kesalahan Presisi (*Precision Error*)

Kesalahan presisi (*precision error*) adalah jenis kesalahan numerik yang berkaitan erat dengan batas kemampuan komputer dalam membedakan dua bilangan yang sangat dekat nilainya, terutama akibat keterbatasan representasi bilangan dalam sistem *floating point*. Komputer menyimpan angka dalam bentuk biner dengan jumlah bit tertentu, seperti single precision (32 bit) atau double precision (64 bit). Setiap format memiliki batas presisi yang disebut machine epsilon, yaitu nilai terkecil yang dapat ditambahkan ke 1 sehingga menghasilkan angka berbeda dari 1 dalam sistem *floating point* tersebut. Dalam sistem double precision IEEE 754, machine epsilon bernilai sekitar 22.2×10^{-16}, yang menunjukkan bahwa setiap perbedaan nilai di bawah angka ini bisa diabaikan oleh komputer (Overton, 2001).

Kesalahan presisi muncul secara nyata ketika kita berurusan dengan angka-angka yang sangat kecil atau sangat besar, atau ketika melakukan operasi antara bilangan yang memiliki magnitudo berbeda secara signifikan. Misalnya, jika dua bilangan seperti 1.000000000000000 dan 1.000000000000001 dikurangkan dalam sistem dengan presisi terbatas, hasil pengurangannya bisa saja menjadi nol karena komputer tidak mampu membedakan keduanya. Fenomena

ini dikenal sebagai cancellation error, dan sering terjadi dalam algoritma yang tidak dirancang untuk mempertimbangkan batas presisi tersebut.

Kesalahan presisi juga memengaruhi hasil dari iterasi numerik, di mana hasil dari satu langkah digunakan sebagai input untuk langkah berikutnya. Jika kesalahan presisi terjadi di awal proses, maka hasil yang terus digunakan dalam iterasi bisa menyebarkan atau bahkan memperbesar kesalahan tersebut. Hal ini menjadi sangat penting dalam metode numerik seperti solusi persamaan diferensial, sistem linier besar, dan optimisasi. Untuk mengurangi dampak kesalahan presisi, beberapa langkah yang dapat diambil antara lain: menggunakan format double precision untuk perhitungan sensitif, menghindari pengurangan antara bilangan yang hampir sama, dan memilih algoritma yang dirancang dengan stabilitas numerik tinggi. Dengan memahami sifat kesalahan presisi, pengguna komputasi numerik dapat merancang solusi yang lebih akurat dan tahan terhadap keterbatasan representasi bilangan digital.

D. Peran Pemrograman dalam Pemecahan Masalah Numerik

Menurut Chapra dan Canale (2010) dalam *Numerical Methods for Engineers,* pemecahan masalah numerik memerlukan serangkaian langkah sistematis yang sering kali tidak dapat dilakukan secara manual, terutama ketika masalah tersebut berskala besar, melibatkan ribuan variabel, atau membutuhkan iterasi kompleks. Dalam konteks ini, pemrograman menjadi perantara yang sangat krusial antara konsep matematika numerik dan implementasi praktisnya. Pemrograman memungkinkan transformasi algoritma matematis ke dalam bentuk yang dapat dijalankan oleh komputer, sehingga mempercepat, mempermudah, dan memperluas cakupan penyelesaian masalah numerik dalam berbagai bidang seperti sains, teknik, keuangan, hingga ilmu data.

1. Pemrograman sebagai Sarana Implementasi Algoritma Numerik

Pemrograman memiliki peran krusial sebagai sarana utama dalam implementasi algoritma numerik, yakni metode matematis yang digunakan untuk menyelesaikan masalah yang tidak dapat diselesaikan secara eksak atau analitik. Algoritma numerik seperti metode bisection, Newton-Raphson, Gauss-Seidel, atau Runge-Kutta memerlukan proses iteratif dan perhitungan yang kompleks, yang secara manual sangat sulit,

lambat, dan rawan kesalahan. Dengan adanya pemrograman, algoritma-algoritma tersebut dapat diubah menjadi rangkaian instruksi logis yang dieksekusi oleh komputer dengan cepat, akurat, dan efisien.

Pemrograman memungkinkan pengguna untuk membuat fungsi modular, mengatur struktur data, melakukan pengulangan (looping), dan mengatur logika percabangan, yang semuanya sangat penting dalam menjalankan algoritma numerik. Sebagai contoh, dalam metode Newton-Raphson untuk mencari akar suatu fungsi non-linier, pemrograman memungkinkan proses iteratif dilakukan otomatis dengan toleransi galat yang dapat disesuaikan. Hal ini membuat algoritma dapat digunakan secara luas dalam berbagai permasalahan, cukup dengan mengganti fungsi yang ingin diselesaikan.

Bahasa pemrograman seperti Python, MATLAB, C++, atau R menyediakan berbagai fitur untuk mempermudah implementasi metode numerik. Python, misalnya, memiliki pustaka NumPy dan SciPy yang menyediakan fungsi bawaan untuk operasi numerik kompleks seperti penyelesaian sistem linier, interpolasi, integrasi numerik, hingga pemodelan diferensial. MATLAB dikenal dengan kemampuannya dalam manipulasi matriks dan visualisasi yang sangat mendukung eksperimen numerik.

Pemrograman juga mendukung proses eksperimen dan validasi algoritma. Pengguna dapat dengan mudah menguji performa algoritma pada berbagai parameter, mengamati konvergensi, dan mengevaluasi kestabilan numerik. Ini memberikan ruang untuk eksplorasi yang luas dalam dunia akademik maupun profesional. Oleh karena itu, penguasaan pemrograman tidak hanya memperkuat kemampuan analisis numerik, tetapi juga membuka jalan untuk pengembangan solusi inovatif berbasis komputasi dalam berbagai bidang ilmu dan teknologi.

2. Bahasa Pemrograman Populer untuk Komputasi Numerik

Di dunia komputasi numerik, pemilihan bahasa pemrograman yang tepat sangat berpengaruh terhadap efisiensi, fleksibilitas, dan keakuratan implementasi algoritma. Beberapa bahasa pemrograman telah terbukti sangat efektif dalam menyelesaikan persoalan numerik karena menyediakan pustaka khusus, kemampuan pemrosesan numerik tinggi, serta kemudahan dalam manipulasi data dan visualisasi. Di antara bahasa yang paling populer digunakan adalah Python, MATLAB, C/C++, dan Fortran.

Python telah menjadi pilihan utama dalam berbagai komunitas ilmiah dan teknik karena sintaksisnya yang sederhana serta dukungan pustaka numerik yang sangat luas. Pustaka seperti NumPy dan SciPy menyediakan fungsi-fungsi penting untuk aljabar linier, kalkulus numerik, statistik, dan integrasi diferensial, yang membuat implementasi algoritma numerik menjadi lebih mudah dan efisien. Python juga mendukung visualisasi hasil perhitungan melalui Matplotlib atau Seaborn, serta terintegrasi dengan pustaka lain seperti Pandas untuk analisis data dan SymPy untuk komputasi simbolik. Kelebihan utama Python adalah kemampuannya untuk beradaptasi lintas disiplin, mulai dari teknik hingga sains data.

MATLAB merupakan bahasa yang secara khusus dirancang untuk perhitungan matriks dan analisis numerik. Dengan lingkungan interaktifnya yang kuat, MATLAB memudahkan pengguna untuk menulis skrip, menguji algoritma numerik, dan memvisualisasikan hasil dalam bentuk grafik atau animasi. MATLAB sangat populer di lingkungan akademik dan industri teknik, khususnya dalam simulasi kontrol, sistem dinamis, dan pemrosesan sinyal.

C dan C++ digunakan ketika performa dan kecepatan eksekusi menjadi prioritas utama, seperti dalam simulasi numerik berskala besar atau real-time. Bahasa ini memungkinkan akses langsung ke memori dan prosesor, serta kompatibel dengan pustaka numerik seperti BLAS dan LAPACK yang mendukung operasi numerik tingkat lanjut. Untuk aplikasi besar dan kompleks, C++ sering dikombinasikan dengan Python (melalui bindings) agar memperoleh keseimbangan antara performa dan kemudahan coding.

Fortran, meskipun tergolong bahasa lama, masih banyak digunakan dalam komputasi ilmiah, khususnya dalam pemodelan fisik dan simulasi cuaca. Bahasa ini dirancang untuk efisiensi dalam pemrosesan numerik dan masih menjadi tulang punggung banyak sistem legacy yang digunakan di institusi riset dan badan antariksa. Setiap bahasa memiliki keunggulan dan kekhususannya, sehingga pemilihan bahasa pemrograman dalam komputasi numerik sangat tergantung pada kebutuhan aplikasi, skala data, serta tingkat kompleksitas algoritma yang akan digunakan.

3. Automasi dan Reproduksibilitas Proses Numerik

Automasi dan reproduksibilitas merupakan dua aspek penting dalam komputasi numerik modern yang sangat dipengaruhi oleh kemampuan pemrograman. Automasi merujuk pada proses menjalankan algoritma numerik secara otomatis tanpa campur tangan manual yang terus-menerus, sedangkan reproduksibilitas mengacu pada kemampuan untuk mengulangi proses komputasi dengan hasil yang konsisten ketika menggunakan data dan parameter yang sama. Dalam konteks pemecahan masalah numerik yang kompleks, kedua aspek ini sangat krusial untuk menjamin efisiensi kerja, keakuratan hasil, dan integritas ilmiah.

Dengan menggunakan bahasa pemrograman seperti Python, MATLAB, atau R, para peneliti dan praktisi dapat mengotomatisasi seluruh rangkaian proses numerik, mulai dari input data, eksekusi algoritma, hingga analisis dan visualisasi hasil. Contohnya, dalam proses simulasi numerik untuk penyelesaian persamaan diferensial parsial (PDE), pengguna bisa menulis skrip yang secara otomatis membaca parameter dari file konfigurasi, menjalankan iterasi hingga konvergensi tercapai, dan menyimpan hasil dalam bentuk grafik atau file data. Ini tidak hanya menghemat waktu, tetapi juga mengurangi risiko kesalahan manusia dalam penginputan data atau pengoperasian perangkat lunak secara manual.

Automasi juga memungkinkan dilakukannya eksperimen numerik berskala besar, seperti studi sensitivitas parameter atau optimisasi multi-variasi, yang akan sangat memakan waktu jika dilakukan secara manual. Dengan pendekatan berbasis kode, ribuan simulasi dapat dijalankan dalam sekali waktu, baik secara berurutan maupun paralel, sehingga mempercepat proses pengambilan keputusan berbasis data.

Reproduksibilitas merupakan fondasi penting dalam dunia akademik dan riset. Ketika proses numerik dituangkan dalam skrip yang terdokumentasi dan terdigitalisasi, siapa pun dapat mengeksekusi ulang perhitungan tersebut dengan hasil identik selama parameter dan lingkungan eksekusinya sama. Hal ini penting untuk validasi, peer review, dan pengembangan lebih lanjut dari penelitian yang sudah ada. Bahkan dalam industri, reproduksibilitas mendukung kontrol kualitas dan pemeliharaan sistem numerik jangka panjang. Dengan demikian, pemrograman tidak hanya menjadi alat bantu teknis, tetapi juga penjamin keberlanjutan, konsistensi, dan kredibilitas proses numerik, baik dalam

skala akademik, industri, maupun kebijakan publik yang berbasis model numerik.

4. Pemrograman untuk Visualisasi dan Analisis Hasil

Pada komputasi numerik, hasil perhitungan sering kali berupa deretan angka atau matriks yang sulit diinterpretasikan secara langsung tanpa bantuan visualisasi. Oleh karena itu, pemrograman berperan penting dalam mentransformasikan hasil numerik menjadi representasi visual yang lebih informatif dan mudah dianalisis. Visualisasi tidak hanya berfungsi sebagai alat bantu presentasi, tetapi juga menjadi sarana eksploratif untuk memahami perilaku sistem, mengidentifikasi pola, menganalisis tren, serta mendeteksi anomali atau kesalahan numerik sejak dini.

Bahasa pemrograman seperti Python, MATLAB, dan R menyediakan pustaka dan fungsi khusus untuk visualisasi data numerik. Di Python, pustaka seperti Matplotlib, Seaborn, dan Plotly memungkinkan pengguna membuat grafik 2D dan 3D, peta kontur, diagram permukaan, hingga animasi dinamis. Misalnya, dalam penyelesaian numerik persamaan diferensial, pengguna dapat memplot solusi terhadap waktu untuk memantau stabilitas dan konvergensi algoritma. Jika solusi tampak mengalami osilasi atau divergensi, maka pengaturan ulang parameter numerik bisa dilakukan sebelum berlanjut ke langkah berikutnya. Dengan demikian, visualisasi berfungsi juga sebagai alat diagnosis numerik.

Analisis hasil numerik juga dapat diotomatisasi melalui pemrograman. Ini mencakup perhitungan galat (error), evaluasi konvergensi, perbandingan metode numerik, serta estimasi performa algoritma berdasarkan waktu eksekusi atau jumlah iterasi. Sebagai contoh, dalam komputasi metode Runge-Kutta untuk ODE, kita bisa membuat skrip yang otomatis membandingkan hasil numerik dengan solusi analitik dan menghitung galat relatif di setiap titik.

Visualisasi yang dihasilkan dari pemrograman juga berperan besar dalam komunikasi ilmiah. Grafik yang jelas dan interaktif membantu menyampaikan temuan kepada audiens teknis maupun non-teknis, termasuk dalam laporan penelitian, presentasi, atau publikasi. Dengan mengintegrasikan hasil komputasi, analisis, dan visualisasi dalam satu alur kerja berbasis kode, pemrograman tidak hanya memperkuat pemahaman hasil, tetapi juga mendorong efisiensi,

transparansi, dan dokumentasi yang baik dalam proses ilmiah dan rekayasa berbasis numerik.

BAB II
BAHASA PEMROGRAMAN UNTUK KOMPUTASI

Di era digital yang serba cepat dan berbasis data, penguasaan bahasa pemrograman menjadi fondasi utama bagi siapa pun yang ingin mengembangkan solusi komputasional terhadap persoalan matematika dan ilmiah yang kompleks. Pemilihan bahasa pemrograman yang tepat tidak hanya berdampak pada efisiensi proses komputasi, tetapi juga pada akurasi, skalabilitas, dan kemudahan integrasi dengan berbagai sistem analitik. Oleh karena itu, bab ini membahas berbagai bahasa pemrograman populer seperti Python, MATLAB, Julia, Fortran, dan C++, serta membahas karakteristik, kelebihan, dan kelemahannya masing-masing dalam konteks numerik. Lebih lanjut, bab ini membahas bagaimana struktur sintaksis, paradigma pemrograman, serta pustaka atau modul yang tersedia dapat memengaruhi kinerja dan efektivitas solusi numerik yang dibangun.

A. Pemilihan Bahasa: Python, MATLAB, atau C++

Pemrograman untuk komputasi numerik menuntut efisiensi, fleksibilitas, dan akurasi dalam menangani data serta proses perhitungan kompleks. Tiga bahasa yang umum digunakan dalam bidang ini adalah Python, MATLAB, dan C++. Setiap bahasa memiliki kekuatan dan kelemahan tersendiri, bergantung pada konteks penggunaannya. Oleh karena itu, pemilihan bahasa pemrograman yang tepat sangat penting dalam menentukan keberhasilan proyek numerik dan ilmiah.

1. Python

Python adalah salah satu bahasa pemrograman paling populer dan serbaguna di era modern, terutama dalam bidang komputasi numerik, data science, kecerdasan buatan, dan pengembangan aplikasi ilmiah. Dikembangkan pertama kali oleh Guido van Rossum pada tahun 1991, Python dirancang dengan filosofi kesederhanaan sintaks dan keterbacaan kode yang tinggi, menjadikannya sangat mudah diakses oleh pemula tanpa mengorbankan kekuatan dan fleksibilitas untuk pengguna tingkat lanjut. Python merupakan bahasa pemrograman tingkat tinggi yang bersifat *open-source*, lintas platform, dan berparadigma multipel mendukung pemrograman prosedural, berorientasi objek, maupun fungsional.

Pada konteks komputasi numerik, Python menonjol karena ketersediaan pustaka (*library*) yang sangat kaya dan kuat. Salah satu pustaka paling fundamental adalah NumPy (*Numerical Python*), yang memungkinkan manipulasi array multidimensi, operasi vektor-matriks, transformasi linier, dan berbagai fungsi matematika tingkat lanjut. NumPy menjadi dasar bagi banyak pustaka numerik lainnya dan memberikan efisiensi komputasi tinggi karena ditulis sebagian besar dalam C, yang membuat Python tetap kompetitif dari sisi performa. Selain itu, SciPy memperluas kemampuan ini dengan menyediakan fungsi-fungsi ilmiah seperti integrasi numerik, optimasi, aljabar linear lanjutan, statistik, dan pemrosesan sinyal.

Python juga sangat unggul dalam bidang visualisasi data. Pustaka seperti Matplotlib memungkinkan pembuatan grafik dua dan tiga dimensi, sedangkan Seaborn dan Plotly menawarkan kemampuan visualisasi statistik dan interaktif yang lebih modern dan estetik. Hal ini sangat penting dalam komputasi numerik karena memungkinkan pengguna tidak hanya menghitung data, tetapi juga memvisualisasikannya untuk pemahaman yang lebih baik dan penyajian hasil yang informatif.

Kelebihan Python tidak berhenti di sana. Dalam praktik pengembangan sistem komputasi yang lebih kompleks, Python dapat diintegrasikan dengan bahasa lain seperti C/C++ menggunakan Cython atau ctypes, serta dengan Fortran melalui f2py, sehingga memungkinkan penggabungan antara kemudahan pemrograman Python dan kecepatan eksekusi dari bahasa compiled. Python juga memiliki kerangka kerja Jupyter Notebook, yang sangat populer di kalangan ilmuwan data dan

akademisi karena memungkinkan kombinasi antara kode, grafik, dan dokumentasi dalam satu antarmuka interaktif.

Ketersediaan komunitas global yang sangat besar, dokumentasi luas, dan pembaruan yang aktif menjadikan Python sangat adaptif terhadap kebutuhan zaman. Tidak heran jika Python kini menjadi bahasa utama dalam banyak bidang dari pengolahan citra, pemodelan keuangan, bioinformatika, hingga komputasi kuantum. Bahkan, banyak lembaga pendidikan dan universitas menggantikan MATLAB atau Java dengan Python dalam pengajaran pemrograman dan matematika komputasi. Namun Python, sebagai bahasa interpreted, memiliki kelemahan dalam hal kecepatan eksekusi murni dibandingkan bahasa compiled seperti C++. Untuk komputasi skala besar atau real-time, optimalisasi kode dan penggunaan pustaka eksternal sering kali dibutuhkan agar performa tetap optimal. Meski demikian, karena kemudahan penggunaan dan skalabilitasnya, Python tetap menjadi pilihan utama bagi banyak praktisi komputasi numerik masa kini.

2. MATLAB

MATLAB, singkatan dari *Matrix Laboratory*, adalah lingkungan komputasi numerik dan bahasa pemrograman tingkat tinggi yang dikembangkan oleh MathWorks. Sejak diperkenalkan pada awal 1980-an oleh Cleve Moler, MATLAB telah menjadi standar industri dan akademik dalam bidang teknik, matematika terapan, dan sains komputer. Fokus utama MATLAB adalah manipulasi matriks, pengembangan algoritma, pemodelan sistem, serta visualisasi dan analisis data. Dengan basis desain yang sangat berorientasi pada komputasi matriks, MATLAB sangat efisien dalam menangani perhitungan numerik, aljabar linear, dan simulasi sistem kompleks yang menjadi tulang punggung di banyak bidang teknik.

Salah satu keunggulan utama MATLAB adalah lingkungan pengembangan terintegrasi (IDE) yang sangat ramah pengguna. Pengguna dapat menulis kode, menjalankan perintah secara interaktif, memvisualisasikan hasil, serta membuat grafik 2D dan 3D dengan sangat mudah. Sintaks MATLAB sangat mirip dengan notasi matematika konvensional, sehingga memudahkan pengguna dari latar belakang non-informatika untuk mengimplementasikan rumus dan algoritma secara langsung tanpa perlu memahami konsep pemrograman tingkat rendah seperti manajemen memori atau pointer. Misalnya, penjumlahan dua

matriks, solusi sistem persamaan linear, atau plotting fungsi bisa dilakukan hanya dengan beberapa baris kode.

MATLAB juga dikenal karena kekayaan toolbox, modul tambahan khusus yang menyediakan fungsi-fungsi siap pakai untuk berbagai disiplin ilmu. Beberapa toolbox populer antara lain *Signal Processing Toolbox, Image Processing Toolbox, Control System Toolbox,* dan *Optimization Toolbox*. Kemampuan ini menjadikan MATLAB sangat disukai dalam lingkungan penelitian dan pengajaran karena memungkinkan eksplorasi dan eksperimen cepat tanpa harus membangun algoritma dari nol. Selain itu, Simulink, sebagai bagian dari MATLAB, merupakan platform pemodelan dan simulasi sistem dinamis berbasis blok diagram yang banyak digunakan di industri otomotif, dirgantara, dan elektronik untuk desain sistem kendali dan sistem embedded.

Pada konteks komputasi numerik tingkat lanjut, MATLAB menyediakan fungsi-fungsi numerik yang sangat stabil dan telah teruji secara luas, seperti metode numerik untuk penyelesaian persamaan diferensial, integrasi numerik, interpolasi, optimasi, dan dekomposisi matriks. Fungsi-fungsi ini dirancang dengan mempertimbangkan kestabilan numerik, efisiensi komputasi, dan kemudahan penggunaan. Selain itu, MATLAB mendukung paralelisasi komputasi dan komputasi GPU melalui Parallel Computing Toolbox, memungkinkan eksekusi program besar atau intensif data secara efisien pada kluster komputer atau perangkat keras modern.

Kelemahan utama MATLAB terletak pada model lisensinya yang komersial dan mahal, baik untuk lisensi individu, institusi, maupun toolbox tambahan. Ini menjadi kendala serius bagi pelajar, institusi kecil, atau proyek open-source yang mengandalkan akses bebas. Selain itu, MATLAB bukanlah bahasa open-source, sehingga pengembangan atau integrasi lintas sistem sering kali tidak sefleksibel bahasa lain seperti Python. Meskipun demikian, untuk proyek-proyek teknik formal dan kebutuhan industri yang menuntut presisi tinggi, dokumentasi kuat, serta dukungan teknis resmi, MATLAB tetap menjadi pilihan unggulan.

3. C++

C++ adalah bahasa pemrograman yang dirancang untuk memberikan kekuatan performa, fleksibilitas, dan kontrol rendah terhadap perangkat keras, menjadikannya sangat ideal untuk

pengembangan aplikasi komputasi numerik berskala besar dan sistem yang memerlukan efisiensi tinggi. Diperkenalkan oleh Bjarne Stroustrup pada awal 1980-an sebagai ekstensi dari bahasa C, C++ menggabungkan paradigma pemrograman prosedural dengan kemampuan berorientasi objek, sekaligus mendukung paradigma generik dan fungsional. Kombinasi ini memberikan kemampuan luar biasa dalam mendesain struktur data kompleks, mengelola memori secara eksplisit, serta menyusun sistem modular dan skalabel, semua sangat penting dalam aplikasi komputasi ilmiah dan teknik.

Pada konteks komputasi numerik, C++ unggul dalam hal kecepatan eksekusi karena merupakan bahasa *compiled*, kode sumbernya dikompilasi langsung menjadi kode mesin sebelum dijalankan. Hal ini memberikan keunggulan signifikan dibanding bahasa interpreted seperti Python atau MATLAB, terutama dalam tugas-tugas intensif seperti simulasi numerik skala besar, pemodelan dinamika fluida, perhitungan finite element, atau pemrosesan data waktu nyata. Selain itu, C++ mendukung pengelolaan memori manual, yang memungkinkan pengguna mengoptimalkan penggunaan RAM dan menghindari overhead dari garbage collection, meskipun hal ini juga menuntut kehati-hatian tinggi agar tidak menyebabkan kebocoran memori (*memory leak*) atau crash.

C++ memiliki ekosistem pustaka numerik yang luas dan kuat. Di antaranya adalah Eigen, sebuah pustaka template untuk aljabar linear, dekomposisi matriks, dan analisis eigenvalue yang sangat efisien dan banyak digunakan dalam pemrosesan citra serta *machine learning*. Ada juga Armadillo, yang menyederhanakan sintaks komputasi numerik dengan performa mendekati Fortran. Boost, salah satu pustaka terlengkap dalam komunitas C++, menyediakan algoritma numerik, struktur data kompleks, dan utilitas lain yang berguna dalam pengembangan aplikasi ilmiah. Pustaka-pustaka ini menjadikan C++ sangat kompetitif dalam membangun sistem komputasi modern yang membutuhkan kombinasi kecepatan dan akurasi.

C++ juga digunakan secara luas dalam pengembangan software sistem dan perangkat keras tertanam (*embedded systems*), seperti firmware, sistem kendali robotik, simulasi fisika, dan grafika komputer. Banyak aplikasi ilmiah dan industri skala besar, seperti OpenFOAM (simulasi fluida), ANSYS (analisis teknik), atau Blender (grafik 3D), menggunakan C++ sebagai bahasa inti karena keunggulannya dalam

menangani perhitungan besar secara efisien dan andal. Namun, kompleksitas sintaks dan kurva pembelajaran yang relatif tinggi menjadi tantangan utama bagi pengguna baru. Penulisan kode yang optimal memerlukan pemahaman mendalam tentang manajemen memori, struktur data, dan prinsip-prinsip pemrograman yang baik. Kesalahan kecil seperti dereferensi pointer yang salah atau *buffer overflow* bisa berakibat fatal pada program. Oleh karena itu, C++ lebih cocok digunakan oleh pengembang yang memiliki pengalaman cukup atau untuk proyek-proyek yang benar-benar membutuhkan efisiensi maksimal.

B. Struktur Dasar Pemrograman: Variabel, Tipe Data, dan Struktur Kontrol

Di dunia pemrograman, memahami struktur dasar adalah fondasi yang sangat penting sebelum seseorang dapat mengembangkan algoritma atau membangun aplikasi yang kompleks. Struktur dasar pemrograman mencakup tiga komponen utama: variabel, tipe data, dan struktur kontrol. Ketiganya membentuk kerangka logika dan operasional dari sebuah program komputer. Tanpa pemahaman yang baik tentang konsep ini, akan sangat sulit untuk membuat program yang efektif, efisien, dan bebas dari kesalahan.

1. Variabel

Variabel merupakan salah satu konsep paling fundamental dalam pemrograman yang berfungsi sebagai penampung data sementara di dalam memori komputer. Dalam istilah sederhana, variabel dapat dianalogikan sebagai "wadah" yang diberi nama tertentu, di mana kita dapat menyimpan nilai, mengubah nilainya, dan menggunakannya kembali dalam berbagai operasi. Variabel memungkinkan suatu program menyimpan informasi secara dinamis selama proses eksekusi berlangsung. Tanpa variabel, program tidak akan mampu menyimpan hasil perhitungan, menampung input pengguna, atau mengatur alur logika berdasarkan data yang berubah-ubah.

Setiap variabel memiliki nama, tipe data, dan nilai. Nama variabel adalah identitas unik yang digunakan untuk merujuk ke nilai yang disimpan. Penamaan variabel biasanya mengikuti aturan sintaks tertentu tergantung bahasa pemrograman yang digunakan, misalnya

harus diawali dengan huruf atau garis bawah (_), tidak mengandung spasi, dan tidak menggunakan kata kunci bawaan bahasa. Pemilihan nama variabel yang baik dan deskriptif, seperti total nilai, nama pengguna, atau kecepatan mobil, sangat dianjurkan untuk meningkatkan keterbacaan dan pemeliharaan kode.

Tipe data yang terkait dengan variabel menentukan jenis nilai yang dapat disimpan di dalamnya, seperti bilangan bulat (integer), bilangan desimal (float atau double), karakter tunggal (char), atau kumpulan karakter (string). Dalam bahasa pemrograman seperti C++, tipe data variabel harus dideklarasikan secara eksplisit. Contoh:

```cpp
int umur = 25;
float tinggi = 172.5;
char huruf = 'A';
```

Sementara dalam bahasa seperti Python, penetapan tipe data dilakukan secara implisit oleh interpreter berdasarkan nilai yang diberikan, karena Python merupakan bahasa bertipe dinamis. Contoh:

```python
umur = 25
nama = "Andi"
```

Variabel dalam pemrograman juga memiliki ruang lingkup (*scope*) dan masa hidup (*lifetime*). Ruang lingkup menunjukkan di bagian mana dari kode program variabel tersebut dapat diakses. Variabel lokal hanya dapat diakses dalam fungsi atau blok tempat ia dideklarasikan, sedangkan variabel global dapat diakses dari seluruh bagian program. Masa hidup variabel berkaitan dengan berapa lama variabel akan "hidup" di dalam memori, biasanya tergantung pada tempat deklarasinya, variabel lokal akan hilang setelah blok program selesai dijalankan, sedangkan variabel global tetap ada sepanjang eksekusi program.

Fungsi utama variabel dalam program adalah untuk menyimpan input, menyimpan hasil perhitungan, mengontrol struktur alur program, dan menyederhanakan penulisan kode. Misalnya, hasil penjumlahan dua angka dapat disimpan dalam variabel hasil, lalu digunakan kembali

dalam operasi atau kondisi berikutnya. Tanpa variabel, setiap nilai harus dihitung dan dituliskan ulang secara manual, yang tidak efisien dan rawan kesalahan.

2. Tipe Data

Tipe data (*data type*) adalah salah satu konsep paling penting dalam pemrograman yang menentukan jenis nilai apa yang dapat disimpan dalam sebuah variabel, serta operasi apa yang sah untuk dilakukan terhadap nilai tersebut. Tipe data mendefinisikan bagaimana data direpresentasikan di dalam memori komputer dan bagaimana bahasa pemrograman memperlakukannya dalam berbagai ekspresi dan fungsi. Tanpa sistem tipe data yang jelas, pengolahan data dalam pemrograman akan menjadi tidak terstruktur dan rentan terhadap kesalahan logika atau sintaks.

Secara umum, tipe data dibedakan menjadi dua kategori besar: tipe data primitif dan tipe data non-primitif (*kompleks*). Tipe data primitif mencakup jenis-jenis data paling dasar seperti bilangan bulat (*integer*), bilangan desimal (*float* atau *double*), karakter (*char*), dan nilai logika (*boolean*). Misalnya, int digunakan untuk menyimpan angka bulat seperti 100, sedangkan float digunakan untuk menyimpan angka pecahan seperti 3.14. Tipe char menyimpan satu karakter tunggal, seperti 'A', dan boolean menyimpan nilai logika true atau false, yang sangat berguna dalam struktur kontrol seperti pernyataan if dan while.

Tipe data non-primitif, di sisi lain, mencakup struktur yang lebih kompleks dan terdiri dari beberapa nilai, seperti *string, array, list, tuple, set, dictionary*, dan objek. Misalnya, *string* adalah kumpulan karakter yang membentuk teks seperti "Halo Dunia", sedangkan *array* menyimpan kumpulan elemen yang sejenis dalam urutan tertentu. Dalam Python, *list* digunakan untuk menyimpan sekumpulan nilai yang dapat terdiri dari berbagai tipe data, dan *dictionary* digunakan untuk menyimpan pasangan kunci-nilai. Sementara dalam bahasa C++, struktur seperti struct dan class memungkinkan programmer untuk mendefinisikan tipe data kustom yang sesuai dengan kebutuhan logika bisnis atau representasi objek dalam dunia nyata.

Setiap bahasa pemrograman memiliki cara tersendiri dalam menangani tipe data. Bahasa seperti C dan C++ bersifat *statically typed*, artinya tipe data harus ditentukan secara eksplisit saat mendeklarasikan variabel. Ini membantu program mendeteksi kesalahan tipe data sejak

proses kompilasi. Sebaliknya, bahasa seperti Python dan JavaScript bersifat *dynamically typed*, yang berarti tipe data ditentukan secara otomatis saat program dijalankan, memberikan fleksibilitas lebih tetapi berisiko menimbulkan kesalahan runtime jika tidak ditangani dengan hati-hati.

Banyak bahasa pemrograman modern mendukung konversi tipe data (*type casting*), yang memungkinkan perubahan tipe data dari satu bentuk ke bentuk lain, seperti dari int ke float, atau dari string ke int. Namun, konversi ini harus dilakukan dengan hati-hati karena berpotensi menyebabkan kehilangan data atau kesalahan logika jika tidak sesuai. Pemilihan tipe data yang tepat sangat penting dalam pengembangan perangkat lunak. Misalnya, menggunakan float untuk perhitungan keuangan dapat menyebabkan ketidakakuratan karena representasi biner angka desimal, sehingga disarankan menggunakan tipe data khusus seperti Decimal dalam Python atau BigDecimal dalam Java. Di sisi lain, penggunaan boolean memungkinkan logika kontrol program menjadi lebih eksplisit dan mudah dimengerti.

3. Struktur Kontrol

Struktur kontrol adalah komponen penting dalam pemrograman yang memungkinkan program untuk mengatur alur eksekusi instruksi berdasarkan kondisi tertentu atau pengulangan perintah. Tanpa struktur kontrol, program hanya akan mengeksekusi perintah secara linear dari atas ke bawah, tanpa kemampuan untuk membuat keputusan atau melakukan iterasi. Struktur kontrol memberikan kemampuan kepada program untuk menjadi dinamis, fleksibel, dan cerdas dalam merespons data atau input yang berubah-ubah. Secara umum, struktur kontrol terbagi menjadi tiga kategori utama: percabangan (*conditional/selection*), perulangan (*looping/iteration*), dan transfer kontrol. Masing-masing kategori memiliki peran yang berbeda namun saling melengkapi dalam menyusun logika program.

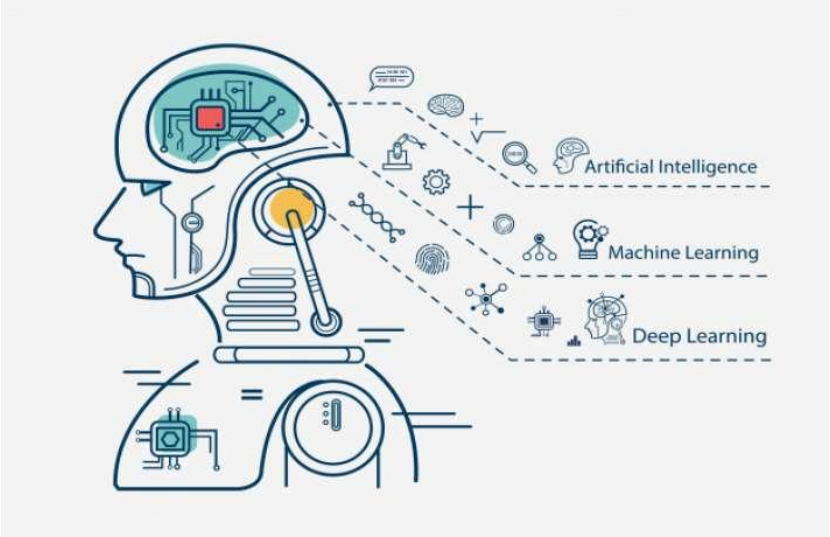
Pertama, struktur percabangan memungkinkan program untuk memilih satu dari beberapa jalur eksekusi berdasarkan kondisi tertentu. Dalam banyak bahasa pemrograman seperti Python, C++, dan Java, bentuk umum dari struktur ini adalah *if, else if* (elif di Python), dan else. Misalnya, jika sebuah nilai memenuhi syarat tertentu (seperti nilai ujian ≥ 75), maka program akan menampilkan "Lulus"; jika tidak, maka akan menampilkan "Tidak Lulus". Selain *if-else*, ada juga *switch-case* dalam

bahasa seperti C++ dan Java yang digunakan untuk menangani banyak kondisi secara lebih terstruktur. Percabangan sangat penting dalam pengambilan keputusan logis, misalnya dalam sistem login, verifikasi data, atau pengkategorian nilai.

Kedua, struktur perulangan (*looping*) digunakan untuk mengeksekusi blok kode secara berulang selama kondisi tertentu masih terpenuhi. Dua bentuk perulangan yang paling umum adalah *for* dan *while*. *For* biasanya digunakan ketika jumlah iterasi sudah diketahui sebelumnya, sedangkan *while* digunakan untuk perulangan yang bergantung pada kondisi yang bersifat dinamis. Contohnya, *for i in range* (10) di Python akan mencetak angka dari 0 hingga 9, sedangkan *while* saldo > 0 bisa digunakan untuk terus mengurangi saldo hingga mencapai nol. *Looping* sangat berguna dalam pemrosesan data, penghitungan matematis berulang, atau pengolahan array dan daftar panjang.

Ketiga, ada transfer kontrol, yaitu perintah khusus yang mengalihkan alur eksekusi program di luar jalur normal. Instruksi seperti *break, continue,* dan *return* termasuk dalam kategori ini. *Break* digunakan untuk keluar dari loop sebelum kondisi selesai, *continue* untuk melewati satu iterasi dan langsung lanjut ke iterasi berikutnya, sedangkan *return* digunakan dalam fungsi untuk mengembalikan nilai sekaligus mengakhiri eksekusi fungsi tersebut. Struktur transfer ini memperkaya fleksibilitas program dalam mengatur alur logikanya secara lebih presisi.

Gambar 2. Kecerdasan Buatan



Sumber: *Codepolitan*

Struktur kontrol juga erat kaitannya dengan pengendalian alur logika algoritmik. Dalam pemrograman tingkat lanjut seperti rekursi, struktur kontrol menjadi instrumen utama dalam mengatur pemanggilan fungsi berulang. Begitu pula dalam pengembangan antarmuka grafis, kecerdasan buatan, dan simulasi fisika komputasional, struktur kontrol berperan sentral dalam membangun perilaku sistem yang adaptif.

C. Fungsi dan Modularisasi Program

Di dunia pemrograman modern, membangun program yang baik bukan hanya soal menghasilkan keluaran yang benar, tetapi juga tentang bagaimana program tersebut disusun secara terstruktur, mudah dibaca, efisien, dan mudah dikelola dalam jangka panjang. Fungsi (*function*) dan modularisasi program merupakan dua konsep penting yang menjadi landasan dalam pencapaian tujuan tersebut. Fungsi memungkinkan programmer memecah program menjadi bagian-bagian kecil yang dapat digunakan kembali, sementara modularisasi menciptakan arsitektur sistem yang lebih tertata dan fleksibel. Keduanya merupakan prinsip utama dalam rekayasa perangkat lunak berbasis praktik terbaik dan sangat penting dalam pengembangan perangkat lunak skala besar.

Menurut Kernighan dan Ritchie (1988) dalam *The C Programming Language,* fungsi adalah blok kode mandiri yang dirancang untuk melakukan tugas tertentu dan dapat dipanggil berulang kali dari bagian lain dalam program (Kernighan & Ritchie, 1988). Dengan kata lain, fungsi bertindak seperti "mesin kecil" yang menerima input (parameter), memprosesnya, dan mengembalikan hasil (*return value*) tanpa harus menulis ulang kode yang sama di berbagai tempat.
Contoh sederhana fungsi dalam Python dan C++:

```python
def hitung_luas_persegi(sisi):
    return sisi * sisi
```

```cpp
int hitungLuasPersegi(int sisi) {
    return sisi * sisi;
}
```

Kedua contoh di atas menunjukkan bagaimana logika perhitungan dapat dibungkus ke dalam satu fungsi yang bisa digunakan berulang kali, cukup dengan memanggil nama fungsinya dan memberikan parameter yang sesuai.

1. Manfaat Fungsi

Fungsi merupakan salah satu komponen kunci dalam pemrograman modern yang memberikan berbagai manfaat penting dalam menyusun kode yang efisien, terstruktur, dan mudah dikelola. Fungsi adalah blok kode mandiri yang dirancang untuk melakukan tugas tertentu. Dengan memisahkan logika program ke dalam fungsi-fungsi kecil, seorang programmer dapat menciptakan sistem yang lebih modular, mudah dipahami, dan dapat digunakan kembali. Salah satu manfaat utama dari penggunaan fungsi adalah reusabilitas, yakni kemampuan untuk menggunakan ulang potongan kode yang sama berkali-kali tanpa harus menuliskannya dari awal. Ini tidak hanya menghemat waktu, tetapi juga mengurangi kemungkinan terjadinya kesalahan penulisan (*human error*) akibat duplikasi kode.

Fungsi juga meningkatkan keterbacaan (*readability*) dan kejelasan struktur program. Ketika program dibagi ke dalam fungsi-fungsi yang memiliki nama deskriptif, seperti hitung gaji, cek login, atau tampilkan menu, pembaca kode akan lebih mudah memahami alur program secara keseluruhan tanpa harus menelusuri seluruh detail implementasi di setiap bagian. Ini sangat membantu dalam tim pengembangan perangkat lunak, di mana kolaborasi antarprogrammer menjadi lebih efektif karena pembagian tugas dapat dilakukan berdasarkan fungsi.

Fungsi memungkinkan isolasi logika, yang berarti setiap bagian program dapat diuji, diperbaiki, atau dimodifikasi tanpa memengaruhi bagian lain. Pendekatan ini mendukung prinsip separation of concerns dalam rekayasa perangkat lunak, yaitu memisahkan tanggung jawab logika program ke dalam unit-unit kecil yang fokus pada satu tugas. Dengan cara ini, pemeliharaan program (*maintenance*) menjadi lebih mudah karena bug dapat dilokalisasi di dalam fungsi tertentu tanpa menelusuri keseluruhan sistem. Lebih jauh, fungsi juga mendukung pengembangan bertahap dan uji unit (*unit testing*). Karena setiap fungsi dapat dieksekusi secara independen, pengembang dapat menguji satu per satu fungsi secara terpisah sebelum mengintegrasikannya ke dalam

sistem utama. Ini mempercepat proses debugging dan meningkatkan keandalan program.

2. Parameter, Return, dan Scope

Pada pemrograman, ketika kita menggunakan fungsi, tiga konsep penting yang perlu dipahami dengan baik adalah parameter, return, dan scope. Ketiganya berkaitan erat dengan bagaimana fungsi berkomunikasi dengan bagian lain dari program serta bagaimana data dikirim, diproses, dan dikembalikan dalam alur eksekusi. Parameter adalah nilai yang dikirimkan ke dalam fungsi saat fungsi dipanggil. Parameter memungkinkan fungsi bekerja secara fleksibel terhadap berbagai input tanpa harus menulis ulang logika kode. Misalnya, fungsi hitung luas (sisi) menerima satu parameter sisi yang dapat bernilai apa saja, sehingga fungsinya bisa digunakan untuk menghitung luas dari berbagai ukuran persegi. Ada dua jenis parameter utama: parameter formal, yaitu yang dideklarasikan dalam definisi fungsi, dan parameter aktual (argumen), yaitu nilai yang diberikan saat fungsi dipanggil.

Return adalah nilai yang dikembalikan oleh fungsi kepada pemanggilnya setelah fungsi selesai diproses. Return memungkinkan hasil dari suatu perhitungan atau proses dalam fungsi digunakan kembali di bagian lain program. Misalnya, return sisi-sisi akan mengembalikan nilai luas ke pemanggilnya, yang bisa disimpan dalam variabel lain atau langsung ditampilkan. Sementara itu, scope atau ruang lingkup, mengatur di mana variabel dapat diakses dalam program. Variabel yang dideklarasikan di dalam fungsi hanya berlaku di dalam fungsi itu saja dan disebut variabel lokal. Sebaliknya, variabel global dideklarasikan di luar fungsi dan bisa diakses dari manapun dalam program. Memahami scope penting untuk mencegah konflik antarvariabel dan menjaga agar data dalam fungsi tidak "bocor" ke luar, yang dapat menyebabkan kesalahan logika.

3. Modularisasi Program

Modularisasi program adalah pendekatan dalam pemrograman yang bertujuan untuk memecah sistem atau program besar menjadi bagian-bagian kecil yang disebut modul, di mana masing-masing modul memiliki tanggung jawab khusus dan independen. Pendekatan ini sangat penting dalam pengembangan perangkat lunak karena mempermudah manajemen kompleksitas, meningkatkan keterbacaan kode, serta

memfasilitasi pemeliharaan dan pengembangan berkelanjutan. Modularisasi mendukung prinsip desain perangkat lunak seperti *separation of concerns* dan *single responsibility*, yang menekankan bahwa setiap bagian dari program sebaiknya hanya melakukan satu tugas tertentu.

Setiap modul dalam program bisa berupa fungsi, kelas, atau bahkan file terpisah yang memiliki logika tertentu, dan biasanya dapat dipanggil dari bagian program lain melalui antarmuka (*interface*) yang jelas. Misalnya, dalam sebuah aplikasi sistem kasir, modul-modul yang umum digunakan meliputi modul input transaksi, modul hitung diskon, modul cetak struk, dan modul laporan harian. Setiap modul ini bisa dikembangkan, diuji, dan dimodifikasi secara terpisah tanpa mengganggu modul lain. Hal ini memberikan keuntungan besar dalam pengembangan tim, karena beberapa programmer bisa bekerja secara paralel pada modul berbeda.

Salah satu manfaat utama dari modularisasi adalah reusabilitas kode. Modul yang dirancang dengan baik dapat digunakan kembali di berbagai proyek atau bagian lain dari sistem tanpa perlu menyalin ulang kode. Selain itu, modularisasi meningkatkan kemudahan pengujian (*testability*), karena setiap modul bisa diuji secara terpisah melalui teknik unit testing, sehingga memudahkan deteksi dan perbaikan bug secara lebih cepat dan akurat.

Modularisasi juga berkontribusi pada efisiensi pengembangan dan perawatan sistem jangka panjang. Dalam sistem besar yang terus berkembang, kebutuhan akan pembaruan, penggantian logika bisnis, atau penambahan fitur baru sangat tinggi. Dengan struktur modular, pengembang dapat fokus pada bagian tertentu tanpa harus memahami seluruh program secara keseluruhan. Ini sangat krusial untuk memastikan keberlanjutan sistem dalam jangka waktu yang lama, terutama ketika terjadi pergantian tim pengembang.

Di sisi teknis, modularisasi juga mendukung penggunaan kembali pustaka eksternal (*library*) dan pemanfaatan framework modern yang berbasis arsitektur modular, seperti penggunaan modul dalam Python (dengan import), file header dan source terpisah di C++, atau modul service dalam arsitektur berbasis microservices. Dengan demikian, modularisasi bukan sekadar praktik struktural, tetapi merupakan strategi penting dalam menyusun program yang fleksibel, scalable, dan maintainable. Kemampuan untuk memecah permasalahan

besar menjadi bagian-bagian kecil yang bisa dikelola secara terpisah adalah ciri utama dari pengembang perangkat lunak yang profesional. Oleh karena itu, modularisasi program menjadi prinsip mendasar yang wajib dikuasai dalam dunia pemrograman modern.

D. Visualisasi Data Numerik (Plotting dan Grafik)

Di era informasi yang didominasi oleh data, visualisasi data numerik menjadi salah satu alat paling penting untuk membantu pengguna memahami pola, tren, dan anomali dalam kumpulan data yang kompleks. Visualisasi data, khususnya dalam bentuk plotting dan grafik, merupakan proses transformasi angka-angka mentah menjadi representasi visual yang lebih mudah dipahami dan dianalisis. Terutama dalam bidang komputasi numerik, sains data, dan teknik, visualisasi bukan hanya alat bantu tambahan, melainkan bagian esensial dari proses eksplorasi, analisis, dan komunikasi hasil.

Menurut Ware (2012) dalam *Information Visualization: Perception for Design*, representasi visual membantu otak manusia memproses informasi secara lebih efisien dibandingkan dengan teks atau angka mentah, karena visualisasi mampu memanfaatkan kekuatan persepsi spasial dan pengenalan pola visual secara alami (Ware, C., 2012). Dalam konteks data numerik, ini sangat relevan karena sebagian besar data yang diolah berupa angka dalam jumlah besar, yang sulit ditafsirkan secara langsung tanpa representasi visual.

1. Jenis-Jenis Grafik dalam Visualisasi Numerik

Pada visualisasi data numerik, pemilihan jenis grafik yang tepat sangat penting untuk menyampaikan informasi dengan jelas dan akurat. Berbagai jenis grafik dirancang untuk membahas aspek yang berbeda dari data, seperti distribusi, hubungan antar variabel, komparasi antar kategori, maupun tren terhadap waktu. Setiap jenis grafik memiliki kekuatan tersendiri dalam mengungkapkan pola-pola tersembunyi dalam angka-angka mentah. Grafik garis (*line chart*) adalah salah satu jenis grafik paling umum dalam visualisasi numerik. Grafik ini digunakan untuk menampilkan perubahan nilai dari waktu ke waktu, seperti pertumbuhan populasi, harga saham, atau suhu harian. Karena kemampuannya menunjukkan arah tren secara halus, grafik garis sangat efektif dalam mengilustrasikan dinamika temporal dari data kontinu.

Grafik batang (*bar chart*) digunakan untuk membandingkan nilai antar kategori diskrit. Misalnya, perbandingan hasil penjualan antar produk, jumlah siswa per jurusan, atau pengeluaran tahunan berdasarkan sektor. Bar chart memudahkan pengguna melihat kategori mana yang paling dominan atau paling rendah, terutama dalam kasus data terklasifikasi. Histogram, meskipun tampak mirip dengan bar chart, berfungsi untuk menunjukkan distribusi frekuensi dari data numerik yang dibagi dalam rentang interval. Histogram sangat berguna untuk mengetahui sebaran nilai, seperti dalam pengukuran statistik tinggi badan, waktu proses, atau nilai ujian.

Scatter plot (diagram sebar) memvisualisasikan hubungan antara dua variabel numerik. Ini sangat penting dalam analisis korelasi dan regresi, di mana kita ingin tahu apakah perubahan satu variabel berkaitan dengan perubahan variabel lain. Scatter plot juga berguna untuk mendeteksi outlier. *Box plot* menampilkan ringkasan statistik dari data, termasuk median, kuartil, dan pencilan (outlier). Grafik ini sangat berguna dalam analisis komparatif antar kelompok, misalnya membandingkan nilai ujian antar kelas atau distribusi pendapatan antar wilayah. Jenis lainnya termasuk *pie chart* untuk proporsi, heatmap untuk korelasi, serta *surface plot* dan contour plot dalam visualisasi tiga dimensi atau data spasial. Pemilihan jenis grafik harus mempertimbangkan jenis data, tujuan analisis, dan target audiens agar informasi yang ditampilkan benar-benar membantu pemahaman dan pengambilan keputusan.

2. Tools dan Library untuk Plotting Data

Di dunia komputasi numerik dan analisis data, keberadaan tools dan library untuk plotting data sangat penting dalam mendukung visualisasi yang efektif. Alat-alat ini memungkinkan pengguna untuk mengubah data numerik menjadi representasi grafis yang intuitif, seperti grafik garis, batang, scatter, dan histogram. Berbagai bahasa pemrograman populer seperti Python, MATLAB, R, dan platform visualisasi modern menyediakan beragam pustaka dan antarmuka visual yang memudahkan proses ini, mulai dari eksplorasi data awal hingga presentasi akhir.

Python merupakan salah satu bahasa pemrograman yang paling banyak digunakan dalam sains data dan visualisasi, karena memiliki ekosistem pustaka yang kuat dan fleksibel. Matplotlib, pustaka plotting

dasar di Python, memungkinkan pembuatan grafik 2D dengan kontrol penuh terhadap setiap elemen visual, seperti judul, label sumbu, warna, dan gaya garis. Untuk visualisasi statistik yang lebih estetis dan cepat, Seaborn menjadi pilihan favorit, karena dibangun di atas Matplotlib dan mampu membuat grafik seperti boxplot, heatmap, dan violin plot dengan sintaks yang ringkas. Selain itu, Plotly dan Bokeh digunakan untuk membuat grafik interaktif berbasis web yang sangat cocok untuk dashboard dan aplikasi visualisasi data real-time.

MATLAB adalah tool proprietary yang sangat populer di bidang teknik dan komputasi ilmiah. MATLAB menyediakan fungsi plotting seperti plot, bar, surf, dan contour, yang sangat ideal untuk menampilkan hasil komputasi numerik, simulasi, atau visualisasi fungsi matematis dalam bentuk 2D maupun 3D. MATLAB dikenal karena kemudahan penggunaannya serta kualitas grafik yang tinggi dan dapat dikustomisasi. Untuk kebutuhan visualisasi tanpa kode, tersedia alat seperti Microsoft Excel, Google Sheets, dan Tableau. Excel dan Google Sheets cocok untuk visualisasi sederhana berbasis spreadsheet, seperti grafik batang dan pie chart. Sedangkan Tableau menawarkan antarmuka drag-and-drop untuk membuat visualisasi interaktif kompleks yang terhubung ke berbagai sumber data.

3. Visualisasi dalam Proses Analisis dan Komputasi

Di dunia analisis data dan komputasi numerik, visualisasi bukan hanya alat presentasi akhir, tetapi bagian penting dari keseluruhan proses analisis yang membantu pengguna memahami, memverifikasi, dan mengkomunikasikan hasil dengan lebih baik. Visualisasi berperan sejak tahap eksplorasi awal data (*exploratory data analysis*/EDA), hingga validasi model dan pelaporan hasil. Dengan mengubah angka-angka menjadi bentuk grafis yang dapat dilihat secara intuitif, visualisasi memungkinkan deteksi pola, anomali, atau kesalahan yang mungkin tidak terlihat hanya melalui tabel data.

Pada tahap eksplorasi data, visualisasi membantu pengguna mengenali distribusi, tren waktu, atau hubungan antar variabel. Misalnya, dalam pemodelan statistik atau *machine learning*, scatter plot dapat digunakan untuk melihat korelasi antara dua variabel numerik sebelum diterapkan regresi. Demikian juga, histogram atau box plot berguna untuk mengevaluasi persebaran dan outlier, yang sangat penting untuk menjaga integritas model komputasi. Pada konteks komputasi

numerik, visualisasi berperan penting dalam memantau proses iteratif atau solutif, seperti penyelesaian persamaan diferensial numerik, simulasi fluida (CFD), atau optimasi non-linear. Sebagai contoh, dalam metode Euler atau Runge-Kutta, grafik solusi terhadap waktu membantu mengevaluasi stabilitas dan akurasi pendekatan numerik yang digunakan. Tanpa visualisasi, peneliti hanya akan melihat deretan angka yang sulit dievaluasi secara intuitif.

Visualisasi berperan dalam verifikasi dan validasi model komputasi. Hasil simulasi atau prediksi dapat dibandingkan dengan data aktual melalui grafik overlay, sehingga memudahkan penilaian terhadap tingkat kesesuaian model. Bahkan dalam pengembangan sistem berbasis kecerdasan buatan, seperti neural network, visualisasi dari loss function atau akurasi terhadap epoch sangat penting dalam menentukan keberhasilan proses pelatihan. Akhirnya, dalam pelaporan dan komunikasi hasil analisis, visualisasi mempermudah penyampaian informasi kepada pihak yang tidak teknis. Grafik yang tepat dapat menjembatani pemahaman antara analis dan pengambil keputusan. Oleh karena itu, integrasi visualisasi dalam setiap tahap analisis dan komputasi adalah praktik terbaik yang wajib diterapkan dalam pengolahan data modern.

BAB III
REPRESENTASI BILANGAN DAN ARITMETIKA KOMPUTASI

Representasi bilangan dan aritmetika komputasi merupakan fondasi utama dalam memahami cara kerja sistem komputasi modern. Dalam dunia digital, bilangan tidak disimpan sebagaimana manusia memahaminya dalam bentuk desimal, melainkan dalam representasi biner, oktal, atau heksadesimal yang lebih sesuai dengan arsitektur perangkat keras. Pemahaman mengenai bagaimana bilangan bulat, bilangan pecahan, maupun bilangan *floating-point* direpresentasikan dalam komputer sangat penting untuk menghindari kesalahan komputasi yang tampak sepele namun berdampak besar, seperti pembulatan atau *underflow* dan *overflow*. Aritmetika komputasi juga menyangkut operasi-operasi dasar seperti penjumlahan, pengurangan, perkalian, dan pembagian yang dilakukan dalam format terbatas dan presisi tertentu. Dalam bab ini, membahas bagaimana komputer menangani bilangan secara internal, termasuk struktur IEEE 754 untuk *floating-point*, serta bagaimana kesalahan numerik dapat muncul dan dikendalikan.

A. Representasi Bilangan *Floating point* dan Biner

Di dunia komputasi modern, representasi data numerik berperan yang sangat penting. Komputer tidak bekerja dengan angka sebagaimana manusia melakukannya; melainkan, semua bentuk data, termasuk angka, dikodekan dalam format biner. Salah satu format paling umum untuk merepresentasikan angka pecahan dalam komputer adalah *floating point*. Representasi ini memungkinkan komputer menangani berbagai angka

dengan rentang yang luas, baik sangat besar maupun sangat kecil, dengan tingkat presisi yang terkontrol.

1. Representasi Biner

Representasi biner adalah sistem bilangan yang hanya menggunakan dua simbol, yaitu 0 dan 1, untuk menyatakan semua jenis data dalam komputer. Sistem ini menjadi dasar dalam dunia komputasi karena perangkat keras komputer seperti transistor dan sirkuit digital hanya mengenali dua keadaan logika: on (1) dan off (0). Dengan menggunakan kombinasi bit-bit ini, komputer dapat merepresentasikan angka, karakter, instruksi, hingga gambar dalam bentuk yang dapat diolah secara elektronik.

Pada konteks bilangan bulat, representasi biner bekerja berdasarkan posisi bit yang merepresentasikan pangkat dua. Sebagai contoh, bilangan desimal 13 ditulis sebagai 1101 dalam biner, yang berarti:

$$(1 \times 2^3) + (1 \times 2^2) + (0 \times 2^1) + (1 \times 2^0) = 8 + 4 + 0 + 1 = 13$$

Untuk bilangan negatif, digunakan metode komplemen dua (two's complement) agar perhitungan aritmetika tetap efisien dalam operasi logika. Misalnya, -5 dalam 8-*bit two's complement* ditulis sebagai 11111011.

Bilangan pecahan atau angka desimal dalam biner direpresentasikan dengan memperluas sistem posisi ke bagian kanan titik biner (*binary point*), menggunakan nilai-nilai seperti $2^{-1}2^{-2}$, dan seterusnya. Contohnya, bilangan 0.625 dalam biner adalah 0.101, karena:

$$(1 \times 2^{-1}) + (0 \times 2^{-2}) + (1 \times 2^{-3}) = 0.5 + 0 + 0.125 = 0.625$$

Meskipun efisien, sistem biner memiliki keterbatasan dalam merepresentasikan beberapa bilangan desimal secara eksak. Sebagai contoh, bilangan 0.1 tidak dapat direpresentasikan secara tepat dalam bentuk biner terbatas, menyebabkan kesalahan pembulatan dalam komputasi. Oleh karena itu, pemahaman tentang representasi biner menjadi krusial bagi siapa pun yang bekerja di bidang komputasi, teknik, maupun sains data untuk memastikan hasil perhitungan yang akurat dan dapat dipertanggungjawabkan.

2. *Floating Point*

Floating point adalah format representasi bilangan dalam komputer yang dirancang untuk menyatakan angka-angka real, baik sangat besar maupun sangat kecil, dengan efisien dan presisi terbatas. Berbeda dengan bilangan bulat (*integer*) yang memiliki nilai tetap dalam rentang tertentu, bilangan *floating point* memungkinkan adanya eksponen untuk memperluas cakupan nilai yang bisa direpresentasikan. Konsep *floating point* dapat dianalogikan seperti notasi ilmiah dalam matematika. Format *floating point* yang paling umum digunakan di seluruh sistem komputasi modern adalah standar IEEE 754, yang menetapkan aturan representasi 32-bit (*single precision*) dan 64-bit (*double precision*).

Pada struktur IEEE 754, satu angka *floating point* terdiri atas tiga bagian utama: bit tanda (*sign bit*), eksponen, dan mantissa. Misalnya, dalam format 32-bit: 1 bit digunakan untuk tanda (positif atau negatif), 8 bit untuk eksponen (dengan bias 127), dan 23 bit untuk mantissa. Nilai aktual bilangan dihitung dengan rumus:

$$(-1)^s \times 1.m \times 2^{(e-bias)}$$

Keunggulan utama *floating point* adalah kemampuannya merepresentasikan nilai sangat besar seperti 1038 dan nilai sangat kecil seperti 10-38, yang penting dalam aplikasi ilmiah seperti simulasi fisika, pemodelan keuangan, dan *machine learning*. Namun, karena hanya sejumlah bit yang tersedia untuk menyimpan mantissa dan eksponen, representasi ini rawan terhadap kesalahan pembulatan, *overflow*, dan underflow. Akibatnya, programmer harus waspada terhadap keterbatasan presisi dan efek numerik yang mungkin terjadi dalam perhitungan. Oleh karena itu, *floating point* bukan hanya solusi teknis, melainkan juga tantangan logika dan presisi dalam komputasi numerik.

3. Perbandingan

Di dunia komputasi numerik, representasi bilangan dapat dibedakan menjadi dua kategori utama: *floating point* dan *fixed point* (representasi tetap). Keduanya memiliki fungsi yang sama, yaitu menyimpan dan memproses angka pecahan atau bilangan real, namun dengan pendekatan dan karakteristik teknis yang sangat berbeda. *Floating point*, seperti yang diatur oleh standar IEEE 754,

memungkinkan representasi angka dalam rentang yang sangat luas melalui penggunaan eksponen berbasis dua. Format ini sangat ideal untuk aplikasi yang memerlukan skala angka yang besar atau kecil seperti simulasi ilmiah, grafik komputer, dan analisis statistik karena dapat secara fleksibel menyesuaikan posisi titik desimal (*floating*) tergantung besar kecilnya angka.

Representasi tetap (*fixed point*) menetapkan posisi titik desimal pada tempat yang konstan. Hal ini menjadikannya lebih sederhana secara implementasi dan efisien dalam hal penggunaan memori serta kecepatan eksekusi, terutama pada sistem tertanam (*embedded systems*) seperti mikrokontroler dan perangkat IoT. Namun, *fixed point* memiliki keterbatasan rentang nilai dan presisi karena tidak mendukung eksponen. Akibatnya, angka yang terlalu besar atau kecil dapat dengan mudah mengalami *overflow* atau *truncation*.

Perbandingan keduanya menunjukkan adanya *trade-off* antara fleksibilitas dan efisiensi. *Floating point* unggul dalam hal presisi dinamis dan cakupan nilai, tetapi membutuhkan perangkat keras yang lebih kompleks dan mahal. *Fixed point* lebih hemat sumber daya dan cocok untuk aplikasi *real-time* dengan batas presisi yang dapat dikontrol. Dalam praktiknya, pemilihan antara keduanya sangat tergantung pada kebutuhan aplikasi: *floating point* untuk komputasi ilmiah berskala besar, dan fixed point untuk sistem dengan keterbatasan sumber daya namun memerlukan performa tinggi dan prediktabilitas.

B. Stabilitas dan Propagasi Kesalahan

Pada komputasi numerik, setiap perhitungan yang dilakukan oleh komputer tidak lepas dari kemungkinan kesalahan. Hal ini disebabkan oleh keterbatasan representasi bilangan dalam format biner dan *floating point*, serta akumulasi kesalahan selama proses komputasi berlangsung. Dua konsep kunci yang sangat penting dalam menganalisis dan mengendalikan akurasi perhitungan numerik adalah stabilitas algoritma dan propagasi kesalahan (*error propagation*). Memahami keduanya sangat penting untuk menghindari hasil perhitungan yang tidak akurat atau bahkan menyesatkan dalam aplikasi sains, teknik, maupun keuangan.

1. Jenis-Jenis Kesalahan dalam Komputasi

Pada komputasi numerik, setiap proses perhitungan tidak terlepas dari berbagai bentuk kesalahan (*error*) yang dapat memengaruhi akurasi hasil. Pemahaman terhadap jenis-jenis kesalahan ini sangat penting agar pengembang algoritma dan praktisi komputasi dapat mengambil langkah-langkah korektif untuk meminimalkan dampaknya. Secara umum, kesalahan dalam komputasi terbagi menjadi tiga kategori utama: kesalahan pembulatan (*round-off error*), kesalahan pemotongan (*truncation error*), dan kesalahan input atau data (*input error*).

Kesalahan pembulatan terjadi karena keterbatasan representasi angka dalam komputer. Komputer menggunakan sistem *floating point* dengan jumlah bit terbatas, sehingga tidak semua bilangan desimal dapat direpresentasikan secara eksak. Misalnya, angka 0.1 dalam sistem desimal tidak dapat ditulis secara tepat dalam biner, sehingga terjadi pembulatan ke angka terdekat. Ketika perhitungan dilakukan berulang-ulang, kesalahan kecil ini bisa terakumulasi dan memengaruhi hasil akhir, terutama pada algoritma yang sensitif secara numerik.

Kesalahan pemotongan muncul ketika pendekatan matematis digunakan untuk menyelesaikan permasalahan yang tidak bisa dihitung secara eksak. Contohnya adalah penggunaan metode numerik seperti deret Taylor, metode Euler, atau integrasi numerik. Dalam metode ini, hanya sebagian dari istilah yang dihitung, sementara sisanya dipotong (*truncated*), sehingga menghasilkan deviasi dari nilai sebenarnya. Sementara itu, kesalahan input atau kesalahan data timbul dari ketidaktepatan data awal yang dimasukkan ke dalam sistem, misalnya hasil pengukuran yang tidak akurat atau data yang sudah mengalami proses konversi. Kesalahan jenis ini sangat bergantung pada konteks aplikasi, tetapi tetap dapat merambat melalui algoritma dan menyebabkan hasil akhir yang menyesatkan jika tidak dikendalikan.

2. Propagasi Kesalahan

Propagasi kesalahan adalah fenomena penting dalam komputasi numerik yang menggambarkan bagaimana kesalahan kecil pada data awal atau hasil perhitungan dapat menyebar dan membesar seiring berjalannya proses komputasi. Dalam praktiknya, hampir semua perhitungan dalam komputer melibatkan kesalahan pembulatan (*round-off*) akibat representasi *floating point* yang terbatas, serta kesalahan pemotongan (*truncation*) dalam penggunaan metode numerik. Propagasi

kesalahan menjadi krusial karena akumulasi dari kesalahan-kesalahan kecil ini dapat menyebabkan hasil akhir yang jauh menyimpang dari nilai yang seharusnya, terutama pada algoritma yang bersifat numerik tidak stabil.

Fenomena ini sering terjadi dalam operasi matematika yang melibatkan angka-angka dengan nilai yang sangat berdekatan, seperti dalam kasus pengurangan dua bilangan hampir sama. Salah satu contoh klasik adalah perhitungan $\sqrt{x^2 + 1} - 1$ untuk nilai x yang sangat kecil. Proses pengurangan ini dapat menghilangkan informasi penting dari angka yang tersimpan, fenomena ini dikenal sebagai *catastrophic cancellation*.

Propagasi kesalahan juga terjadi dalam metode iteratif seperti pada penyelesaian sistem persamaan linear, persamaan diferensial, atau perhitungan akar fungsi. Jika algoritma yang digunakan tidak stabil, maka kesalahan pada satu iterasi dapat diperkuat pada iterasi berikutnya, sehingga kesalahan total menjadi tidak terkendali. Hal ini diperparah jika masalah yang diselesaikan bersifat *ill-conditioned*, yaitu masalah di mana sedikit perubahan pada input menghasilkan perubahan besar pada output. Untuk mengatasi propagasi kesalahan, strategi numerik seperti normalisasi data, penggunaan metode numerik stabil, transformasi aljabar untuk menghindari pengurangan kritis, serta penggunaan kendali kesalahan (*error control*) dalam metode iteratif sering digunakan. Pemahaman tentang bagaimana dan kapan kesalahan tersebar adalah kunci dalam merancang algoritma yang handal dan memastikan akurasi hasil dalam aplikasi sains, teknik, maupun keuangan.

3. Stabilitas Algoritma

Stabilitas algoritma adalah konsep penting dalam komputasi numerik yang mengacu pada sejauh mana suatu algoritma dapat mengendalikan atau membatasi dampak kesalahan kecil selama proses perhitungan. Dalam konteks ini, kesalahan yang dimaksud bisa berasal dari pembulatan angka akibat keterbatasan representasi *floating point*, kesalahan pemotongan dalam metode numerik, maupun kesalahan input dari data yang tidak presisi. Algoritma yang stabil adalah algoritma yang mampu menghasilkan hasil akhir yang mendekati solusi sebenarnya meskipun terdapat gangguan kecil atau kesalahan pada data atau selama perhitungan berlangsung. Sebaliknya, algoritma yang tidak stabil akan

memperbesar kesalahan ini sehingga hasil akhirnya menjadi tidak dapat dipercaya.

Stabilitas sangat berkaitan dengan bagaimana kesalahan terpropagasi atau menyebar selama serangkaian langkah perhitungan. Sebagai contoh, dalam metode eliminasi Gauss untuk menyelesaikan sistem persamaan linear, pembagian oleh angka yang sangat kecil dapat menyebabkan hasil yang sangat melenceng karena pembulatan yang ekstrem. Tanpa teknik stabilisasi seperti pivoting (penukaran baris untuk memaksimalkan elemen pivot), metode ini menjadi tidak stabil secara numerik. Inilah sebabnya metode partial pivoting atau scaled partial pivoting banyak digunakan dalam praktik untuk menjaga kestabilan hasil.

Stabilitas juga menjadi isu kritis dalam metode numerik yang digunakan untuk menyelesaikan persamaan diferensial, seperti metode Euler atau Runge-Kutta. Sebagai contoh, metode Euler eksplisit cenderung tidak stabil jika digunakan pada sistem dengan dinamika cepat atau dengan langkah waktu (*step size*) yang besar. Ketidakstabilan ini menyebabkan nilai solusi menyimpang jauh dari solusi eksak, bahkan bisa menjadi tak hingga. Oleh karena itu, penting dilakukan analisis stabilitas terhadap metode numerik, termasuk dengan mengevaluasi region of stability atau batas nilai langkah yang masih menghasilkan solusi stabil.

C. Operasi Aritmetika dan Pembulatan dalam Mesin

Pada sistem komputasi modern, semua operasi numerik yang dilakukan komputer seperti penjumlahan, pengurangan, perkalian, dan pembagian dijalankan oleh unit pemroses (CPU) dalam bentuk operasi aritmetika biner. Namun, keterbatasan dalam representasi angka, khususnya angka real (pecahan), membuat hasil dari operasi ini sering kali tidak presisi sempurna. Oleh karena itu, penting untuk memahami operasi aritmetika dalam mesin dan bagaimana pembulatan (*rounding*) diterapkan sebagai bagian dari proses ini.

1. Representasi Bilangan *Floating point*

Representasi bilangan *floating point* merupakan cara standar yang digunakan komputer untuk menyimpan dan memanipulasi bilangan real (pecahan), terutama bilangan yang sangat besar atau sangat kecil.

Sistem ini mengadopsi prinsip notasi ilmiah, di mana sebuah bilangan dinyatakan dalam bentuk $\pm m \times b^e$, dengan mmm sebagai mantissa (*significand*), bbb sebagai basis (biasanya 2 dalam sistem komputer), dan eee sebagai eksponen. Untuk menjamin keseragaman dan interoperabilitas antar sistem komputasi, representasi ini dikendalikan oleh standar IEEE 754, yang paling umum digunakan di hampir semua perangkat keras dan bahasa pemrograman saat ini.

Pada standar IEEE 754, terdapat dua format utama: *single precision* (32-bit) dan *double precision* (64-bit). Untuk *single precision,* satu bilangan *floating point* terdiri dari 1 bit tanda (*sign bit*), 8 bit eksponen dengan bias 127, dan 23 bit mantissa. Sedangkan dalam double precision, digunakan 1 bit tanda, 11 bit eksponen dengan bias 1023, dan 52 bit mantissa. Nilai eksponen yang disimpan sebenarnya adalah hasil penjumlahan eksponen aktual dengan nilai bias, yang memungkinkan penyimpanan bilangan positif dan negatif secara efisien.

Salah satu fitur penting dari sistem ini adalah normalisasi, di mana angka disimpan dalam bentuk sedemikian rupa sehingga digit paling signifikan dari mantissa adalah bukan nol (kecuali untuk nol atau bilangan denormal). Proses normalisasi ini memastikan bahwa presisi maksimum dimanfaatkan dalam keterbatasan bit yang tersedia. Namun demikian, karena panjang mantissa terbatas, banyak bilangan desimal yang tidak bisa direpresentasikan secara eksak (misalnya 0.1), sehingga muncul kesalahan pembulatan (*round-off error*) dalam perhitungan.

Representasi *floating point* memungkinkan komputer untuk menangani perhitungan ilmiah dengan skala luas, namun pengguna harus berhati-hati terhadap akumulasi kesalahan, underflow, *overflow*, dan fenomena seperti cancellation yang bisa muncul akibat keterbatasan presisi. Oleh karena itu, pemahaman menyeluruh tentang representasi ini menjadi fondasi penting dalam desain algoritma numerik yang akurat dan stabil.

2. Operasi Aritmetika dalam Mesin

Operasi aritmetika dalam mesin merupakan proses dasar yang dilakukan oleh unit pemroses (CPU atau FPU) untuk menyelesaikan perhitungan matematis seperti penjumlahan, pengurangan, perkalian, dan pembagian. Berbeda dengan operasi manual pada manusia, komputer melakukan semua operasi tersebut dalam bentuk biner menggunakan sistem representasi *floating point*, sebagaimana diatur

dalam standar IEEE 754. Proses ini sangat kompleks karena melibatkan normalisasi, penyelarasan eksponen, manipulasi bit-bit mantissa, serta pembulatan akhir agar hasil sesuai dengan kapasitas penyimpanan bit yang tersedia.

Pada penjumlahan dan pengurangan *floating point*, langkah awal yang dilakukan adalah penyamaan eksponen. Operand dengan eksponen lebih kecil akan disesuaikan dengan menggeser mantissanya ke kanan, sehingga eksponennya cocok dengan operand lain. Setelah eksponen disamakan, barulah operasi mantissa dilakukan. Hasilnya kemudian dinormalisasi jika hasil memiliki digit paling signifikan yang bukan di posisi standar, maka mantissa digeser dan eksponen disesuaikan. Terakhir, dilakukan pembulatan (*rounding*) ke dalam format bit mantissa yang ditentukan (misalnya 23 bit untuk *single precision*), karena hasil sebenarnya sering kali tidak bisa disimpan secara eksak.

Perkalian dan pembagian *floating point* memiliki mekanisme berbeda. Eksponen operand dijumlahkan (untuk perkalian) atau dikurangkan (untuk pembagian), sementara mantissa dikalikan atau dibagi. Proses ini pun diakhiri dengan normalisasi dan pembulatan. Seluruh langkah ini membuat operasi *floating point* lebih mahal secara komputasi dibanding operasi integer, dan lebih rentan terhadap kesalahan pembulatan (*round-off error*). Masalah juga bisa muncul jika terjadi *overflow* (nilai melebihi batas maksimum eksponen) atau underflow (nilai terlalu kecil untuk direpresentasikan). Karena sifat aritmetika *floating point* yang tidak sepenuhnya asosiatif atau distributif, hasil operasi bisa berbeda tergantung urutan kalkulasi. Oleh karena itu, dalam pemrograman numerik, sangat penting untuk menyusun operasi secara hati-hati dan memilih algoritma yang stabil secara numerik guna meminimalkan akumulasi kesalahan dan menjamin keandalan hasil perhitungan.

3. Pembulatan (*Rounding*)

Pembulatan (*rounding*) adalah proses penting dalam komputasi numerik yang terjadi ketika suatu bilangan real tidak dapat direpresentasikan secara eksak dalam format biner *floating point*, sehingga harus disesuaikan ke nilai terdekat yang bisa diwakili oleh komputer. Hal ini disebabkan oleh keterbatasan jumlah bit yang tersedia untuk menyimpan angka, khususnya pada bagian mantissa. Sebagai contoh, dalam format IEEE 754 single precision, hanya tersedia 23 bit

untuk mantissa, sehingga banyak bilangan desimal seperti 0.1 atau 1/3, tidak dapat disimpan secara tepat. Akibatnya, proses pembulatan tidak hanya tak terhindarkan, tetapi juga sangat berpengaruh terhadap akurasi hasil perhitungan.

Menurut (Higham, 2002), pembulatan merupakan sumber utama dari kesalahan pembulatan (*round-off error*), yaitu perbedaan antara nilai aktual dan nilai yang disimpan atau dihitung oleh komputer. Dalam standar IEEE 754, terdapat beberapa mode pembulatan yang diimplementasikan untuk mengatur cara komputer menentukan nilai terdekat, yaitu: *round to nearest* (*default*), *round toward zero*, *round toward +infinity*, dan *round toward –infinity*. Mode round to nearest, ties to even adalah yang paling umum, karena secara statistik dapat meminimalkan akumulasi kesalahan dalam perhitungan berulang.

Proses pembulatan terjadi setiap kali hasil operasi aritmetika tidak muat dalam mantissa yang tersedia. Misalnya, saat dua bilangan dikalikan dan menghasilkan mantissa yang lebih panjang dari kapasitas, komputer akan memotong digit-digit tak signifikan dan menyimpan hasil yang dibulatkan. Jika proses ini terjadi secara berulang dalam algoritma yang panjang atau iteratif, kesalahan pembulatan dapat terakumulasi dan berdampak signifikan terhadap hasil akhir, terutama dalam algoritma yang tidak stabil secara numerik. Untuk mengurangi efek negatif pembulatan, praktisi komputasi numerik harus memahami sifat pembulatan dalam mesin dan memilih strategi yang sesuai. Hal ini termasuk menyusun ulang ekspresi matematis, menggunakan presisi lebih tinggi jika diperlukan, serta menghindari operasi seperti pengurangan dua angka yang hampir sama yang rentan terhadap hilangnya digit signifikan akibat pembulatan.

D. Standard IEEE 754

Di dunia komputasi, angka real (pecahan) berperan an penting, baik dalam aplikasi ilmiah, teknik, statistik, maupun grafika. Namun, representasi angka-angka ini dalam komputer tidaklah sesederhana penulisan desimal. Karena komputer hanya mengenal angka dalam bentuk biner dan memiliki keterbatasan memori, dibutuhkan sistem representasi numerik yang efisien, konsisten, dan mampu menangani angka sangat besar maupun sangat kecil. Untuk menjawab kebutuhan

tersebut, diperkenalkanlah standar IEEE 754 yang hingga kini menjadi patokan global dalam representasi dan perhitungan *floating point*.

Menurut (IEEE Standards Association, 2008), IEEE 754 adalah standar yang dikembangkan oleh *Institute of Electrical and Electronics Engineers* (IEEE) dan pertama kali diperkenalkan pada tahun 1985. Standar ini mendefinisikan format penyimpanan, aturan pembulatan, penanganan nilai khusus (seperti NaN dan Infinity), serta metode operasi aritmetika *floating point* yang konsisten di seluruh arsitektur komputer dan bahasa pemrograman. Sebelum adanya IEEE 754, produsen perangkat keras memiliki implementasi *floating point* masing-masing yang berbeda-beda, sehingga menyebabkan inkonsistensi hasil perhitungan numerik antar sistem. IEEE 754 hadir untuk menyatukan standar ini dan memastikan interoperabilitas serta akurasi komputasi di berbagai platform dan aplikasi.

1. Struktur Representasi *Floating point*

Struktur representasi *floating point* dalam komputer adalah cara menyimpan bilangan real menggunakan format biner dengan tiga komponen utama: bit tanda (*sign bit*), eksponen (*exponent*), dan fraksi atau mantissa (*fraction/mantissa*). Standar representasi yang digunakan secara luas dalam industri dan akademik adalah IEEE 754, yang menjamin konsistensi, efisiensi, dan interoperabilitas dalam perhitungan numerik di berbagai sistem perangkat keras dan perangkat lunak. *Floating point* digunakan karena mampu mewakili rentang angka yang sangat luas, baik yang sangat kecil mendekati nol maupun yang sangat besar, tanpa memerlukan format data yang terlalu besar secara fisik.

Pada format *single precision* (32 bit), bilangan *floating point* terdiri dari 1 bit tanda, 8 bit eksponen, dan 23 bit mantissa. Bit tanda menunjukkan apakah bilangan positif (0) atau negatif (1). Eksponen digunakan untuk mengalikan basis dua sehingga bilangan bisa dinormalisasi, dan disimpan dalam format "*biased exponent*" dengan bias sebesar 127. Artinya, nilai eksponen aktual diperoleh dengan mengurangkan nilai yang disimpan dengan 127. Sedangkan bagian mantissa menyimpan angka-angka setelah titik desimal, dan dalam representasi normalisasi selalu diasumsikan memiliki bit tersembunyi (*implicit bit*) yaitu angka 1 di depan, sehingga bagian mantissa sebenarnya adalah 1.x… dalam basis biner. Pada *double precision* (64 bit), strukturnya terdiri dari 1 bit tanda, 11 bit eksponen, dan 52 bit

mantissa, dengan bias eksponen sebesar 1023. Dengan kapasitas mantissa yang lebih besar, *double precision* memungkinkan representasi bilangan yang jauh lebih presisi dan mengurangi kemungkinan kesalahan pembulatan (*round-off error*) dalam operasi aritmetika.

Salah satu keunggulan struktur *floating point* ini adalah kemampuannya untuk menangani bilangan desimal sangat besar atau sangat kecil secara efisien, yang tidak dapat dilakukan oleh representasi integer biasa. Namun, struktur ini juga memperkenalkan tantangan, seperti ketidakakuratan representasi bilangan desimal tertentu (misalnya 0.1) dan efek propagasi kesalahan dalam operasi berulang. Oleh karena itu, pemahaman terhadap struktur ini sangat penting dalam merancang algoritma numerik yang stabil dan efisien di berbagai aplikasi sains, teknik, dan keuangan.

2. Nilai Khusus dalam IEEE 754

Pada standar IEEE 754 untuk representasi bilangan *floating point*, tidak semua pola bit digunakan untuk menyatakan bilangan real biasa. Sebagian pola disediakan untuk merepresentasikan nilai-nilai khusus yang memiliki makna penting dalam komputasi numerik, terutama dalam penanganan kondisi ekstrem seperti pembagian nol, *overflow, underflow*, atau operasi tak terdefinisi. Nilai-nilai khusus ini mencakup: Nol positif/negatif (±0), Tak hingga (±∞), NaN (*Not a Number*), dan bilangan denormal atau subnormal.

Pertama, ±0 menunjukkan bahwa angka nol dapat disimpan dengan tanda positif atau negatif. Meskipun secara matematis tidak berbeda, dalam komputasi ±0 digunakan untuk mempertahankan arah pendekatan limit atau derivatif, yang penting dalam analisis numerik dan kalkulus. Contohnya, hasil dari $-1/\infty$ dapat berupa -0, menunjukkan bahwa pendekatan berasal dari arah negatif.

Kedua, tak hingga (*positive/negative infinity*) muncul saat hasil perhitungan melebihi batas representasi eksponen tertinggi (*overflow*), seperti ketika membagi angka besar dengan angka sangat kecil atau pembagian angka bukan nol dengan nol. Dalam IEEE 754, ini diwakili dengan eksponen maksimum (semua bit eksponen = 1) dan mantissa = 0. Hasil operasi terhadap tak hingga mengikuti aturan aljabar, misalnya $a+\infty=\infty$, tetapi operasi seperti $\infty-\infty$ menghasilkan NaN.

Ketiga, NaN (*Not a Number*) digunakan untuk menunjukkan hasil dari operasi yang tidak valid secara matematis, seperti $\sqrt{-1,0/0}$, atau $\infty-\infty$. NaN memiliki eksponen semua bit 1, seperti tak hingga, tetapi mantissanya tidak nol. Terdapat dua jenis NaN: *quiet* NaN (qNaN) yang terus propagasi dalam perhitungan, dan *signaling* NaN (sNaN) yang dimaksudkan untuk menghasilkan error jika digunakan tanpa penanganan.

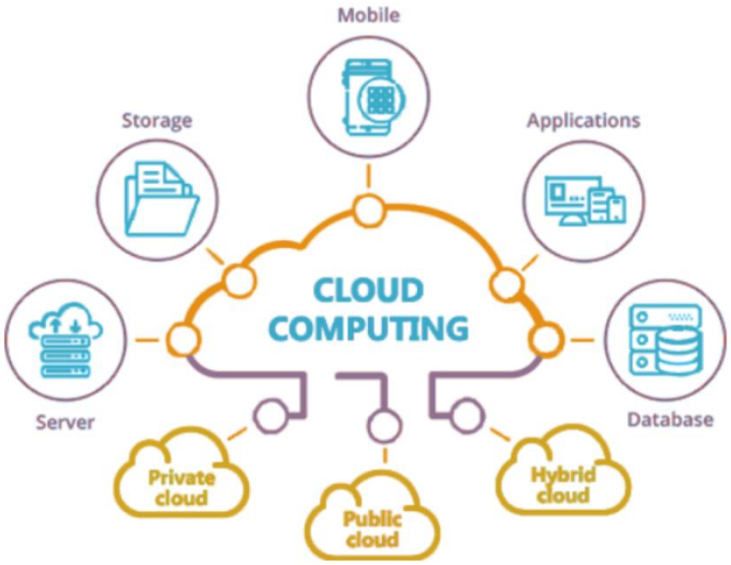
Keempat, bilangan denormal (subnormal) digunakan saat hasil bilangan sangat kecil sehingga tidak lagi bisa dinormalisasi dalam format standar. Dalam kasus ini, angka disimpan dengan eksponen nol (bukan eksponen bias), dan tanpa bit 1 tersembunyi pada mantissa. Nilai ini memperluas rentang representasi menuju nol dan memungkinkan *graceful underflow*, yakni hasil perhitungan tidak langsung menjadi nol, melainkan bertahap mendekati nol.

3. Evolusi dan Versi Terbaru

Standar IEEE 754 telah mengalami beberapa kali evolusi sejak pertama kali diperkenalkan pada tahun 1985, seiring dengan berkembangnya kebutuhan komputasi numerik di berbagai bidang seperti sains, teknik, keuangan, hingga kecerdasan buatan. Versi awal IEEE 754-1985 berfokus pada definisi representasi *floating point* untuk dua format utama: *single precision* (32-bit) dan *double precision* (64-bit). Standar ini memperkenalkan konsep penting seperti pembulatan standar (*rounding modes*), nilai khusus (NaN dan Infinity), serta pengaturan *overflow* dan underflow, yang hingga kini menjadi dasar utama komputasi numerik.

Seiring meningkatnya kompleksitas dan skala komputasi modern, IEEE kemudian memperbarui standar ini melalui versi IEEE 754-2008. Versi ini membawa beberapa pembaruan signifikan, seperti penambahan format baru, termasuk *quadruple precision* (128-bit) dan decimal *floating point*, yang dirancang untuk aplikasi yang memerlukan akurasi sangat tinggi atau manipulasi data desimal secara presisi, seperti dalam sistem keuangan. Versi ini juga menetapkan operasi baru seperti *fused multiply-add* (FMA) yang menggabungkan operasi perkalian dan penjumlahan dalam satu langkah untuk mengurangi kesalahan pembulatan, serta memperluas aturan konversi antar format dan representasi bilangan kompleks.

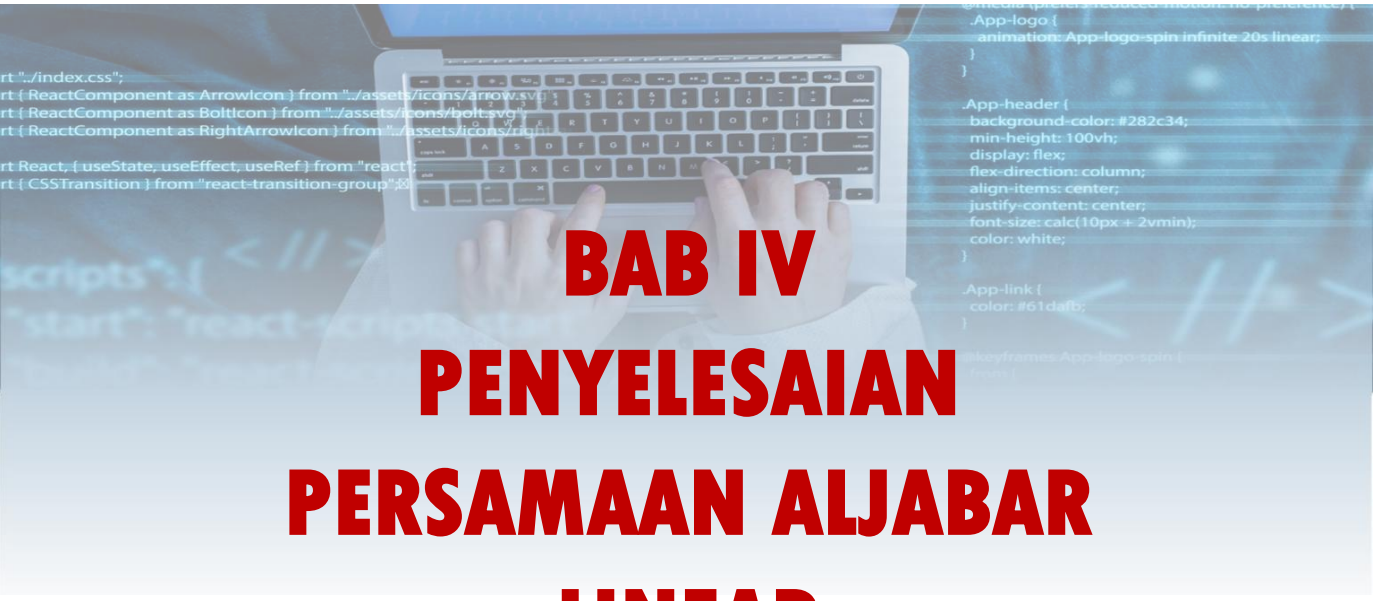
Gambar 3. *Cloud Computing*



Sumber: *Btech*

Terbaru, versi IEEE 754-2019 memperbaiki dan menyempurnakan standar sebelumnya, dengan tujuan meningkatkan kejelasan implementasi dan interoperabilitas. Beberapa perbaikan yang dibawa termasuk definisi lebih eksplisit tentang perilaku nilai NaN, pelabelan tipe minimal (minimal *floating-point types*) untuk perangkat keras dengan sumber daya terbatas, serta penyempurnaan dokumentasi operasi pembulatan, konversi, dan penanganan pengecualian. IEEE 754-2019 juga menegaskan kembali pentingnya akurasi, determinisme, dan portabilitas dalam lingkungan komputasi yang terus berubah, seperti cloud computing dan komputasi paralel.

Evolusi standar IEEE 754 menunjukkan bagaimana komunitas ilmiah dan teknis merespons tantangan komputasi numerik secara progresif. Dengan menetapkan aturan yang konsisten untuk semua jenis sistem dan platform, standar ini memungkinkan pengembang dan peneliti untuk membangun algoritma yang stabil, dapat direproduksi, dan andal, serta mampu menangani kompleksitas perhitungan skala besar dengan keakuratan tinggi.



BAB IV
PENYELESAIAN PERSAMAAN ALJABAR LINEAR

Persamaan aljabar linear merupakan salah satu fondasi utama dalam bidang matematika terapan dan komputasi numerik. Penyelesaiannya tidak hanya penting dalam ranah teori, tetapi juga memiliki aplikasi luas dalam berbagai disiplin ilmu, seperti fisika, teknik, ekonomi, dan ilmu komputer. Dalam praktiknya, sistem persamaan linear sering kali muncul dalam bentuk matriks dan vektor, serta membutuhkan pendekatan numerik yang efisien untuk mendapatkan solusi yang akurat, terutama ketika berhadapan dengan sistem berskala besar atau yang tidak dapat diselesaikan secara analitik. Buku atau materi ini disusun untuk memberikan pemahaman menyeluruh tentang teknik penyelesaian sistem persamaan linear, mulai dari metode eliminasi Gauss, dekomposisi matriks seperti LU decomposition, hingga pendekatan iteratif seperti metode Jacobi dan Gauss-Seidel. Di samping penjelasan teoritis, pembahasan juga dilengkapi dengan implementasi algoritma menggunakan bahasa pemrograman modern, sehingga pembaca dapat secara langsung menerapkan konsep yang dipelajari dalam pemecahan masalah nyata.

A. Sistem Persamaan Linear dan Matriks Koefisien

Menurut Anton & Rorres (2010) dalam Elementary Linear Algebra, sistem persamaan linear adalah sekumpulan persamaan linear

yang memiliki satu atau lebih variabel yang saling berkaitan. Dalam bentuk umum, sistem ini dapat dituliskan sebagai berikut:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \vdots \\ a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mn}x_n = b_m \end{cases}$$

Sistem seperti ini disebut sistem persamaan linear dengan *m* persamaan dan *n* variabel. Koefisien a_{ij} menyatakan konstanta pengali variabel ke-j dalam persamaan ke-i, sedangkan b_i merupakan konstanta pada ruas kanan.

1. Representasi Matriks Koefisien

Representasi matriks koefisien merupakan pendekatan sistematis untuk menuliskan sistem persamaan linear dalam bentuk yang lebih ringkas dan terstruktur. Dalam sistem persamaan linear, setiap persamaan melibatkan sejumlah variabel dengan koefisien tertentu. Jika sistem tersebut memiliki mmm persamaan dan nnn variabel, maka semua koefisien dapat disusun dalam sebuah matriks berukuran m×n, yang dikenal sebagai matriks koefisien. Misalnya, sistem tiga persamaan dengan tiga variabel:

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + a_{13}x_3 = b_1 \\ a_{21}x_1 + a_{22}x_2 + a_{23}x_3 = b_2 \\ a_{31}x_1 + a_{32}x_2 + a_{33}x_3 = b_3 \end{cases}$$

dapat direpresentasikan menjadi:

$$A = \begin{bmatrix} a_{11} & a_{12} & a_{13} \\ a_{21} & a_{22} & a_{23} \\ a_{31} & a_{32} & a_{33} \end{bmatrix}, \quad \mathbf{x} = \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix}, \quad \mathbf{b} = \begin{bmatrix} b_1 \\ b_2 \\ b_3 \end{bmatrix}$$

Sehingga, sistem tersebut ditulis sebagai Ax=b. Representasi ini memiliki keunggulan dalam efisiensi notasi, kemudahan manipulasi matematis, dan sangat sesuai untuk diimplementasikan secara

komputasi. Dalam konteks algoritma numerik, operasi terhadap sistem linear seperti eliminasi Gauss, dekomposisi matriks, atau metode iteratif dapat dilakukan dengan jauh lebih mudah menggunakan bentuk matriks ini. Selain itu, dengan menambahkan vektor konstanta b sebagai kolom terakhir dari matriks koefisien, diperoleh matriks *augmented* $[A|b]$, yang sangat bermanfaat dalam menyelesaikan sistem dengan metode operasi baris elementer. Oleh karena itu, representasi matriks koefisien bukan hanya alat bantu notasi, tetapi merupakan dasar penting dalam teori dan aplikasi sistem persamaan linear.

2. Matriks *Augmented* dan Transformasi Baris

Matriks *augmented* adalah representasi matriks gabungan yang menyatukan matriks koefisien dari sistem persamaan linear dengan vektor konstanta di sisi kanan persamaan. Bentuk ini ditulis sebagai $[A|b]$, di mana A adalah matriks koefisien berukuran m×n dan b adalah vektor kolom dari konstanta ruas kanan berukuran m×1. Tujuan dari matriks *augmented* adalah untuk memfasilitasi penyelesaian sistem linear melalui manipulasi baris secara langsung, tanpa perlu menuliskan ulang seluruh sistem persamaan dalam bentuk aljabar konvensional. Representasi ini sangat efektif dalam metode numerik seperti eliminasi Gauss dan Gauss-Jordan.

Proses penyederhanaan matriks *augmented* dilakukan melalui transformasi baris elementer, yang terdiri dari tiga jenis: (1) menukar dua baris, (2) mengalikan suatu baris dengan skalar tak nol, dan (3) menambahkan kelipatan suatu baris ke baris lainnya. Transformasi ini bertujuan mengubah bentuk matriks *augmented* menjadi eselon baris atau bahkan eselon baris tereduksi, sehingga solusi sistem dapat diperoleh dengan mudah melalui substitusi mundur atau langsung terbaca dari hasil akhir.

Sebagai contoh, sistem dua persamaan linear yang direpresentasikan sebagai matriks *augmented*:

$$\left[\begin{array}{cc|c} 2 & 3 & 8 \\ 4 & -1 & 2 \end{array}\right]$$

dapat disederhanakan menggunakan operasi baris hingga mencapai bentuk:

$$\left[\begin{array}{cc|c} 1 & 0 & x \\ 0 & 1 & y \end{array}\right]$$

yang secara langsung menyatakan solusi dari sistem. Menurut Lay (2012), transformasi baris tidak mengubah solusi dari sistem, sehingga semua bentuk yang ekuivalen baris tetap merepresentasikan sistem persamaan yang sama. Oleh karena itu, penggunaan matriks *augmented* dan transformasi baris menjadi pendekatan yang sangat kuat dan fundamental dalam penyelesaian sistem linear secara manual maupun komputasional.

3. Solusi Sistem Persamaan Linear

Solusi sistem persamaan linear merujuk pada himpunan nilai variabel yang memenuhi semua persamaan dalam sistem secara simultan. Menurut Strang (2016) dalam *Introduction to Linear Algebra,* sistem linear dapat memiliki tiga kemungkinan solusi: (1) satu solusi unik, (2) tak hingga banyak solusi, atau (3) tidak memiliki solusi sama sekali. Jenis solusi yang mungkin sangat bergantung pada hubungan antara jumlah persamaan, jumlah variabel, dan sifat dari matriks koefisien.

Solusi unik terjadi apabila sistem terdiri dari nnn persamaan independen dengan nnn variabel dan determinan matriks koefisien tidak nol (dalam kasus matriks persegi). Solusi tak hingga muncul jika terdapat redundansi atau ketergantungan linier antar persamaan, sehingga sistem memiliki lebih sedikit persamaan efektif dibanding variabel umumnya terjadi dalam sistem underdetermined. Sementara itu, sistem dikatakan tidak konsisten atau tidak memiliki solusi jika terdapat kontradiksi antar persamaan.

Untuk menentukan jenis solusi, konsep rank sangat penting. Rank adalah jumlah maksimum baris atau kolom linear independen dalam matriks. Berdasarkan Teorema Rouché–Capelli, solusi sistem ditentukan dengan membandingkan rank matriks koefisien A dan rank matriks *augmented* $[A|b]$. Jika rank-nya sama dan setara dengan jumlah variabel, sistem memiliki solusi unik. Jika rank sama tetapi kurang dari jumlah variabel, terdapat tak hingga solusi. Jika rank berbeda, sistem tidak memiliki solusi.

Pemahaman tentang jenis solusi sangat penting dalam penerapan praktis, seperti dalam analisis struktur teknik sipil, pemodelan ekonomi, atau sistem pengendalian dalam teknik elektro. Tanpa mengetahui sifat solusi, penggunaan algoritma komputasi bisa menghasilkan hasil yang salah atau tidak bermakna.

4. Interpretasi Geometris

Interpretasi geometris dari sistem persamaan linear memberikan pemahaman visual mengenai bagaimana solusi dari sistem tersebut terbentuk. Menurut Anton & Rorres (2010) dalam *Elementary Linear Algebra,* setiap persamaan linear dalam dua variabel dapat direpresentasikan sebagai sebuah garis lurus di bidang dua dimensi (2D), sementara dalam tiga variabel akan direpresentasikan sebagai bidang dalam ruang tiga dimensi (3D). Titik perpotongan dari garis atau bidang ini menjadi representasi dari solusi sistem.

Pada ruang dua dimensi, misalnya, sistem dua persamaan linear dapat divisualisasikan sebagai dua garis. Jika garis-garis tersebut berpotongan di satu titik, maka sistem memiliki satu solusi unik, yaitu koordinat titik perpotongan tersebut. Jika kedua garis saling berimpit, artinya merepresentasikan persamaan yang sama dan sistem memiliki tak hingga banyak solusi. Namun, jika garis-garis tersebut sejajar namun tidak berpotongan, maka sistem tidak memiliki solusi, yang menandakan bahwa sistem tersebut inkonsisten.

Pada ruang tiga dimensi, setiap persamaan linear tiga variabel mewakili sebuah bidang. Tiga bidang dapat berpotongan di satu titik (solusi unik), sepanjang garis (tak hingga solusi), atau tidak berpotongan sama sekali (tidak ada solusi). Misalnya, dua bidang yang sejajar atau tiga bidang yang membentuk prisma tanpa titik temu merupakan sistem yang tidak konsisten. Interpretasi ini juga berlaku di ruang berdimensi lebih tinggi secara abstrak, meskipun tidak mudah divisualisasikan. Konsep vektor, ruang vektor, dan subruang membantu memahami posisi relatif antar persamaan dalam konteks geometris. Dengan demikian, interpretasi geometris bukan hanya berguna untuk visualisasi, tetapi juga memberikan intuisi mendalam tentang kondisi eksistensi dan keunikan solusi, serta hubungan linier antar persamaan dalam sistem.

B. Eliminasi Gauss dan Pivoting

Menurut Burden dan Faires (2011) dalam *Numerical Analysis*, metode eliminasi Gauss (*Gaussian Elimination*) adalah salah satu algoritma dasar dalam penyelesaian sistem persamaan linear. Metode ini bekerja dengan mengubah sistem persamaan menjadi bentuk segitiga atas (*upper triangular matrix*) melalui operasi baris elementer. Dengan bentuk ini, solusi sistem linear dapat diperoleh secara efisien melalui teknik substitusi mundur (*back substitution*).

$$\begin{cases} a_{11}x_1 + a_{12}x_2 + \cdots + a_{1n}x_n = b_1 \\ a_{21}x_1 + a_{22}x_2 + \cdots + a_{2n}x_n = b_2 \\ \vdots \\ a_{n1}x_1 + a_{n2}x_2 + \cdots + a_{nn}x_n = b_n \end{cases}$$

dapat direpresentasikan dalam bentuk matriks *augmented* $[A|b]$. Tujuan eliminasi Gauss adalah untuk menghilangkan elemen-elemen di bawah diagonal utama agar sistem menjadi bentuk upper triangular, yaitu hanya elemen diagonal dan elemen di atasnya yang bukan nol.

1. Operasi Baris Elementer

Operasi baris elementer adalah tiga jenis transformasi dasar yang digunakan untuk memodifikasi baris-baris dalam sebuah matriks tanpa mengubah solusi dari sistem persamaan linear yang direpresentasikannya. Menurut Lay (2012) dalam *Linear Algebra and Its Applications,* operasi baris elementer sangat penting dalam metode penyelesaian sistem linear seperti eliminasi Gauss, Gauss-Jordan, dan proses reduksi matriks ke bentuk eselon. Operasi ini memungkinkan kita untuk menyederhanakan sistem persamaan linear menjadi bentuk yang lebih mudah diselesaikan tanpa kehilangan karakteristik solusinya.

Tiga jenis operasi baris elementer adalah: (1) Pertukaran dua baris (*interchange*), (2) Perkalian baris dengan skalar tak nol (*scaling*), dan (3) Penjumlahan kelipatan suatu baris ke baris lain (*replacement*). Setiap operasi ini memiliki peran unik dalam proses manipulasi matriks.

Pertama, pertukaran dua baris digunakan ketika elemen pivot (elemen diagonal yang akan digunakan untuk mengeliminasi elemen di

bawahnya) adalah nol atau mendekati nol. Dalam kasus seperti itu, untuk menghindari pembagian dengan nol atau angka sangat kecil yang dapat menyebabkan ketidakstabilan numerik, baris tersebut ditukar dengan baris lain yang memiliki elemen pivot lebih besar secara nilai mutlak. Operasi ini sering digunakan dalam strategi partial pivoting, yang sangat penting dalam komputasi numerik.

Kedua, perkalian baris dengan skalar tak nol berguna untuk menyederhanakan elemen pivot menjadi satu (1), sehingga memudahkan eliminasi elemen lainnya. Misalnya, jika elemen pivot adalah 4, maka seluruh baris dapat dikalikan dengan $\frac{1}{4}$ agar pivot menjadi 1. Ini juga digunakan dalam metode Gauss-Jordan, di mana tujuan akhirnya adalah mencapai bentuk eselon baris tereduksi (*reduced row echelon form*), di mana semua elemen pivot bernilai 1 dan elemen-elemen di atas dan di bawah pivot bernilai nol. Ketiga, penjumlahan kelipatan suatu baris ke baris lain adalah operasi paling umum dalam proses eliminasi. Tujuannya adalah menghilangkan elemen tertentu di bawah atau di atas pivot agar tercapai struktur segitiga atas atau bentuk eselon. Proses ini dilakukan berulang hingga semua elemen di bawah (atau di atas) pivot menjadi nol.

Menurut Strang (2016) dalam *Introduction to Linear Algebra*, ketiga operasi baris ini secara matematis bersifat *reversible*, artinya setiap operasi memiliki operasi kebalikannya yang dapat mengembalikan matriks ke bentuk semula. Hal ini memastikan bahwa struktur sistem tetap terjaga dan solusi tetap valid. Karena itu, operasi baris elementer digunakan tidak hanya dalam penyelesaian sistem persamaan linear, tetapi juga dalam proses mencari invers matriks, menghitung determinan (secara tidak langsung), dan menemukan rank sebuah matriks.

Pada implementasi komputasi, operasi baris elementer diaplikasikan secara sistematis dan efisien. Misalnya, dalam algoritma eliminasi Gauss, baris pertama digunakan untuk menghilangkan elemen di kolom pertama pada baris-baris di bawahnya, kemudian baris kedua digunakan untuk mengeliminasi elemen di kolom kedua, dan seterusnya. Operasi-operasi ini juga menjadi dasar dalam algoritma pustaka numerik populer seperti LAPACK dan NumPy.

2. Pivoting

Pivoting adalah teknik penting dalam penyelesaian sistem persamaan linear yang digunakan untuk meningkatkan stabilitas numerik dan keakuratan hasil dalam metode eliminasi Gauss. Menurut Trefethen dan Bau (1997) dalam Numerical Linear Algebra, pivoting dilakukan dengan memilih elemen terbesar (secara nilai absolut) di kolom atau seluruh submatriks sebagai elemen pivot, lalu menukar baris (dan kadang kolom) untuk menempatkan elemen tersebut pada posisi utama diagonal. Tujuannya adalah untuk menghindari pembagian dengan angka yang sangat kecil atau nol, yang dapat menyebabkan kesalahan pembulatan yang besar dalam perhitungan numerik.

Pada konteks metode eliminasi Gauss, setiap langkah mengharuskan kita membagi elemen-elemen di bawah pivot dengan nilai pivot itu sendiri. Jika nilai pivot sangat kecil, pembagian tersebut akan menghasilkan bilangan besar yang rentan terhadap kesalahan pembulatan. Di sinilah pivoting menjadi penting. Dengan memilih elemen terbesar sebagai pivot, kita meminimalkan potensi kesalahan akibat keterbatasan presisi dalam komputasi *floating-point*.

Ada tiga jenis pivoting yang umum digunakan: *partial pivoting, complete pivoting,* dan *scaled pivoting. Partial pivoting*, yang paling umum dan efisien, melibatkan pencarian elemen terbesar di kolom pivot dan menukar baris yang bersangkutan ke posisi baris aktif saat ini. *Complete pivoting* lebih ekstrem, di mana pencarian dilakukan di seluruh submatriks dan baik baris maupun kolom dapat dipertukarkan. Sementara itu, *scaled pivoting* mempertimbangkan rasio antara elemen pivot dan elemen maksimum pada barisnya untuk mencegah kesalahan akibat perbedaan skala antar baris. Sebagai contoh, perhatikan sistem:

$$\begin{cases} 0.0003x + 3.0000y = 2.0001 \\ 1.0000x + 1.0000y = 2.0000 \end{cases}$$

Jika kita menggunakan baris pertama sebagai pivot tanpa melakukan pivoting, maka kita akan membagi dengan angka 0.0003, yang sangat kecil. Ini berpotensi menghasilkan kesalahan pembulatan besar. Namun, jika kita menerapkan partial pivoting dan menukar baris pertama dengan baris kedua, kita akan menggunakan 1.0000 sebagai pivot, sehingga perhitungan menjadi jauh lebih stabil dan akurat.

Menurut Golub dan Van Loan (2013) dalam Matrix Computations, penggunaan pivoting khususnya *partial pivoting* telah

menjadi standar dalam hampir semua implementasi algoritma penyelesaian sistem linear pada perangkat lunak numerik modern seperti MATLAB, LAPACK, dan NumPy. Hal ini karena partial pivoting menyediakan keseimbangan antara kestabilan numerik dan efisiensi komputasi. Dalam dunia nyata, stabilitas hasil perhitungan sangat penting, terutama dalam aplikasi teknik, simulasi ilmiah, dan pemrosesan data berskala besar. Tanpa pivoting, metode eliminasi Gauss dapat menghasilkan hasil yang sangat tidak akurat atau bahkan gagal menyelesaikan sistem. Oleh karena itu, pemahaman dan penerapan pivoting adalah aspek krusial dalam komputasi numerik modern.

C. Metode Iteratif: Jacobi dan Gauss-Seidel

Menurut Burden dan Faires (2011) dalam *Numerical Analysis*, metode iteratif merupakan pendekatan yang digunakan untuk menyelesaikan sistem persamaan linear, khususnya ketika sistem tersebut sangat besar atau memiliki struktur matriks koefisien yang jarang (*sparse*). Berbeda dengan metode langsung seperti eliminasi Gauss yang mencari solusi dalam jumlah langkah terbatas, metode iteratif menghasilkan serangkaian pendekatan yang mendekati solusi sejati secara bertahap. Dua metode iteratif klasik yang paling dikenal adalah metode Jacobi dan metode Gauss-Seidel, yang keduanya memiliki prinsip kerja yang relatif sederhana namun efektif.

Saad (2003) dalam *Iterative Methods for Sparse Linear Systems* menjelaskan bahwa dalam sistem berdimensi besar, metode langsung sering kali tidak praktis karena kompleksitas komputasi dan kebutuhan memori yang tinggi. Hal ini terutama berlaku pada sistem dengan matriks berukuran ribuan hingga jutaan baris dan kolom, seperti dalam simulasi numerik fluida atau analisis struktur teknik. Dalam konteks inilah metode iteratif menjadi solusi ideal karena hemat memori, mampu menangani matriks sparse, dan dapat dihentikan pada tingkat akurasi yang diinginkan.

1. Prinsip Dasar Metode Iteratif

Prinsip dasar metode iteratif dalam penyelesaian sistem persamaan linear adalah membentuk serangkaian pendekatan yang secara bertahap mendekati solusi yang benar dari sistem tersebut. Menurut Burden dan Faires (2011) dalam *Numerical Analysis*, metode

iteratif memulai prosesnya dengan sebuah tebakan awal terhadap nilai-nilai variabel, lalu melalui rumus perbaikan tertentu, menghasilkan solusi baru yang diharapkan semakin mendekati nilai sebenarnya. Proses ini diulang terus-menerus sampai kriteria konvergensi terpenuhi biasanya ditentukan oleh toleransi kesalahan yang sangat kecil atau jumlah iterasi maksimum.

Secara matematis, sistem linear Ax = b akan diubah menjadi bentuk rekursif $X^{(k:1)} = Gx^{(k)} + c, \text{dimana } X^{(k)}$ adalah pendekatan solusi pada iterasi ke-k, G adalah matriks transformasi iteratif, dan c adalah vektor tetap hasil transformasi dari A dan b. Tujuan dari iterasi ini adalah agar $X^{(k)}$ konvergen terhadap solusi sebenarnya x, yaitu saat limk → ∞ x(k) – x.

Keunggulan utama metode iteratif terletak pada efisiensinya dalam menangani sistem besar dan sparse, karena tidak memerlukan penyimpanan semua elemen matriks. Selain itu, pengguna memiliki fleksibilitas dalam mengatur presisi solusi sesuai kebutuhan aplikasi. Namun, konvergensi tidak selalu dijamin. Faktor seperti struktur matriks, kondisi awal, dan nilai eigen dari matriks iterasi sangat menentukan keberhasilan metode ini. Karena itu, analisis konvergensi seperti dominansi diagonal atau sifat positif-definit dari matriks sangat penting sebelum menerapkan metode iteratif secara praktis.

2. Metode Jacobi

Metode Jacobi adalah salah satu teknik iteratif paling dasar yang digunakan untuk menyelesaikan sistem persamaan linear, khususnya ketika sistem tersebut besar dan memiliki struktur matriks sparse. Menurut Burden dan Faires (2011) dalam *Numerical Analysis*, metode Jacobi bekerja dengan prinsip bahwa setiap variabel dalam sistem diselesaikan secara terpisah menggunakan nilai-nilai dari iterasi sebelumnya, tanpa segera memanfaatkan nilai yang baru dihitung dalam iterasi yang sama. Ini membuat metode Jacobi bersifat paralel secara alami, karena semua elemen solusi diperbarui secara bersamaan pada akhir setiap iterasi.

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j \neq i} a_{ij} x_j^{(k)} \right), \quad i = 1, 2, \ldots, n$$

Pada setiap iterasi ke-$^{k+1}$, nilai variabel x_i dihitung berdasarkan nilai-nilai variabel lain pada iterasi sebelumnya ke-k. Syarat penting agar metode Jacobi konvergen adalah matriks A harus dominan diagonal, yaitu nilai absolut dari elemen diagonal setiap baris lebih besar daripada jumlah absolut elemen-elemen lainnya dalam baris tersebut. Tanpa sifat ini, iterasi dapat gagal mencapai solusi atau bahkan divergen.

Metode Jacobi sangat cocok untuk implementasi dalam sistem komputasi paralel karena pembaruan setiap variabel tidak saling tergantung selama iterasi berjalan. Namun, dibandingkan dengan metode iteratif lainnya seperti Gauss-Seidel, metode Jacobi umumnya lebih lambat konvergen karena tidak segera memanfaatkan hasil perhitungan terbaru. Meski demikian, metode ini tetap penting secara konseptual dan praktis dalam pengantar komputasi numerik.

Contoh Soal: Penyelesaian Sistem Linear Menggunakan Metode Jacobi

Diketahui sistem persamaan linear berikut:

$$\begin{cases} 10x + 2y - z = 27 \\ -3x - 6y + 2z = -61.5 \\ x + y + 5z = -21.5 \end{cases}$$

$$x = \frac{1}{10}(27 - 2y + z)$$
$$y = \frac{1}{-6}(-61.5 + 3x - 2z)$$
$$z = \frac{1}{5}(-21.5 - x - y)$$

$$x^{(1)} = \frac{1}{10}(27 - 2(0) + 0) = 2.7$$
$$y^{(1)} = \frac{1}{-6}(-61.5 + 3(0) - 2(0)) = 10.25$$
$$z^{(1)} = \frac{1}{5}(-21.5 - 0 - 0) = -4.3$$

$$x^{(2)} = \frac{1}{10}(27 - 2(10.25) + (-4.3)) = \frac{1}{10}(27 - 20.5 - 4.3) = 0.22$$
$$y^{(2)} = \frac{1}{-6}(-61.5 + 3(2.7) - 2(-4.3)) = \frac{1}{-6}(-61.5 + 8.1 + 8.6) = -7.8$$
$$z^{(2)} = \frac{1}{5}(-21.5 - 2.7 - 10.25) = \frac{1}{5}(-34.45) = -6.89$$

$$x^{(2)} \approx 0.22, \quad y^{(2)} \approx -7.8, \quad z^{(2)} \approx -6.89$$

3. Metode Gauss-Seidel

Metode Gauss-Seidel merupakan salah satu teknik iteratif yang digunakan untuk menyelesaikan sistem persamaan linear, dan merupakan pengembangan dari metode Jacobi. Menurut Strang (2016) dalam *Introduction to Linear Algebra*, perbedaan utama antara metode Gauss-Seidel dan Jacobi terletak pada pemanfaatan nilai-nilai variabel yang baru dihitung. Jika metode Jacobi menggunakan nilai dari iterasi sebelumnya untuk seluruh variabel, maka metode Gauss-Seidel langsung menggunakan nilai terbaru dari iterasi saat ini segera setelah diperoleh. Pendekatan ini umumnya mempercepat laju konvergensi, menjadikan Gauss-Seidel lebih efisien dibanding Jacobi dalam banyak kasus.

$$x_i^{(k+1)} = \frac{1}{a_{ii}} \left(b_i - \sum_{j=1}^{i-1} a_{ij} x_j^{(k+1)} - \sum_{j=i+1}^{n} a_{ij} x_j^{(k)} \right)$$

Rumus ini menunjukkan bahwa untuk menghitung x_i pada iterasi ke-$^{k+}$, metode ini menggunakan nilai-nilai terbaru dari variabel-variabel sebelumnya ($x_1^{(k+1)}$, $x_2^{(k+1)}$,…., $x_{i-1}^{(k+1)}$) dan nilai-nilai lama dari variabel yang belum diperbarui ($x_1^{(k+1)}$,..., $x_n^{(k)}$). Pendekatan ini menghasilkan proses konvergensi yang lebih efisien, terutama jika matriks koefisien A bersifat *symmetric positive definite* atau dominan diagonal.

Agar metode Gauss-Seidel konvergen, matriks A umumnya harus memiliki sifat dominansi diagonal atau positif definit. Metode ini sangat efisien untuk sistem sparse berdimensi besar yang muncul dalam rekayasa struktur, simulasi fluida, dan pemodelan fisik lainnya. Meski tidak sebaik Jacobi untuk paralelisasi, Gauss-Seidel lebih unggul dalam

kecepatan konvergensi. Oleh karena itu, metode ini menjadi salah satu pendekatan iteratif yang paling banyak digunakan dalam praktik komputasi numerik.

$$\begin{cases} 4x + y + z = 7 \\ x + 3y + z = -8 \\ x + y + 5z = 6 \end{cases}$$

$$x = \frac{1}{4}(7 - y - z)$$
$$y = \frac{1}{3}(-8 - x - z)$$
$$z = \frac{1}{5}(6 - x - y)$$

Selanjutnya, kita gunakan tebakan awal $x^{(0)} = 0$, $y^{(0)} = 0$, $z^{(0)} = 0$. Dalam metode Gauss-Seidel, setiap nilai variabel baru langsung digunakan dalam perhitungan selanjutnya. Pada iterasi pertama, kita hitung:

$$x^{(1)} = \frac{1}{4}(7 - 0 - 0) = 1.75$$
$$y^{(1)} = \frac{1}{3}(-8 - 1.75 - 0) = \frac{-9.75}{3} = -3.25$$
$$z^{(1)} = \frac{1}{5}(6 - 1.75 - (-3.25)) = \frac{1}{5}(7.5) = 1.5$$

Untuk iterasi kedua, nilai-nilai baru dari iterasi pertama digunakan:

$$x^{(2)} = \frac{1}{4}(7 - (-3.25) - 1.5) = \frac{1}{4}(8.75) = 2.1875$$
$$y^{(2)} = \frac{1}{3}(-8 - 2.1875 - 1.5) = \frac{-11.6875}{3} = -3.8958$$
$$z^{(2)} = \frac{1}{5}(6 - 2.1875 - (-3.8958)) = \frac{1}{5}(7.7083) = 1.5417$$

D. Implementasi dalam Python/MATLAB

Menurut Burden dan Faires (2011) dalam *Numerical Analysis,* metode numerik untuk menyelesaikan sistem persamaan linear terutama metode iteratif seperti Jacobi dan Gauss-Seidel sangat berguna ketika diterapkan menggunakan perangkat lunak komputasi modern. Di antara

banyak platform yang tersedia, Python dan MATLAB merupakan dua lingkungan paling populer dan kuat untuk pemrograman ilmiah dan teknik. Kedua bahasa ini menyediakan pustaka numerik dan struktur data yang efisien untuk menangani sistem linier berskala besar dan kompleks.

1. Python

Python merupakan bahasa pemrograman tingkat tinggi yang sangat populer di bidang komputasi ilmiah dan teknik karena sintaksnya yang sederhana, fleksibel, dan didukung oleh berbagai pustaka numerik yang kuat. Menurut Oliphant (2007) dalam Guide to NumPy, pustaka NumPy menyediakan array multidimensi yang efisien dan mendukung berbagai operasi aljabar linear, sedangkan SciPy memperluas fungsionalitas ini dengan menyediakan alat numerik tingkat lanjut termasuk solver untuk sistem persamaan linear, baik dengan metode langsung maupun iteratif.

Pada konteks penyelesaian sistem persamaan linear, Python menawarkan beberapa pendekatan. Untuk sistem berukuran kecil hingga sedang, metode langsung seperti *numpy.linalg.solve* () sangat efisien. Sebagai contoh, untuk menyelesaikan sistem Ax=b, pengguna cukup menulis:

```python
import numpy as np

A = np.array([[10, 2, 1],
              [1, 5, 1],
              [2, 3, 10]], dtype=float)
b = np.array([7, -8, 6], dtype=float)

x = np.linalg.solve(A, b)
print("Solusi:", x)
```

Untuk sistem berdimensi besar atau matriks yang bersifat sparse (jarang), metode langsung menjadi tidak efisien baik dari segi memori maupun waktu. Dalam kasus ini, metode iteratif seperti Jacobi dan Gauss-Seidel lebih disarankan karena hemat memori dan dapat dihentikan setelah mencapai toleransi kesalahan tertentu. Implementasi metode Jacobi secara manual di Python melibatkan iterasi pembaruan

nilai setiap variabel menggunakan nilai dari iterasi sebelumnya. Berikut contoh kode sederhana:

```
def jacobi(A, b, x0, tol=1e-10, max_iter=100):
    n = len(b)
    x = x0.copy()
    for _ in range(max_iter):
        x_new = x.copy()
        for i in range(n):
            s = sum(A[i][j] * x[j] for j in range(n) if j !=
            x_new[i] = (b[i] - s) / A[i][i]
        if np.linalg.norm(x_new - x, ord=np.inf) < tol:
            return x_new
        x = x_new
    return x
```

Untuk Gauss-Seidel, struktur kode hampir serupa, namun dengan penggunaan nilai-nilai terbaru yang diperoleh selama iterasi:

```
def gauss_seidel(A, b, x0, tol=1e-10, max_iter=100):
    n = len(b)
    x = x0.copy()
    for _ in range(max_iter):
        x_old = x.copy()
        for i in range(n):
            s1 = sum(A[i][j] * x[j] for j in range(i))
            s2 = sum(A[i][j] * x_old[j] for j in range(i + 1, n))
            x[i] = (b[i] - s1 - s2) / A[i][i]
        if np.linalg.norm(x - x_old, ord=np.inf) < tol:
            return x
    return x
```

Pustaka SciPy juga menyediakan metode iteratif seperti Conjugate Gradient (CG) dan BiCGSTAB untuk sistem yang sangat besar. Fungsi seperti *scipy.sparse.linalg.cg*() sangat efisien jika digunakan bersama objek matriks sparse (csr_matrix).

Python juga unggul dalam visualisasi dan dokumentasi hasil, menggunakan pustaka seperti Matplotlib untuk plotting grafik konvergensi atau residual. Secara keseluruhan, Python memberikan kombinasi optimal antara kemudahan pemrograman, fleksibilitas, dan

efisiensi, menjadikannya platform ideal untuk menerapkan dan membahas metode numerik seperti Jacobi dan Gauss-Seidel dalam penyelesaian sistem linear.

2. MATLAB

MATLAB adalah lingkungan komputasi numerik yang dirancang khusus untuk menangani operasi matematika teknik dan ilmiah. Menurut Chapman (2017) dalam MATLAB for Engineers, MATLAB menyediakan sintaks yang ringkas dan efisien untuk melakukan berbagai operasi aljabar linear, termasuk penyelesaian sistem persamaan linear dengan metode langsung maupun metode iteratif. Karena fokus utamanya pada pemrosesan matriks dan vektor, MATLAB menjadi pilihan utama dalam banyak aplikasi teknik, sains komputer, dan analisis data numerik.

Untuk sistem berukuran kecil hingga sedang, MATLAB memiliki operator backslash (\)yang sangat efisien dalam menyelesaikan sistem Ax=b secara langsung. Contohnya:

```
A = [10 2 1; 1 5 1; 2 3 10];
b = [7; -8; 6];
x = A\b;
disp(x)
```

Untuk sistem yang besar atau memiliki struktur sparse (matriks dengan banyak nol), metode langsung menjadi kurang efisien baik dari segi waktu maupun konsumsi memori. Dalam kasus ini, metode iteratif seperti Jacobi dan Gauss-Seidel lebih sesuai karena mampu menangani sistem skala besar dengan lebih ringan. MATLAB mendukung penerapan metode iteratif melalui pemrograman prosedural, serta menyediakan alat bantu visualisasi untuk memantau konvergensi solusi.

Implementasi metode Jacobi dalam MATLAB dapat ditulis secara eksplisit menggunakan loop for:

```
function x = jacobi(A, b, x0, tol, max_iter)
    n = length(b);
    x = x0;
    for k = 1:max_iter
        x_new = zeros(n, 1);
        for i = 1:n
            s = 0;
            for j = 1:n
                if j ~= i
                    s = s + A(i,j)*x(j);
                end
            end
            x_new(i) = (b(i) - s)/A(i,i);
        end
        if norm(x_new - x, inf) < tol
            break;
        end
        x = x_new;
    end
end
```

Untuk sistem berdimensi besar atau matriks yang bersifat sparse (jarang), metode langsung menjadi tidak efisien baik dari segi memori maupun waktu. Dalam kasus ini, metode iteratif seperti Jacobi dan Gauss-Seidel lebih disarankan karena hemat memori dan dapat dihentikan setelah mencapai toleransi kesalahan tertentu.

Implementasi metode Jacobi secara manual di Python melibatkan iterasi pembaruan nilai setiap variabel menggunakan nilai dari iterasi sebelumnya. Berikut contoh kode sederhana:

```
function x = gauss_seidel(A, b, x0, tol, max_iter)
    n = length(b);
    x = x0;
    for k = 1:max_iter
        x_old = x;
        for i = 1:n
            s1 = A(i,1:i-1) * x(1:i-1);
            s2 = A(i,i+1:n) * x_old(i+1:n);
            x(i) = (b(i) - s1 - s2) / A(i,i);
        end
        if norm(x - x_old, inf) < tol
            break;
        end
    end
end
```

Kedua fungsi tersebut menggunakan norma maksimum (*infinity norm*) untuk mengevaluasi apakah solusi telah konvergen pada tingkat toleransi tertentu.

MATLAB juga memiliki fungsi internal seperti pcg (*Preconditioned Conjugate Gradient*) dan lsqr untuk menyelesaikan sistem sparse atau overdetermined. Fungsi-fungsi ini dapat digunakan bersama objek matriks sparse (sparse(A)) untuk meningkatkan efisiensi memori. *Fitur command window* dan *plotting tools* di MATLAB sangat membantu untuk memvisualisasikan error atau kecepatan konvergensi iterasi. Dengan antarmuka grafis yang intuitif, dokumentasi bawaan, dan kapabilitas debugging yang kuat, MATLAB memberikan platform yang sangat sesuai bagi mahasiswa, peneliti, maupun profesional teknik untuk menerapkan dan menguji algoritma numerik dalam penyelesaian sistem linear.

BAB V
INTERPOLASI DAN APROKSIMASI FUNGSI

Interpolasi dan aproksimasi fungsi merupakan salah satu cabang penting dalam komputasi numerik yang berperan besar dalam menyederhanakan persoalan kompleks menjadi bentuk yang dapat dianalisis dan dihitung secara efisien. Dalam banyak kasus praktis, data yang tersedia tidak selalu dalam bentuk fungsi eksak, melainkan berupa himpunan titik diskrit yang dihasilkan dari pengukuran atau eksperimen. Di sinilah interpolasi berfungsi untuk membangun fungsi baru yang melewati seluruh titik data, sementara aproksimasi bertujuan mencari fungsi yang mendekati pola umum data dengan kesalahan seminimal mungkin. Kedua metode ini tidak hanya menjadi fondasi dalam pengolahan sinyal, pemodelan fisik, hingga analisis ekonomi, tetapi juga membentuk dasar bagi pengembangan algoritma dalam *machine learning* dan simulasi numerik. Buku ini menyajikan pembahasan mendalam tentang berbagai teknik interpolasi seperti metode Lagrange, Newton, dan spline, serta pendekatan aproksimasi menggunakan metode Least Squares. Setiap konsep dijelaskan dengan teori yang kuat dan dilengkapi contoh implementasi dalam Python dan MATLAB agar mudah dipahami dan langsung dapat diaplikasikan.

A. Interpolasi Polinomial (Lagrange, Newton)

Interpolasi polinomial merupakan salah satu metode paling fundamental dalam komputasi numerik, digunakan untuk mendekati fungsi atau data diskrit dengan fungsi polinomial. Dua pendekatan yang paling banyak digunakan untuk interpolasi polinomial adalah metode Lagrange dan Newton. Keduanya memiliki perbedaan dalam struktur

penyusunan polinomial, namun sama-sama bertujuan mencari polinomial orde-n yang melewati semua titik data yang diberikan.

Menurut Burden dan Faires (2010), interpolasi adalah proses mencari suatu fungsi yang melewati serangkaian titik data $(x_0,y_0),(x_1,y_1),...,(x_n,y_n)$, di mana tidak ada dua nilai x_{ix} yang sama. Dalam interpolasi polinomial, fungsi interpolasi dicari dalam bentuk polinomial derajat paling tinggi n yang cocok dengan n+1 titik data tersebut. Secara umum, bentuk polinomial interpolasi adalah:

$$P_n(x) = a_0 + a_1 x + a_2 x^2 + \cdots + a_n x^n$$

Daripada menyusun sistem persamaan linear untuk menyelesaikan koefisien ai, pendekatan Lagrange dan Newton menawarkan cara yang lebih sistematis dan efisien.

1. *Interpolasi Polinomial Lagrange*

Interpolasi polinomial Lagrange merupakan salah satu metode klasik dalam komputasi numerik yang digunakan untuk membangun fungsi polinomial yang melewati sekumpulan titik data diskret. Pendekatan ini diperkenalkan oleh Joseph-Louis Lagrange pada abad ke-18 sebagai solusi untuk masalah interpolasi, yaitu mencari suatu fungsi polinomial $P_n(x)$ yang memuat tepat n+1 titik data $(x_0,y_0),(x_1,y_1),...,(x_n,y_n)$, di mana tidak ada dua nilai x_{ix} yang sama. Ide utama dari interpolasi Lagrange adalah menyusun polinomial sebagai kombinasi linier dari basis polinomial $L_i(x)$, yang masing-masing bernilai satu di titik data tertentu dan nol di titik lainnya, sehingga setiap kontribusi y_i hanya aktif pada posisi x_i saja. Secara matematis, polinomial interpolasi Lagrange dinyatakan dalam bentuk:

$$P_n(x) = \sum_{i=0}^{n} y_i \cdot L_i(x)$$

dengan

$$L_i(x) = \prod_{\substack{j=0 \\ j \neq i}}^{n} \frac{x - x_j}{x_i - x_j}$$

Rumus ini menunjukkan bahwa setiap basis polinomial $L_i(x)$ dikonstruksi dengan mengalikan fraksi-fraksi yang memastikan bahwa

nilai $L_i(x_j)=0$ untuk semua $j \neq ij$ dan $L_i(x_i)=1$. Dengan demikian, $P_n(x)$ merupakan penjumlahan dari hasil perkalian antara nilai y_i dan fungsi basis $L_i(x)$, yang menjamin bahwa hasil interpolasi akan melewati semua titik data yang diberikan.

Keunggulan metode Lagrange terletak pada kesederhanaan bentuk matematisnya. Tanpa perlu menyelesaikan sistem persamaan linear atau melakukan operasi matriks, interpolasi dapat dilakukan langsung dari data yang tersedia. Hal ini sangat berguna dalam pengajaran dasar komputasi numerik dan dalam situasi di mana efisiensi bukanlah kendala utama. Namun, metode ini memiliki kekurangan signifikan. Salah satunya adalah kesulitan dalam menambahkan titik data baru; penambahan satu titik baru mengharuskan rekalkulasi seluruh basis polinomial, sehingga metode ini tidak efisien untuk data dinamis atau jumlah data yang besar. Selain itu, metode ini cenderung menghasilkan osilasi besar di bagian tepi domain ketika digunakan pada titik-titik yang tersebar luas (fenomena yang dikenal sebagai osilasi Runge).

Pada penerapan praktis, interpolasi Lagrange banyak digunakan untuk estimasi nilai fungsi di antara data eksperimen, rekonstruksi kurva dalam pemodelan numerik, serta dalam bidang rekayasa dan fisika yang membutuhkan aproksimasi fungsi kompleks dari data terbatas. Implementasinya dalam bahasa pemrograman seperti Python pun cukup sederhana, dan sering digunakan untuk tujuan pendidikan atau aplikasi ringan. Secara keseluruhan, interpolasi polinomial Lagrange memberikan pemahaman fundamental yang penting tentang bagaimana fungsi dapat dibangun dari sekumpulan titik, meskipun dalam kasus aplikasi berskala besar atau data tak beraturan, metode interpolasi lain seperti spline atau Newton mungkin lebih disukai.

Diketahui tiga titik data sebagai berikut:

$$x_0 = 1, \quad f(x_0) = 2$$
$$x_1 = 2, \quad f(x_1) = 3$$
$$x_2 = 4, \quad f(x_2) = 1$$

Gunakan metode interpolasi polinomial Lagrange untuk membentuk polinomial L(x), dan hitung nilai pendekatan fungsi di x=3.

Bentuk umum polinomial Lagrange orde dua (untuk tiga titik) adalah:

$$L(x) = f(x_0)L_0(x) + f(x_1)L_1(x) + f(x_2)L_2(x)$$

Dengan:

$$L_0(x) = \frac{(x - x_1)(x - x_2)}{(x_0 - x_1)(x_0 - x_2)}$$

$$L_1(x) = \frac{(x - x_0)(x - x_2)}{(x_1 - x_0)(x_1 - x_2)}$$

$$L_2(x) = \frac{(x - x_0)(x - x_1)}{(x_2 - x_0)(x_2 - x_1)}$$

Substitusi nilai:

$$L_0(x) = \frac{(x-2)(x-4)}{(1-2)(1-4)} = \frac{(x-2)(x-4)}{(-1)(-3)} = \frac{(x-2)(x-4)}{3}$$

$$L_1(x) = \frac{(x-1)(x-4)}{(2-1)(2-4)} = \frac{(x-1)(x-4)}{(1)(-2)} = -\frac{(x-1)(x-4)}{2}$$

$$L_2(x) = \frac{(x-1)(x-2)}{(4-1)(4-2)} = \frac{(x-1)(x-2)}{(3)(2)} = \frac{(x-1)(x-2)}{6}$$

Maka:

$$L(x) = 2 \cdot L_0(x) + 3 \cdot L_1(x) + 1 \cdot L_2(x)$$

Hitung nilai L(3):

$$L_0(3) = \frac{(3-2)(3-4)}{3} = \frac{(1)(-1)}{3} = -\frac{1}{3}$$

$$L_1(3) = -\frac{(3-1)(3-4)}{2} = -\frac{(2)(-1)}{2} = 1$$

$$L_2(3) = \frac{(3-1)(3-2)}{6} = \frac{(2)(1)}{6} = \frac{1}{3}$$

Maka:

$$L(3) = 2 \cdot \left(-\frac{1}{3}\right) + 3 \cdot (1) + 1 \cdot \left(\frac{1}{3}\right) = -\frac{2}{3} + 3 + \frac{1}{3} = 2.6667$$

2. *Interpolasi Polinomial Newton*

Interpolasi polinomial Newton adalah salah satu metode interpolasi numerik yang dirancang untuk menyusun polinomial yang melewati sekumpulan titik data $(x_0,y_0),(x_1,y_1),\ldots,(x_n,y_n)$ dengan cara yang efisien dan fleksibel. Berbeda dengan metode Lagrange yang menghitung seluruh bentuk polinomial sekaligus, metode Newton

menggunakan pendekatan rekursif berdasarkan konsep selisih terbagi (*divided differences*). Metode ini memungkinkan pembangunan polinomial secara bertahap, sehingga sangat efisien jika diperlukan penambahan titik baru tanpa harus menghitung ulang seluruh polinomial yang telah dibentuk sebelumnya.
Polinomial Newton ditulis dalam bentuk:

$$P_n(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \cdots + a_n(x - x_0)(x - x_1)\ldots(x - :$$

Koefisien aia_iai di sini diperoleh dari tabel selisih terbagi, yang dihitung secara rekursif dari nilai-nilai y_i. Proses ini dimulai dari nilai $f[x_i]=y_i$, kemudian menghitung selisih dua nilai berturut-turut dibagi dengan selisih titik x-nya:

$$f[x_i, x_{i+1}] = \frac{f[x_{i+1}] - f[x_i]}{x_{i+1} - x_i}$$

dan dilanjutkan ke orde lebih tinggi seperti:

$$f[x_i, x_{i+1}, x_{i+2}] = \frac{f[x_{i+1}, x_{i+2}] - f[x_i, x_{i+1}]}{x_{i+2} - x_i}$$

Keunggulan utama interpolasi Newton adalah kemampuannya dalam menyusun polinomial secara bertahap, menjadikannya lebih efisien dibanding Lagrange, terutama ketika data baru ditambahkan. Dengan hanya menghitung satu suku tambahan dan satu koefisien baru, polinomial yang telah dibentuk dapat diperluas tanpa perhitungan ulang seluruhnya. Ini membuat metode Newton sangat cocok untuk aplikasi dengan jumlah data bertambah secara dinamis. Selain itu, bentuk rekursifnya juga mempermudah proses komputasi numerik, baik secara manual maupun dalam program komputer.

Metode Newton memiliki kekurangan dalam kompleksitas awal pembuatan tabel selisih terbagi, terutama bila tidak dilakukan secara otomatis. Kesalahan dalam perhitungan selisih terbagi dapat menjalar ke hasil akhir, karena setiap koefisien bergantung pada hasil sebelumnya. Selain itu, jika titik x_i sangat berdekatan atau data mengandung noise tinggi, perhitungan dapat menjadi tidak stabil.

Pada praktiknya, interpolasi Newton banyak digunakan dalam rekayasa, fisika komputasi, ekonomi, dan bidang-bidang yang

memerlukan estimasi nilai fungsi di antara titik-titik data. Kelebihannya dalam fleksibilitas dan efisiensi menjadikannya metode yang disukai dalam implementasi algoritmik. Di berbagai bahasa pemrograman seperti Python dan MATLAB, algoritma Newton sangat mudah diimplementasikan menggunakan array dan operasi rekursif, menjadikannya alat penting dalam toolkit numerik modern. Dengan dasar teori yang kuat dan struktur perhitungan yang sistematis, interpolasi Newton merupakan pendekatan yang sangat relevan dalam pengolahan dan pemodelan data numerik.

Diberikan tiga titik data berikut:

$$x_0 = 1, \quad f(x_0) = 1$$
$$x_1 = 2, \quad f(x_1) = 4$$
$$x_2 = 3, \quad f(x_2) = 9$$

Gunakan metode interpolasi Newton untuk membentuk polinomial interpolasi dan hitung nilai pendekatan fungsi di x=2.5.

Buat tabel selisih terbagi:

x	$f[x]$	$f[x_1, x_0]$	$f[x_2, x_1, x_0]$
1	1		
2	4	$\frac{4-1}{2-1} = 3$	
3	9	$\frac{9-4}{3-2} = 5$	$\frac{5-3}{3-1} = 1$

Jadi:

- $f[x_0] = 1$
- $f[x_1, x_0] = 3$
- $f[x_2, x_1, x_0] = 1$

Rumus polinomial Newton:

$$P(x) = f[x_0] + f[x_1, x_0](x - x_0) + f[x_2, x_1, x_0](x - x_0)(x - x_1)$$

Substitusi nilai:

$$P(x) = 1 + 3(x - 1) + 1(x - 1)(x - 2)$$

Langkah 3: Hitung $P(2.5)$

$$\begin{aligned} P(2.5) &= 1 + 3(2.5 - 1) + (2.5 - 1)(2.5 - 2) \\ &= 1 + 3(1.5) + (1.5)(0.5) \\ &= 1 + 4.5 + 0.75 = 6.25 \end{aligned}$$

Hasil pendekatan dengan interpolasi Newton menunjukkan bahwa $f(2.5) \approx 6.25$, yang mendekati nilai eksak dari fungsi $f(x)=x^2$ pada $x=2.5$.

B. Interpolasi Spline dan Kurva Halus

Interpolasi spline merupakan salah satu metode numerik yang dirancang untuk menghasilkan kurva halus yang melewati sekumpulan titik data, dengan menghindari osilasi ekstrem yang sering muncul pada interpolasi polinomial derajat tinggi. Metode ini menjadi penting dalam berbagai bidang seperti grafik komputer, pemodelan geometri, simulasi ilmiah, dan teknik rekayasa karena mampu menghasilkan kurva yang tidak hanya akurat tetapi juga estetis dan stabil secara numerik.

Menurut Chapra dan Canale (2015) dalam bukunya Numerical Methods for Engineers, interpolasi spline adalah proses menyusun potongan-potongan fungsi polinomial berorde rendah yang disambungkan secara kontinu pada titik-titik data. Fungsi spline dirancang sedemikian rupa sehingga setiap potongan kurva (disebut segmen spline) hanya berlaku pada interval tertentu di antara dua titik data, dan memiliki kontinuitas hingga turunan kedua atau lebih pada titik sambungan (disebut knots). Hal ini membuat spline menjadi solusi ideal dalam interpolasi yang menuntut kurva halus dan stabil.

Jenis spline yang paling umum digunakan adalah Spline Kubik (Cubic Spline), di mana masing-masing segmen kurva adalah polinomial derajat tiga. Bentuk umum spline kubik pada setiap interval $[x_i, x_{i+1}]$ adalah:

$$S_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3$$

Koefisien a_i,b_i,c_i, di ditentukan berdasarkan kondisi interpolasi (kurva harus melewati titik data), dan syarat kekontinuan turunan pertama dan kedua di titik sambung antar segmen.

1. Kelebihan Spline Dibandingkan Polinomial Global

Interpolasi spline memiliki sejumlah keunggulan penting dibandingkan interpolasi polinomial global, terutama dalam hal kestabilan numerik, fleksibilitas, dan keakuratan lokal. Polinomial global, seperti interpolasi Lagrange atau Newton, menyusun satu fungsi polinomial berderajat tinggi yang mencakup seluruh domain data, artinya satu fungsi harus melewati semua titik data. Meskipun pendekatan ini secara teoritis valid, dalam praktiknya sering kali menimbulkan masalah, terutama jika jumlah titik data cukup banyak atau jika titik-titik tersebut tersebar secara tidak merata. Salah satu masalah paling terkenal adalah osilasi Runge, yaitu fenomena di mana polinomial derajat tinggi berosilasi secara ekstrem di dekat ujung-ujung domain, menyebabkan interpolasi yang tidak akurat dan tidak realistis. Hal ini terutama terjadi jika titik data tersebar secara ekuidistan. Dalam konteks ini, interpolasi spline memberikan solusi yang jauh lebih stabil dan dapat diandalkan.

Spline, khususnya spline kubik, menyusun interpolasi dalam bentuk segmen-segmen polinomial rendah (biasanya derajat tiga) yang diterapkan pada setiap interval antar dua titik data. Setiap segmen ini memiliki koefisiennya sendiri, namun disatukan melalui syarat kekontinuan nilai fungsi, turunan pertama, dan bahkan turunan kedua di titik sambung. Karena setiap polinomial hanya berlaku pada satu interval lokal, spline menghindari masalah osilasi yang terjadi pada pendekatan global. Sebagaimana dijelaskan oleh Burden dan Faires (2010), penggunaan polinomial derajat rendah secara lokal jauh lebih stabil secara numerik, karena galat interpolasi terkendali dan tidak berkembang secara ekstrem seiring bertambahnya jumlah titik data.

Keunggulan lain *spline* adalah fleksibilitas dalam menangani jumlah data yang besar. Dalam interpolasi polinomial global, penambahan titik data mengubah struktur seluruh polinomial, sehingga memerlukan penghitungan ulang keseluruhan fungsi interpolasi.

Sebaliknya, dalam *spline*, penambahan titik data hanya memengaruhi segmen di sekitar titik baru, sehingga perhitungan dapat dilakukan secara lebih modular dan efisien. Ini sangat berguna dalam aplikasi dinamis, seperti dalam grafik komputer atau pemrosesan sinyal waktu nyata, di mana data terus berkembang.

Spline juga mendukung pengaturan kondisi batas yang lebih fleksibel, seperti dalam *natural spline* (dengan turunan kedua nol di ujung), *clamped spline* (dengan kemiringan ujung yang ditentukan), dan *smoothing spline* (yang memungkinkan penyimpangan dari titik data untuk menghindari *overfitting*). Dengan demikian, *spline* tidak hanya menyediakan interpolasi yang akurat, tetapi juga memberikan kontrol yang lebih besar terhadap bentuk kurva.

2. Jenis-Jenis *Spline*

Spline adalah bentuk interpolasi numerik yang mengandalkan potongan-potongan polinomial derajat rendah untuk membentuk kurva halus yang melewati titik-titik data. Keunggulan metode ini terletak pada kemampuannya menghasilkan interpolasi yang stabil dan halus tanpa harus menggunakan polinomial derajat tinggi yang rentan terhadap osilasi. Dalam praktiknya, terdapat beberapa jenis *spline* yang dikembangkan untuk memenuhi berbagai kebutuhan interpolasi dan pemodelan data. Jenis-jenis *spline* ini dibedakan berdasarkan kondisi batas, derajat polinomial yang digunakan, serta cara pengontrolan kekontinuan dan kelengkungan antar segmen. Jenis *spline* yang paling umum meliputi natural s*pline, clamped spline, not-a-knot spline, smoothing spline*, dan *B-spline*.

Natural spline adalah jenis spline kubik yang menetapkan bahwa turunan kedua dari fungsi spline di titik ujung (*boundary*) adalah nol, yaitu $S''(x_0)=0$ dan $S''(x_n)=0$. Kondisi ini memberikan bentuk kurva yang cenderung datar di ujung domain, mencerminkan asumsi bahwa kelengkungan di luar titik data dianggap tidak signifikan. *Natural spline* sangat populer karena secara matematis sederhana dan cocok untuk data yang tidak memiliki informasi tambahan di batas domain.

Berbeda dengan itu, *clamped spline* menetapkan nilai turunan pertama (kemiringan) pada titik ujung domain. Artinya, pengguna harus mengetahui atau memperkirakan $S'(x_0)$ dan $S'(x_n)$. *Clamped spline* sangat berguna ketika kemiringan atau kecepatan perubahan data pada batas domain sudah diketahui, misalnya dalam pemodelan mekanika

atau fisika, di mana gradien pada batas bisa dihitung dari teori atau eksperimen.

Jenis lainnya, *not-a-knot spline*, menghilangkan status simpul pada titik kedua dan titik kedua dari akhir, yaitu x_1 dan x_{n-1}. Dengan kata lain, *spline* pada interval $[x_0,x_2]$ dan $[x_{n-2},x_n]$ diperlakukan seolah-olah sebagai satu segmen tunggal. Tujuan pendekatan ini adalah untuk meminimalkan jumlah kondisi sambungan dan menyederhanakan sistem, sambil tetap menjaga kekontinuan hingga turunan kedua.

Smoothing spline adalah jenis *spline* yang tidak memaksa kurva untuk melewati setiap titik data, tetapi berusaha meminimalkan gabungan antara kesalahan interpolasi dan kelengkungan kurva. *Spline* ini sangat cocok untuk data yang mengandung noise, karena tidak terlalu sensitif terhadap fluktuasi kecil. Fungsi objektif *smoothing spline* biasanya berbentuk:

$$\min_g \left[\sum_{i=1}^{n} (y_i - g(x_i))^2 + \lambda \int (g''(x))^2 dx \right]$$

di mana λ adalah parameter regularisasi. Ketika λ besar, *spline* menjadi lebih halus; ketika kecil, *spline* lebih mendekati data.

B-spline (*Basis spline*) dan Spline NURBS (*Non-Uniform Rational B-Splines*) digunakan secara luas dalam grafik komputer dan CAD (*Computer-Aided Design*). B-spline adalah representasi spline dalam basis tertentu yang memberikan fleksibilitas tinggi dan kontrol lokal. Tidak seperti spline polinomial biasa, perubahan pada satu titik kontrol hanya memengaruhi segmen tertentu, membuatnya sangat efisien untuk manipulasi bentuk dalam desain. Dengan berbagai jenis spline yang tersedia, pengguna dapat memilih metode yang paling sesuai dengan sifat data dan kebutuhan aplikasi. Pemilihan jenis spline yang tepat akan menghasilkan interpolasi yang tidak hanya akurat, tetapi juga halus, stabil, dan representatif terhadap perilaku data sebenarnya.

3. Proses Pembentukan Spline Kubik

Proses pembentukan spline kubik merupakan tahap penting dalam interpolasi numerik yang bertujuan menghasilkan kurva halus yang melewati serangkaian titik data diskrit. Spline kubik adalah interpolasi yang menggunakan potongan-potongan fungsi polinomial derajat tiga pada setiap interval antar dua titik data. Masing-masing

segmen spline diwakili oleh suatu fungsi $S_i(x)$ yang berbentuk polinomial kubik:

$$S_i(x) = a_i + b_i(x - x_i) + c_i(x - x_i)^2 + d_i(x - x_i)^3$$

di mana $x \in [x_i, x_{i+1}]$. Tujuannya adalah menemukan koefisien a_i, b_i, c_i, dan d_i untuk setiap interval sehingga seluruh potongan spline tersambung secara mulus membentuk kurva kontinyu, baik dalam nilai fungsinya maupun turunannya.

Langkah pertama dalam membentuk spline kubik adalah menetapkan syarat bahwa kurva harus melewati semua titik data, artinya $S_i(x_i) = y_i$ dan $S_i(x_{i+1}) = y_{i+1}$. Ini menghasilkan dua persamaan untuk setiap segmen. Selanjutnya, karena spline harus membentuk kurva yang halus, maka diperlukan syarat kekontinuan turunan pertama dan turunan kedua di setiap titik sambungan x_1, x_2,..., x_{n-1}. Syarat ini menghasilkan dua persamaan tambahan per titik sambungan, yakni: $S'_i(x_{i+1}) = S_{i+1}$ dan $S''_i(x_{i+1}) = S''_{i+1}(x_{i+1})$.

Untuk n titik data, akan terbentuk $n-1$ segmen spline, dan total ada $4(n-1)$ koefisien yang harus dihitung, sehingga diperlukan jumlah persamaan yang sama. Untuk melengkapi sistem persamaan, dua syarat tambahan harus diberikan sebagai kondisi batas. Dalam *natural spline*, misalnya, ditetapkan bahwa turunan kedua di kedua ujung domain nol, yaitu $S''_0(x_0) = 0$ dan $S''_{n-2}(x_n) = 0$. Alternatifnya, dalam *clamped spline*, ditentukan nilai kemiringan di ujung domain, seperti $S'_0(x_0) = m_0$ dan $S'_{n-2}(x_n) = m_n$, berdasarkan informasi yang diketahui atau diasumsikan.

Setelah semua syarat dituliskan, langkah selanjutnya adalah menyusun sistem persamaan linear dalam bentuk matriks, khususnya matriks tridiagonal, karena hanya koefisien dari titik-titik berdekatan yang saling terkait. Umumnya, sistem ini difokuskan pada pencarian nilai koefisien c_i (turunan kedua), karena dari nilai c_i, koefisien lainnya dapat dihitung dengan rumus langsung. Sistem tridiagonal ini kemudian diselesaikan menggunakan metode eliminasi Gauss atau algoritma Thomas. Setelah semua koefisien diperoleh, fungsi spline dapat digunakan untuk mengestimasi nilai fungsi di antara titik-titik data secara halus dan stabil. Dengan desain proses yang memastikan kekontinuan fungsi dan turunannya, spline kubik menjadi metode

interpolasi unggulan dalam berbagai aplikasi rekayasa, grafik komputer, dan pemodelan ilmiah.

C. *Least Squares* dan Regresi Polinomial

Pada analisis data dan komputasi numerik, kita sering kali dihadapkan pada sekumpulan data diskrit yang tidak memiliki hubungan eksak atau pasti satu sama lain, baik karena adanya noise, kesalahan pengukuran, maupun karena hubungan antara variabel memang tidak linier. Dalam situasi seperti ini, metode interpolasi tidak lagi memadai karena interpolasi mensyaratkan kurva harus melalui seluruh titik data. Sebaliknya, metode aproksimasi diperlukan untuk menemukan suatu fungsi yang mendekati pola umum dari data tersebut, dan salah satu metode paling populer dalam pendekatan ini adalah metode Least Squares atau kuadrat terkecil. Ketika fungsi pendekatan berbentuk polinomial, metode ini dikenal sebagai regresi polinomial.

1. Konsep *Least Squares*

Konsep *Least Squares* atau metode kuadrat terkecil merupakan pendekatan dasar dan penting dalam statistik dan komputasi numerik yang digunakan untuk mencari fungsi aproksimasi terbaik terhadap sekumpulan data yang tidak sepenuhnya presisi atau tidak mengikuti pola tertentu secara eksak. Metode ini dirancang untuk meminimalkan jumlah kuadrat selisih antara nilai-nilai hasil observasi atau eksperimen dengan nilai-nilai yang diprediksi oleh suatu model matematis. Artinya, dalam konteks hubungan antara dua variabel, metode *least squares* bertujuan menemukan garis atau kurva yang paling "pas" di tengah data, bukan yang melalui setiap titik secara sempurna, seperti dalam interpolasi. Hal ini sangat relevan dalam dunia nyata, karena data hasil observasi sering kali mengandung noise atau kesalahan pengukuran sehingga tidak cocok diinterpolasi secara langsung.

Secara matematis, diberikan sekumpulan data $(x_1,y_1),(x_2,y_2),...,(x_n,y_n)$, tujuan dari least squares adalah mencari fungsi aproksimasi f(x) sedemikian rupa sehingga total kuadrat galat S diminimalkan:

$$S = \sum_{i=1}^{n} [y_i - f(x_i)]^2$$

Fungsi f(x) dapat berupa model linier, polinomial, eksponensial, atau bentuk lainnya tergantung kebutuhan. Dalam kasus paling sederhana, yaitu regresi linier, model $f(x)$ diasumsikan berbentuk garis lurus $f(x)=a_0+a_1x$, dan metode least squares digunakan untuk menentukan koefisien a0dan a 1yang memberikan nilai minimum bagi S. Proses ini melibatkan penurunan fungsi kesalahan total S terhadap masing-masing parameter, menghasilkan sistem persamaan normal yang kemudian diselesaikan untuk mendapatkan parameter terbaik.

Keunggulan metode *least squares* terletak pada kesederhanaannya, baik dalam konsep maupun implementasi. Ia tidak hanya digunakan untuk menemukan parameter model linier, tetapi juga dapat diperluas untuk regresi polinomial, multivariat, dan model non-linier melalui modifikasi algoritma atau penggunaan transformasi basis. Bahkan, least squares menjadi pondasi utama dalam banyak algoritma *machine learning*, pengolahan sinyal, dan pemodelan ekonomi. Dalam banyak situasi praktis, metode ini menawarkan solusi yang cepat dan akurat, terutama ketika hubungan antara variabel sulit didekati secara eksak.

Metode *least squares* juga memiliki keterbatasan. Ia sangat sensitif terhadap outlier titik data yang menyimpang ekstrem dari pola umum karena kuadrat galat memperbesar pengaruh deviasi besar. Oleh karena itu, dalam kasus data dengan banyak outlier, digunakan pendekatan alternatif seperti least absolute deviations atau robust regression. Meskipun begitu, secara keseluruhan, konsep least squares tetap menjadi alat analisis numerik dan statistika yang sangat penting karena kemampuannya menyederhanakan persoalan aproksimasi data yang kompleks menjadi bentuk matematis yang dapat dipecahkan secara sistematis dan efisien.

2. Regresi Linier sebagai Kasus Khusus

Regresi linier merupakan bentuk paling sederhana dan paling fundamental dari metode Least Squares, menjadikannya kasus khusus yang sangat penting dalam analisis data dan komputasi numerik. Regresi linier digunakan ketika pola hubungan antara dua variabel dapat didekati dengan fungsi garis lurus, yaitu dalam bentuk model matematis:

$$y = a_0 + a_1 x + \varepsilon$$

di mana y adalah variabel dependen (respons), x adalah variabel independen (prediktor), a_0 adalah intersep (titik potong sumbu-y), a_1 adalah kemiringan garis (gradien), dan ε adalah komponen kesalahan (residual). Tujuan regresi linier adalah mencari nilai a_0 dan a_1 yang meminimalkan jumlah kuadrat galat antara nilai aktual y_i dan nilai yang diprediksi oleh garis regresi, yaitu $f(x_i)=a_0+a_1x_i$.

Proses penurunan model regresi linier sederhana melibatkan penggunaan prinsip Least Squares, dengan membentuk fungsi galat total:

$$S = \sum_{i=1}^{n} (y_i - a_0 - a_1 x_i)^2$$

Kemudian, nilai S diminimalkan terhadap parameter a_0 dan a_1 dengan mengambil turunan parsial terhadap masing-masing parameter dan menyamakannya dengan nol, sehingga diperoleh sistem persamaan normal sebagai berikut:

$$\begin{cases} na_0 + a_1 \sum x_i = \sum y_i \\ a_0 \sum x_i + a_1 \sum x_i^2 = \sum x_i y_i \end{cases}$$

Sistem ini dapat diselesaikan secara aljabar untuk memperoleh estimasi parameter regresi. Setelah parameter diperoleh, garis regresi linier dapat digunakan untuk memprediksi nilai y untuk input x baru, dan juga untuk mengukur sejauh mana variabel x berpengaruh terhadap y.

Menurut Montgomery, Peck & Vining (2012), regresi linier tidak hanya memberikan garis terbaik, tetapi juga menyertakan kemampuan untuk mengukur keakuratan model melalui nilai-nilai statistik seperti koefisien determinasi (R^2), nilai p, dan analisis residual. Nilai R^2, misalnya, mengindikasikan seberapa besar proporsi variansi data yang dapat dijelaskan oleh model regresi. Jika R^2=0.95, maka 95% variasi dalam data y dapat dijelaskan oleh variasi dalam x, sedangkan sisanya dianggap sebagai noise atau kesalahan.

Regresi linier juga sangat mudah diterapkan dalam perangkat lunak statistik dan bahasa pemrograman seperti Python, R, dan MATLAB. Fungsionalitas ini membuatnya menjadi alat utama dalam eksplorasi data awal (*exploratory data analysis*), pemodelan prediktif, serta dalam validasi hipotesis hubungan antar variabel. Namun, regresi linier memiliki asumsi dasar yang perlu diperhatikan agar hasilnya valid, seperti linearitas hubungan, normalitas residual, homoskedastisitas (kesamaan variansi), dan tidak adanya autokorelasi. Jika asumsi ini

dilanggar, hasil model bisa menjadi bias atau menyesatkan. Berikut contoh soalnya.

Seorang peneliti ingin mengetahui hubungan antara jumlah jam belajar (X) dan nilai ujian matematika (Y) siswa. Berikut data lima siswa:

Jam Belajar (X)	Nilai Ujian (Y)
2	65
3	70
5	75
7	85
9	95

Gunakan regresi linier sederhana untuk menentukan persamaan regresi dan prediksi nilai ujian jika seseorang belajar selama 6 jam.

Jawaban :

$$\bar{X} = \frac{2+3+5+7+9}{5} = 5.2, \quad \bar{Y} = \frac{65+70+75+85+95}{5} = 78$$

Hitung koefisien b dan a

$$b = \frac{\sum(X_i - \bar{X})(Y_i - \bar{Y})}{\sum(X_i - \bar{X})^2}$$

- $\sum(X_i - \bar{X})(Y_i - \bar{Y}) = 166$
- $\sum(X_i - \bar{X})^2 = 32.8$

$$b = \frac{166}{32.8} \approx 5.06$$

$$a = \bar{Y} - b\bar{X} = 78 - (5.06)(5.2) \approx 51.67$$

Jadi, **persamaan regresi:**

$$Y = 51.67 + 5.06X$$

$$Y = 51.67 + 5.06(6) = 51.67 + 30.36 = 82.03$$

Dengan regresi linier sederhana, diperoleh model Y=51.67+5.06X. Jika seseorang belajar selama 6 jam, diperkirakan akan memperoleh nilai sekitar 82.03.

3. Regresi Polinomial

Regresi polinomial adalah perluasan dari regresi linier yang memungkinkan model untuk menangkap hubungan yang bersifat nonlinier antara variabel independen (x) dan variabel dependen (y). Dalam regresi linier, model dibatasi hanya pada garis lurus ($y=a_0+a_1x$), sehingga kurang fleksibel ketika data menunjukkan pola lengkung atau perubahan arah yang tidak bisa ditangkap oleh garis lurus. Untuk mengatasi keterbatasan ini, regresi polinomial menggunakan fungsi polinomial sebagai model aproksimasi, yaitu:

$$y = a_0 + a_1x + a_2x^2 + a_3x^3 + \cdots + a_mx^m + \varepsilon$$

di mana mmm adalah derajat polinomial, $a_0,a_1,...,a_m$ adalah koefisien regresi, dan ε\varepsilonε adalah komponen galat (error). Tujuan dari regresi polinomial tetap sama seperti pada regresi linier: meminimalkan jumlah kuadrat selisih antara nilai prediksi dan data aktual menggunakan metode *Least Squares*.

Menurut Chapra dan Canale (2015) dalam *Numerical Methods for Engineers*, regresi polinomial berguna ketika terdapat indikasi bahwa data memiliki hubungan melengkung, misalnya seperti kurva parabola, eksponensial, atau siklikal. Model polinomial memungkinkan kita menangkap berbagai bentuk tren tersebut dengan cara menambahkan pangkat variabel independen ke dalam model regresi. Derajat polinomial yang digunakan sangat menentukan bentuk dan fleksibilitas kurva hasil. Misalnya, polinomial orde dua ($y=a_0+a_{1x}+a_2x_2$) cukup untuk mendeskripsikan tren berbentuk parabola, sementara orde tiga atau lebih tinggi digunakan untuk bentuk yang lebih kompleks. Proses perhitungan

koefisien regresi polinomial melibatkan penyusunan sistem persamaan normal yang lebih besar dibanding regresi linier. Dalam hal ini, kita menghitung jumlah hasil kali dari x dengan berbagai pangkatnya, serta dengan y, untuk membentuk sistem persamaan linear yang dapat diselesaikan menggunakan eliminasi Gauss atau metode numerik lainnya.

Regresi polinomial sangat fleksibel dan banyak digunakan dalam bidang teknik, ekonomi, dan ilmu data, terutama ketika pola data tidak dapat dijelaskan dengan baik oleh model linier. Namun demikian, model ini juga memiliki kelemahan. Jika derajat polinomial terlalu tinggi, model cenderung mengalami *overfitting*, yaitu menyesuaikan diri secara berlebihan dengan data pelatihan hingga kehilangan kemampuan generalisasi terhadap data baru. Selain itu, polinomial derajat tinggi bisa menimbulkan osilasi tajam antara titik-titik data, mirip dengan fenomena osilasi Runge pada interpolasi.

Untuk mengatasi hal tersebut, pemilihan derajat polinomial harus hati-hati, bisa menggunakan teknik validasi silang (*cross-validation*), kriteria Akaike (AIC), atau *Bayesian Information Criterion* (BIC). Secara keseluruhan, regresi polinomial adalah alat yang sangat berguna dalam pendekatan aproksimatif terhadap data nonlinier, selama digunakan dengan pertimbangan metodologis yang tepat.

```python
import numpy as np
import matplotlib.pyplot as plt

# Data
x = np.array([1, 2, 3, 4, 5])
y = np.array([2.2, 2.8, 3.6, 4.5, 5.1])

# Regresi polinomial orde 2
coeffs = np.polyfit(x, y, 2)
p = np.poly1d(coeffs)

# Evaluasi kurva
x_new = np.linspace(min(x), max(x), 100)
y_new = p(x_new)

# Visualisasi
plt.scatter(x, y, label="Data")
plt.plot(x_new, y_new, 'r-', label="Polinomial")
plt.legend()
plt.title("Regresi Polinomial Orde 2")
plt.grid(True)
plt.show()
```

Kode ini menunjukkan bagaimana polinomial orde dua dapat diaproksimasi ke data yang cenderung non-linier menggunakan fungsi *np.polyfit*().Berikut adalah contoh soal dan jawabannya mengenai Regresi Polinomial.

Seorang analis ingin memodelkan hubungan antara usia kendaraan (X, dalam tahun) dan biaya perawatan tahunan (Y, dalam juta rupiah). Data berikut dikumpulkan:

Usia Kendaraan (X)	Biaya Perawatan (Y)
1	2
2	2.5
3	3.7
4	5.8
5	9

Gunakan regresi polinomial orde 2 (kuadrat) untuk menemukan persamaan regresi dan prediksi biaya perawatan saat usia kendaraan 3,5 tahun.

Langkah 1: Bentuk model regresi polinomial kuadrat

Model umumnya:

$$Y = a + bX + cX^2$$

Gunakan software seperti Excel, Python, atau kalkulasi manual untuk mendapatkan koefisien a, b, c. Dengan metode least squares, diperoleh hasil:

$$Y = 0.46 + 0.29X + 0.56X^2$$

Langkah 2: Prediksi saat $X = 3.5$

$$\begin{aligned} Y &= 0.46 + 0.29(3.5) + 0.56(3.5)^2 \\ &= 0.46 + 1.015 + 0.56(12.25) \\ &= 0.46 + 1.015 + 6.86 = 8.335 \end{aligned}$$

Model regresi polinomial yang diperoleh adalah:

$$Y = 0.46 + 0.29X + 0.56X^2$$

Dengan model tersebut, diperkirakan biaya perawatan kendaraan saat usia 3,5 tahun adalah sekitar **8,34 juta rupiah**.

D. Visualisasi dan Evaluasi Aproksimasi

Pada konteks komputasi numerik dan analisis data, aproksimasi merupakan metode penting untuk mendekati fungsi atau data yang tidak diketahui bentuk analitiknya secara eksak. Aproksimasi sering digunakan ketika data hasil eksperimen atau pengamatan tidak dapat diwakili secara sempurna oleh model eksak, sehingga dibutuhkan pendekatan numerik seperti regresi atau interpolasi. Namun, membangun model aproksimasi hanyalah langkah awal yang tak kalah penting adalah visualisasi dan evaluasi dari hasil aproksimasi tersebut. Visualisasi memungkinkan pemahaman intuitif terhadap kualitas kurva aproksimasi, sementara evaluasi memberikan ukuran kuantitatif terhadap akurasi dan reliabilitas model tersebut.

1. Pentingnya Visualisasi Aproksimasi

Visualisasi aproksimasi merupakan langkah esensial dalam proses analisis data dan komputasi numerik karena membantu menyampaikan secara intuitif sejauh mana model aproksimatif mewakili data yang sebenarnya. Dalam konteks aproksimasi, baik menggunakan regresi linier, polinomial, atau teknik lain seperti spline, visualisasi memungkinkan kita mengevaluasi kualitas kurva hasil secara langsung melalui representasi grafis. Tanpa visualisasi, analisis terhadap model aproksimasi hanya akan mengandalkan metrik numerik seperti RMSE, MAE, atau R^2, yang meskipun objektif, sering kali tidak cukup menggambarkan perilaku model terhadap data secara menyeluruh. Visualisasi memperlihatkan aspek-aspek yang tidak tertangkap oleh angka, seperti outlier, pola sistematis dalam residual, atau indikasi *overfitting* dan *underfitting*.

Menurut Chapra dan Canale (2015) dalam *Numerical Methods for Engineers,* visualisasi sangat berguna untuk mendeteksi kecocokan antara model dan data secara lokal maupun global. Misalnya, ketika kurva hasil aproksimasi diplot bersamaan dengan titik-titik data aktual, kita bisa segera melihat apakah kurva tersebut terlalu kaku (underfit) atau terlalu berlekuk mengikuti data (overfit). Bahkan ketika nilai koefisien determinasi R^2 tinggi, bisa saja kurva menampilkan osilasi liar akibat pemilihan model yang tidak tepat, seperti pada regresi polinomial derajat tinggi. Hal seperti ini hanya bisa diidentifikasi dengan jelas melalui visualisasi, bukan sekadar dari nilai metrik statistik.

Visualisasi juga penting dalam memahami distribusi kesalahan (residual). Plot residual terhadap variabel independen dapat menunjukkan apakah galat tersebar secara acak atau membentuk pola tertentu. Jika residual menunjukkan pola sistematis, seperti pola melengkung atau menaik-menurun, hal itu menunjukkan bahwa model tidak menangkap karakteristik data dengan baik. Sebaliknya, jika residual tersebar acak di sekitar garis nol, ini mengindikasikan bahwa model telah menangkap tren data dengan cukup baik. Visualisasi residual ini juga menjadi langkah penting dalam menguji asumsi-asumsi statistik pada regresi, seperti linearitas, homoskedastisitas, dan normalitas.

Pada aplikasi dunia nyata, visualisasi aproksimasi juga memudahkan komunikasi dan interpretasi hasil. Peneliti, analis, atau pengambil keputusan sering kali bukan ahli statistik atau numerik,

sehingga menyampaikan hasil dalam bentuk grafik yang mudah dipahami jauh lebih efektif dibandingkan tabel angka dan persamaan. Grafik regresi atau kurva aproksimasi juga sangat membantu dalam presentasi teknis, laporan penelitian, dan dokumentasi ilmiah.

2. Evaluasi Aproksimasi

Evaluasi aproksimasi adalah proses penting untuk menilai seberapa baik suatu model matematis atau numerik dalam merepresentasikan hubungan antara variabel-variabel dalam sekumpulan data. Dalam konteks komputasi numerik dan analisis data, aproksimasi sering digunakan ketika model eksak tidak tersedia atau hubungan antar variabel terlalu kompleks untuk dijelaskan secara analitik. Oleh karena itu, setelah membentuk model aproksimasi baik melalui regresi linier, regresi polinomial, spline, atau metode lainnya kita perlu mengevaluasi performa model tersebut secara kuantitatif dan objektif. Evaluasi ini bertujuan untuk memastikan bahwa model tidak hanya cocok pada data yang tersedia (fit), tetapi juga memiliki kemampuan generalisasi yang baik terhadap data baru atau tak terlihat sebelumnya.

Menurut Burden dan Faires (2010) dalam *Numerical Analysis*, salah satu cara paling umum dalam mengevaluasi aproksimasi adalah dengan menghitung galat (error) antara nilai aktual dan nilai hasil prediksi model. Galat ini dapat diukur dalam berbagai bentuk, yang paling mendasar adalah galat absolut $|yi−y^i|$ dan galat kuadrat $(y_i−y_i)^2$, di mana y_i adalah nilai aktual dan y^i adalah hasil prediksi. Dari sini, beberapa metrik evaluasi dapat diturunkan, seperti Mean Absolute Error (MAE), yang memberikan ukuran rata-rata kesalahan absolut, dan Root Mean Squared Error (RMSE), yang mengkuadratkan kesalahan terlebih dahulu sebelum dirata-rata, sehingga memberikan penalti lebih besar pada kesalahan besar.

Salah satu metrik yang paling banyak digunakan dalam evaluasi regresi adalah koefisien determinasi (R^2), yang mengukur proporsi variabilitas data yang dapat dijelaskan oleh model aproksimasi. Nilai R^2 berkisar dari 0 hingga 1, di mana nilai mendekati 1 menunjukkan bahwa model menjelaskan hampir seluruh variasi dalam data, sedangkan nilai mendekati 0 menunjukkan bahwa model kurang efektif dalam menjelaskan data. Namun, meskipun R^2 berguna, ia bisa menyesatkan jika digunakan tanpa memperhatikan kompleksitas model. Model yang

terlalu kompleks bisa memiliki R^2 tinggi tetapi sebenarnya mengalami overfitting, yaitu menyesuaikan diri secara berlebihan dengan data pelatihan hingga gagal bekerja dengan baik pada data baru.

Evaluasi aproksimasi juga melibatkan analisis residual, yaitu perbedaan antara nilai aktual dan nilai prediksi model. Pola residual yang acak mengindikasikan bahwa model telah menangkap struktur data dengan baik, sedangkan pola sistematis (misalnya membentuk kurva atau tren) menunjukkan bahwa model belum cukup baik. Visualisasi residual dapat memperjelas hal ini dan membantu dalam diagnosis model.

Pada praktik profesional, evaluasi biasanya tidak dilakukan hanya dengan satu metrik. Kombinasi antara MAE, RMSE, R^2, dan analisis residual memberikan gambaran yang lebih menyeluruh tentang kualitas aproksimasi. Evaluasi ini juga sangat penting dalam pemilihan model terbaik dari beberapa alternatif, penyesuaian parameter model, dan validasi hasil sebelum digunakan untuk prediksi atau pengambilan keputusan. Oleh karena itu, evaluasi aproksimasi merupakan langkah kritis dalam siklus pemodelan numerik dan statistik yang tidak boleh diabaikan.

3. Visualisasi Residual dan Diagnostik

Visualisasi residual dan diagnostik merupakan aspek penting dalam evaluasi model aproksimasi, khususnya dalam konteks regresi numerik dan statistik. Residual adalah selisih antara nilai sebenarnya dari data (y_i) dengan nilai yang diprediksi oleh model aproksimasi (y^i), yakni $r_i = y_i - y^i$. Analisis terhadap residual memberikan wawasan mendalam tentang seberapa baik model menangkap pola hubungan dalam data. Visualisasi residual bertujuan untuk mengidentifikasi apakah galat (kesalahan prediksi) terdistribusi secara acak, atau justru membentuk pola tertentu yang mengindikasikan kelemahan model. Ketika model aproksimasi dinilai hanya berdasarkan metrik seperti RMSE atau R^2, kita bisa saja melewatkan masalah penting yang tersembunyi, seperti non-linearitas, heteroskedastisitas, atau autokorelasi dan semua ini bisa terdeteksi melalui analisis grafis terhadap residual.

Menurut Montgomery, Peck, dan Vining (2012) dalam *Introduction to Linear Regression Analysis*, pola residual yang baik harus menyerupai sebaran acak yang simetris di sekitar garis nol. Ini menunjukkan bahwa model sudah cukup baik dalam menangkap struktur

data, dan sisa galat hanyalah fluktuasi acak (*noise*). Namun, jika residual menunjukkan pola melengkung, berbentuk U atau terdistribusi asimetris, hal itu menandakan bahwa model belum cukup baik mungkin karena hubungan antar variabel bersifat nonlinier, namun model yang digunakan hanya linier. Dalam regresi polinomial, visualisasi residual dapat membantu memilih orde polinomial yang tepat. Jika residual melengkung, maka kemungkinan besar derajat polinomial masih terlalu rendah.

Jenis plot residual yang paling umum adalah plot residual terhadap nilai prediksi (y) dan plot residual terhadap variabel independen (x). Pada kedua plot ini, idealnya residual tersebar acak di sekitar garis nol tanpa pola yang jelas. Pola yang berbentuk kipas (menyempit atau melebar) menunjukkan adanya heteroskedastisitas, yaitu varian galat yang tidak konstan. Ini menjadi masalah serius dalam regresi klasik karena melanggar asumsi homoskedastisitas dan dapat membuat estimasi varians tidak akurat.

Visualisasi diagnostik lain termasuk *normal probability plot* (Q-Q plot), yang digunakan untuk mengevaluasi apakah residual berdistribusi normal. Dalam regresi linear klasik, normalitas residual diperlukan untuk validitas uji statistik seperti t-test dan F-test. Jika titik-titik dalam Q-Q plot menyimpang jauh dari garis diagonal, maka residual tidak normal dan model tidak memenuhi asumsi klasik. Selain itu, histogram residual dapat digunakan untuk evaluasi visual distribusi residual secara langsung.

Dengan melakukan visualisasi residual, pengguna dapat memahami lebih dalam mengapa suatu model bekerja dengan baik atau buruk, dan apakah perlu dilakukan transformasi data, penambahan variabel, atau penggantian bentuk model. Oleh karena itu, visualisasi residual bukan hanya alat bantu, tetapi merupakan bagian integral dari proses diagnosis dan validasi model aproksimasi numerik maupun statistik yang andal.

4. Evaluasi Kinerja Model pada Data Baru

Evaluasi kinerja model pada data baru merupakan langkah penting dalam proses validasi model aproksimasi atau prediktif untuk memastikan bahwa model yang telah dibangun tidak hanya cocok dengan data pelatihan (*training data*), tetapi juga memiliki kemampuan generalisasi yang baik terhadap data yang belum pernah dilihat

sebelumnya. Dalam praktik nyata, tujuan utama dari pembangunan model aproksimasi bukanlah hanya untuk menyesuaikan model terhadap data historis, melainkan untuk digunakan dalam prediksi dan estimasi nilai di masa depan atau dalam konteks yang berbeda. Oleh karena itu, evaluasi model pada data baru sangat krusial untuk menilai apakah model bersifat overfit, underfit, atau benar-benar mampu menangkap pola dasar dari hubungan antar variabel.

Menurut Hastie, Tibshirani, dan Friedman (2009) dalam *The Elements of Statistical Learning*, overfitting adalah situasi ketika model sangat akurat pada data pelatihan, tetapi berkinerja buruk pada data baru karena model terlalu kompleks dan menangkap noise atau fluktuasi acak dalam data, bukan pola yang mendasarinya. Sebaliknya, underfitting terjadi jika model terlalu sederhana sehingga gagal menangkap pola penting bahkan dalam data pelatihan. Kedua kondisi ini bisa sulit dikenali jika hanya mengevaluasi model berdasarkan data yang sama dengan yang digunakan untuk melatihnya. Untuk itu, evaluasi pada data baru yang tidak digunakan dalam proses pelatihan menjadi keharusan dalam penilaian performa model secara menyeluruh.

Salah satu pendekatan yang paling umum adalah dengan membagi data menjadi dua bagian: training set dan testing set. Model dibangun menggunakan training set, lalu performanya diuji pada testing set. Evaluasi dilakukan dengan menghitung metrik seperti *Root Mean Squared Error* (RMSE), *Mean Absolute Error* (MAE), atau R-squared pada data uji. Jika nilai error pada data uji jauh lebih besar dibandingkan dengan data pelatihan, maka model cenderung overfit. Sebaliknya, jika error besar pada kedua data, maka kemungkinan besar model underfit.

Pada situasi dengan data terbatas, evaluasi model dapat dilakukan dengan metode *cross-validation*, di mana data dibagi menjadi beberapa subset (fold), dan model dilatih serta diuji secara bergilir di seluruh subset. Teknik ini, terutama k-fold *cross-validation*, sangat efektif dalam memberikan estimasi performa model yang lebih stabil dan bebas dari pengaruh pemisahan data yang kebetulan tidak representatif. Selain itu, evaluasi pada data baru juga mencakup pengamatan terhadap distribusi residual, robustness model terhadap noise, serta kemampuan menangani data ekstrem atau outlier. Hal ini sangat penting dalam aplikasi dunia nyata di mana data masa depan tidak selalu bersih atau konsisten dengan pola historis. Model yang hanya diuji pada data pelatihan cenderung

menyesatkan karena tidak mencerminkan kondisi operasional sesungguhnya.

Dengan demikian, evaluasi kinerja model pada data baru adalah langkah kritis dalam menjamin keandalan dan kegunaan praktis suatu model. Model yang baik bukan hanya yang cocok dengan data masa lalu, tetapi juga yang mampu memprediksi dengan akurat dalam kondisi dan situasi yang bervariasi. Evaluasi yang menyeluruh ini menjadi kunci sukses dalam penerapan model numerik, statistik, maupun *machine learning* di berbagai bidang aplikasi.

BAB VI
DIFERENSIASI DAN INTEGRASI NUMERIK

Diferensiasi dan Integrasi Numerik, yang menjadi landasan dalam menyelesaikan berbagai persoalan matematika terapan dan teknik. Di dunia nyata, fungsi-fungsi yang digunakan tidak selalu memiliki bentuk analitik yang sederhana atau mudah diturunkan secara simbolik, sehingga pendekatan numerik menjadi solusi efektif dalam memperkirakan turunan maupun integral dari suatu fungsi. Dalam bab ini, pembaca akan dikenalkan pada berbagai metode diferensiasi numerik seperti metode selisih maju, mundur, dan tengah, yang digunakan untuk menghitung laju perubahan fungsi secara mendekati. Selain itu, metode integrasi numerik seperti metode Trapezoid, Simpson, dan teknik Romberg akan dijelaskan secara rinci, lengkap dengan pembahasan galat (error) dan kestabilan perhitungan. Topik ini tidak hanya penting secara teoritis, tetapi juga memiliki banyak aplikasi praktis seperti dalam pemrosesan sinyal, perhitungan energi dalam sistem mekanik, serta estimasi luas dan volume dalam berbagai konteks sains dan rekayasa. Dengan memahami konsep dan teknik diferensiasi serta integrasi numerik, pembaca diharapkan mampu mengembangkan solusi numerik yang akurat, efisien, dan aplikatif untuk berbagai permasalahan yang kompleks dan dinamis.

A. Metode Selisih Hingga (*Finite Difference*)

Metode selisih hingga (*finite difference method*, FDM) merupakan salah satu pendekatan numerik yang paling umum digunakan untuk menyelesaikan persoalan matematika, terutama dalam hal diferensiasi, integrasi, serta penyelesaian persamaan diferensial biasa

(ODE) dan persamaan diferensial parsial (PDE). Konsep dasar metode ini adalah menggantikan turunan fungsi kontinu dengan pendekatan diskrit menggunakan nilai-nilai fungsi pada titik-titik diskrit dalam domain tertentu.

Burden dan Faires (2011) dalam *Numerical Analysis*, metode selisih hingga bekerja dengan cara mendekati nilai turunan suatu fungsi menggunakan selisih nilai fungsi pada titik-titik yang berjarak tertentu. Dalam bentuk yang paling sederhana, jika kita memiliki fungsi kontinu $f(x)$, maka turunan pertama di titik x dapat dihampiri dengan selisih antara dua nilai $f(x+h)$ dan $f(x)$, dibagi dengan h, yaitu:

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

Ini dikenal sebagai metode selisih maju (*forward difference*). Nilai h merupakan langkah diskret (*step size*), dan pendekatan ini menjadi lebih akurat seiring semakin kecilnya nilai h, meskipun efek galat pembulatan bisa menjadi signifikan jika h terlalu kecil.

Gunakan **metode selisih hingga** untuk menyelesaikan persamaan diferensial berikut:

$$\frac{d^2y}{dx^2} = -2, \quad \text{untuk } 0 \le x \le 1$$

Dengan syarat batas:

$$y(0) = 0, \quad y(1) = 0$$

Gunakan 3 titik (termasuk titik batas) untuk menghitung nilai pendekatan y pada titik tengah, yaitu x=0.5, dengan metode beda hingga orde 2.

Jawaban:

Karena terdapat 3 titik: $x_0 = 0$, $x_1 = 0.5$, dan $x_2 = 1$, maka selang $h = \frac{1}{2} = 0.5$.

Gunakan pendekatan selisih hingga untuk turunan kedua:

$$\frac{d^2y}{dx^2} \approx \frac{y_{i-1} - 2y_i + y_{i+1}}{h^2}$$

Untuk $x = 0.5$ (titik tengah), indeks $i = 1$. Maka:

$$\frac{y_0 - 2y_1 + y_2}{h^2} = -2$$

Substitusi nilai:

- $y_0 = 0$ (dari syarat batas)
- $y_2 = 0$ (dari syarat batas)
- $h = 0.5$

$$\frac{0 - 2y_1 + 0}{(0.5)^2} = -2 \Rightarrow \frac{-2y_1}{0.25} = -2 \Rightarrow -8y_1 = -2 \Rightarrow y_1 = \frac{1}{4} = 0.25$$

1. Jenis-Jenis Selisih Hingga

Metode selisih hingga (*finite difference*) adalah teknik numerik yang digunakan untuk menghampiri turunan suatu fungsi menggunakan pendekatan diskrit. Metode ini memanfaatkan nilai-nilai fungsi pada titik-titik tertentu yang berjarak tetap dalam domain yang telah dibagi menjadi grid atau titik diskrit. Dalam praktik komputasi, turunan eksak suatu fungsi seringkali tidak dapat dihitung secara simbolik, terutama jika fungsi tersebut hanya diketahui dalam bentuk data atau hasil pengukuran. Oleh karena itu, pendekatan selisih hingga menjadi solusi efektif dalam memperkirakan turunan-turunan tersebut secara numerik. Terdapat tiga jenis utama selisih hingga yang umum digunakan dalam komputasi numerik, yaitu selisih maju (*forward difference*), selisih mundur (*backward difference*), dan selisih tengah (*central difference*). Masing-masing metode memiliki keunggulan, keterbatasan, dan akurasi yang berbeda, serta digunakan dalam konteks aplikasi yang spesifik.

Jenis pertama adalah selisih maju (*forward difference*). Metode ini memperkirakan turunan suatu fungsi pada titik x dengan memanfaatkan nilai fungsi di titik tersebut dan titik sesudahnya, yaitu $x+h$, di mana h adalah panjang langkah diskrit atau grid spacing. Rumus matematisnya adalah:

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

Metode ini tergolong sederhana dan mudah diterapkan karena hanya memerlukan nilai fungsi pada dua titik berurutan. Namun, selisih maju memiliki tingkat akurasi yang lebih rendah dibanding metode lain karena memiliki galat truncation orde pertama ($O(h)$). Galat truncation terjadi karena pendekatan yang digunakan hanya merepresentasikan sebagian dari deret Taylor, sehingga hasil yang diperoleh menjadi kurang akurat jika h tidak cukup kecil. Metode ini umumnya digunakan ketika informasi nilai fungsi hanya tersedia mulai dari titik tertentu ke depan,

seperti dalam kasus proses waktu berjalan ke arah positif, atau pada domain batas awal.

Jenis kedua adalah selisih mundur (*backward difference*). Pendekatan ini memperkirakan turunan dengan menggunakan nilai fungsi pada titik x dan titik sebelumnya, yaitu $x-h$. Rumusnya dituliskan sebagai:

$$f'(x) \approx \frac{f(x) - f(x-h)}{h}$$

Selisih mundur juga memiliki galat truncation orde pertama, sehingga tingkat akurasinya setara dengan selisih maju. Metode ini cocok digunakan pada kasus di mana data hanya tersedia dari akhir domain ke belakang, atau pada titik-titik batas akhir suatu interval, terutama saat menangani kondisi batas dalam simulasi numerik. Dalam konteks tertentu seperti pemodelan proses historis atau simulasi numerik berbasis waktu mundur, selisih mundur menjadi pilihan utama karena bentuknya yang mempertimbangkan titik-titik masa lalu.

Jenis ketiga dan yang paling banyak digunakan adalah selisih tengah (*central difference*). Metode ini memanfaatkan nilai fungsi pada dua titik yang simetris terhadap titik x, yaitu $x-h$ dan $x+h$. Pendekatan ini dituliskan sebagai:

$$f'(x) \approx \frac{f(x+h) - f(x-h)}{2h}$$

Metode ini memiliki galat truncation orde kedua ($O(h2)$)), sehingga jauh lebih akurat dibandingkan dengan selisih maju dan selisih mundur, khususnya jika h tidak terlalu kecil. Karena nilai turunan dihitung berdasarkan rata-rata perubahan ke depan dan ke belakang, maka pendekatan ini lebih stabil dan lebih baik dalam banyak kasus analisis numerik. Central difference banyak diterapkan dalam simulasi fisika, mekanika fluida, dan teknik karena mampu memberikan keseimbangan antara akurasi dan efisiensi komputasi.

Metode selisih hingga juga dikembangkan untuk menghampiri turunan orde lebih tinggi, seperti turunan kedua. Untuk turunan kedua $f''(x)$, pendekatan selisih tengah dituliskan sebagai:

$$f''(x) \approx \frac{f(x+h) - 2f(x) + f(x-h)}{h^2}$$

Turunan kedua sangat penting dalam pemodelan fisik, seperti dalam penyelesaian persamaan diferensial parsial yang melibatkan fluks, perpindahan panas, atau percepatan. Akurasi metode ini tetap tinggi karena menggunakan pendekatan simetris yang mengurangi pengaruh kesalahan lokal.

Beberapa variasi lain dari metode selisih hingga juga tersedia, seperti metode selisih hingga tak seragam (*non-uniform finite difference*), di mana jarak antar titik h tidak konstan, serta pendekatan selisih orde lebih tinggi (*higher-order difference*), yang menggunakan lebih banyak titik dan menghasilkan akurasi lebih tinggi namun dengan kompleksitas komputasi yang lebih besar.

Pada praktiknya, pemilihan jenis selisih hingga bergantung pada beberapa faktor: kondisi batas domain, distribusi data fungsi, kebutuhan akurasi, dan efisiensi perhitungan. Misalnya, jika kita memiliki data eksperimental yang hanya mencakup nilai dari satu sisi domain, maka pendekatan selisih maju atau mundur lebih sesuai. Sebaliknya, untuk fungsi yang didefinisikan secara lengkap di sekitar titik evaluasi, selisih tengah lebih disarankan karena ketelitiannya.

2. Galat dan Akurasi

Pada komputasi numerik, dua konsep fundamental yang sangat menentukan kualitas hasil perhitungan adalah galat (*error*) dan akurasi (*accuracy*). Keduanya tidak dapat dipisahkan dan saling berkaitan dalam proses evaluasi hasil numerik. Galat menggambarkan sejauh mana hasil perhitungan menyimpang dari nilai sebenarnya (eksak), sedangkan akurasi menunjukkan tingkat kedekatan hasil komputasi terhadap nilai tersebut. Memahami asal-usul, jenis, dan dampak galat sangat penting agar seseorang tidak hanya dapat menilai kualitas solusi numerik, tetapi juga mengambil langkah-langkah korektif untuk memperbaikinya.

Secara umum, galat dalam komputasi numerik dapat diklasifikasikan ke dalam beberapa jenis utama, yaitu galat pembulatan (*round-off error*), galat pemotongan (*truncation error*), galat propagasi (*propagation error*), serta galat total (*total error*) yang merupakan kombinasi dari beberapa sumber galat. Galat pembulatan terjadi karena komputer tidak dapat menyimpan angka real secara presisi tak hingga. Sebagian besar komputer menggunakan standar bilangan floating-point (misalnya IEEE 754) yang hanya mampu merepresentasikan sejumlah digit tertentu. Ketika angka tidak dapat direpresentasikan secara tepat,

maka sistem akan membulatkan angka tersebut ke representasi terdekat, menyebabkan terjadinya galat pembulatan. Galat ini sering kali sangat kecil, namun jika dikombinasikan dalam perhitungan berulang seperti iterasi atau integrasi numerik, dampaknya dapat menjadi signifikan.

Galat pemotongan merupakan galat yang muncul ketika pendekatan numerik digunakan untuk menggantikan prosedur matematika yang ideal, seperti deret tak hingga atau turunan eksak. Misalnya, ketika metode selisih hingga digunakan untuk menghampiri turunan suatu fungsi, nilai sebenarnya dari turunan tersebut digantikan dengan bentuk diskrit yang hanya mempertimbangkan sebagian dari deret Taylor. Ketidaksesuaian ini menghasilkan galat pemotongan. Jenis galat ini sangat bergantung pada ukuran langkah diskret (misalnya h dalam metode selisih hingga): semakin kecil h, semakin kecil pula galat pemotongan, meskipun pada titik tertentu galat pembulatan akan mulai meningkat dan mendominasi, menyebabkan kehilangan presisi.

Galat propagasi terjadi ketika galat yang muncul pada tahap awal perhitungan terbawa dan diperbesar pada tahap-tahap selanjutnya. Proses propagasi ini sering kali terjadi dalam metode iteratif atau pada penyelesaian sistem persamaan diferensial numerik. Jika suatu metode numerik tidak stabil (unstable), galat kecil yang tidak signifikan dapat berkembang secara eksponensial dan menghasilkan solusi yang sama sekali tidak representatif terhadap kenyataan. Oleh karena itu, penting untuk melakukan analisis kestabilan terhadap metode numerik yang digunakan.

Pada praktiknya, galat total adalah gabungan dari semua bentuk galat di atas. Meskipun dalam teori kita bisa membahas jenis-jenis galat secara terpisah, dalam aplikasi nyata, galat-galat ini muncul secara simultan dan berinteraksi satu sama lain. Oleh karena itu, pendekatan komputasi numerik yang baik adalah yang mampu meminimalkan galat total, baik melalui pemilihan metode yang tepat, pengaturan parameter langkah diskret yang optimal, maupun pemrosesan data dengan presisi tinggi.

Berbicara tentang akurasi, konsep ini menjelaskan seberapa dekat hasil numerik terhadap solusi eksak yang sesungguhnya. Akurasi dapat dinyatakan secara absolut maupun relatif. Galat absolut didefinisikan sebagai selisih antara nilai hasil komputasi dan nilai eksak:

$$\text{Galat absolut} = |x_{\text{numerik}} - x_{\text{eksak}}|$$

Sedangkan **galat relatif** merupakan perbandingan antara galat absolut dan nilai eksak:

$$\text{Galat relatif} = \frac{|x_{\text{numerik}} - x_{\text{eksak}}|}{|x_{\text{eksak}}|}$$

Pada banyak kasus, galat relatif lebih berguna karena memberikan gambaran proporsional terhadap besarnya kesalahan terhadap nilai yang dihitung. Misalnya, galat absolut sebesar 0.01 pada nilai eksak 1000 mungkin dapat diabaikan, namun galat yang sama pada nilai eksak 0.01 bisa menjadi sangat signifikan.

Untuk mengevaluasi kualitas dan akurasi suatu metode numerik, konsep orde konvergensi atau orde galat sering digunakan. Orde ini menunjukkan seberapa cepat galat menyusut ketika ukuran langkah diskret dikurangi. Misalnya, metode dengan galat $O(h)$ dikatakan memiliki akurasi orde satu, dan metode dengan galat $O(h2)$ memiliki akurasi orde dua. Hal ini berarti jika kita mengurangi h menjadi setengahnya, galat pada metode orde satu akan berkurang separuh, sementara pada metode orde dua galat akan berkurang seperempat. Dengan demikian, orde galat memberikan ukuran kuantitatif dari efisiensi suatu metode dalam mencapai hasil yang akurat.

Akurasi tinggi tidak selalu menjamin hasil yang benar, terutama jika metode tersebut tidak stabil atau jika galat pembulatan mendominasi. Oleh karena itu, dalam banyak kasus numerik, dibutuhkan keseimbangan antara akurasi, stabilitas, dan efisiensi komputasi. Strategi praktis dalam menghadapi isu galat dan akurasi mencakup pemilihan algoritma numerik yang sesuai dengan jenis masalah, pengujian hasil dengan menggunakan ukuran langkah yang berbeda (*refinement*), serta verifikasi terhadap solusi analitik (jika tersedia) atau solusi numerik yang sudah terverifikasi.

Pemahaman menyeluruh tentang galat dan akurasi merupakan fondasi dalam komputasi numerik yang andal. Meskipun galat tidak dapat dihindari dalam setiap perhitungan numerik, pengelolaan yang tepat terhadap sumber-sumber galat akan sangat menentukan seberapa efektif suatu metode numerik dalam memodelkan fenomena nyata secara kuantitatif. Evaluasi akurasi tidak hanya bersifat matematis, tetapi juga

menjadi instrumen penting dalam menjembatani kesenjangan antara teori matematika dan penerapan teknis dalam dunia sains dan rekayasa.

B. Metode Trapezoid, Simpson, dan Romberg

Pada komputasi numerik, integrasi numerik adalah teknik penting yang digunakan untuk menghitung luas di bawah kurva atau integral dari fungsi yang tidak dapat dihitung secara analitik. Ketika fungsi tidak memiliki bentuk antiturunan yang diketahui, atau ketika hanya tersedia dalam bentuk data diskrit (seperti hasil eksperimen), maka pendekatan numerik menjadi solusi utama. Tiga metode yang paling dikenal dan banyak digunakan dalam integrasi numerik adalah metode Trapezoid, metode Simpson, dan metode Romberg. Ketiganya menawarkan pendekatan yang berbeda dalam mendekati nilai integral suatu fungsi dan masing-masing memiliki kelebihan serta batasan tergantung pada konteks penggunaannya.

1. Metode Trapezoid

Metode trapezoid adalah salah satu teknik dasar dalam integrasi numerik yang digunakan untuk menghitung aproksimasi nilai integral tentu dari suatu fungsi yang kontinu. Dalam banyak kasus praktis, fungsi yang ingin diintegrasikan tidak memiliki bentuk antiturunan yang diketahui atau terlalu kompleks untuk diselesaikan secara simbolik. Oleh karena itu, metode numerik seperti metode trapezoid menjadi solusi yang efisien dan relatif mudah diterapkan. Nama "trapezoid" berasal dari cara pendekatan yang digunakan, yaitu memperkirakan luas di bawah kurva fungsi sebagai jumlah dari luas beberapa trapesium yang dibentuk oleh segmen-segmen garis lurus antara titik-titik evaluasi fungsi.

Secara matematis, jika $f(x)$ adalah fungsi kontinu pada interval $[a,b]$, maka integral tentu $\int abf(x)\,dx$ dapat dihampiri menggunakan metode trapezoid sederhana sebagai berikut:

$$\int_a^b f(x)\,dx \approx \frac{h}{2}\left[f(a) + f(b)\right]$$

di mana h = b - a adalah panjang interval. Rumus ini sebenarnya merupakan pendekatan yang sangat kasar karena hanya menggunakan dua titik, titik awal dan akhir, dan menghubungkannya dengan garis

lurus. Luas di bawah kurva di antara kedua titik itu kemudian diaproksimasi sebagai luas sebuah trapesium. Oleh karena itu, metode ini akan memberikan hasil yang cukup baik hanya jika fungsi f(x) hampir linear di antara a dan b.

Untuk meningkatkan akurasi, digunakan metode trapezoid komposit, yaitu dengan membagi interval [a,b][a, b][a,b] menjadi nnn subinterval yang sama panjang. Setiap subinterval dihitung luasnya menggunakan pendekatan trapesium, kemudian dijumlahkan seluruhnya. Formula komposit metode trapezoid dinyatakan sebagai:

$$\int_a^b f(x)\,dx \approx \frac{h}{2}\left[f(x_0) + 2\sum_{i=1}^{n-1} f(x_i) + f(x_n)\right]$$

Dalam pendekatan ini, kita membentuk nnn trapesium yang masing-masing mencakup dua titik evaluasi, dan menjumlahkan luasnya secara keseluruhan. Nilai fungsi di titik-titik tengah dikalikan dua karena merupakan titik yang digunakan dua kali dalam perhitungan luas dua trapesium yang bersebelahan.

Menurut Burden dan Faires (2011), metode trapezoid memiliki orde akurasi kedua ($O(h^2)$), yang berarti jika panjang langkah hhh dibagi dua, maka galat (kesalahan aproksimasi) akan berkurang hingga seperempat. Galat dari metode ini dapat diperkirakan dengan rumus:

$$E_T = -\frac{(b-a)^3}{12n^2} f''(\xi)$$

untuk suatu $\xi \in [a,b]$, yang menunjukkan bahwa galat tergantung pada nilai turunan kedua dari fungsi f(x). Oleh karena itu, metode trapezoid akan lebih akurat jika fungsi yang diintegrasikan memiliki turunan kedua yang kecil atau mendekati nol (yakni, mendekati linear). Namun, jika fungsi memiliki kelengkungan yang tajam, metode ini akan menghasilkan galat yang cukup signifikan.

Salah satu keunggulan utama metode trapezoid adalah kesederhanaannya. Karena hanya melibatkan operasi dasar (penjumlahan dan perkalian skalar), metode ini sangat mudah diimplementasikan dalam pemrograman. Bahkan, banyak kalkulator ilmiah dan perangkat lunak spreadsheet seperti Excel menyediakan fungsi bawaan untuk metode trapezoid, menjadikannya sangat praktis untuk analisis numerik cepat. Dalam bahasa pemrograman seperti

Python, MATLAB, atau C++, metode ini juga menjadi dasar untuk algoritma integrasi numerik yang lebih kompleks.

Keterbatasan metode trapezoid tetap perlu diperhatikan. Salah satu kelemahan terbesarnya adalah kecenderungan untuk kurang akurat ketika diterapkan pada fungsi yang sangat melengkung, osilatif, atau diskontinu. Dalam kasus seperti itu, hasil aproksimasi bisa menyimpang jauh dari nilai eksak. Untuk meningkatkan ketelitian tanpa harus mengurangi panjang langkah secara ekstrem (yang dapat meningkatkan beban komputasi), biasanya digunakan metode numerik dengan orde lebih tinggi seperti metode Simpson atau Romberg.

Pada konteks tertentu, metode trapezoid justru menjadi pilihan utama. Misalnya, dalam pengolahan sinyal atau data eksperimental di mana nilai fungsi hanya diketahui pada titik-titik tertentu secara diskrit (tanpa bentuk fungsional eksplisit), metode trapezoid menjadi pendekatan praktis yang dapat langsung diterapkan. Demikian pula dalam integrasi pada domain waktu riil dalam sistem tertanam (embedded systems), pendekatan berbasis trapezoid sering digunakan karena kecepatan dan ringannya komputasi.

Pada aplikasi dunia nyata, metode trapezoid digunakan dalam berbagai bidang, seperti fisika (untuk menghitung kerja mekanik dari grafik gaya vs. perpindahan), ekonomi (untuk estimasi nilai rata-rata fungsi permintaan), biologi (untuk menghitung area di bawah kurva pertumbuhan), dan teknik (untuk menghitung energi listrik berdasarkan tegangan dan arus). Meski sederhana, metode ini tetap relevan karena fleksibilitasnya untuk digunakan dalam beragam kondisi praktis.

Hitung pendekatan nilai integral berikut menggunakan metode trapezoid:

$$\int_1^3 (x^2 + 1)\, dx$$

- Batas bawah: $a = 1$
- Batas atas: $b = 3$
- Jumlah subinterval: $n = 2$
- Lebar subinterval:

$$h = \frac{b-a}{n} = \frac{3-1}{2} = 1$$

Titik-titiknya:

$$x_0 = 1, \quad x_1 = 2, \quad x_2 = 3$$

Maka,

$$f(x) = x^2 + 1$$

$$f(x_0) = f(1) = 1^2 + 1 = 2$$
$$f(x_1) = f(2) = 2^2 + 1 = 5$$
$$f(x_2) = f(3) = 3^2 + 1 = 10$$

Maka,

$$\int_a^b f(x)\,dx \approx \frac{h}{2}\left[f(x_0) + 2f(x_1) + f(x_2)\right]$$

$$= \frac{1}{2}\left[2 + 2(5) + 10\right] = \frac{1}{2}(2 + 10 + 10) = \frac{1}{2} \cdot 22 = 11$$

Dengan metode trapezoid dan 2 subinterval, nilai pendekatan dari:

$$\int_1^3 (x^2 + 1)\,dx \approx 11$$

Nilai eksak dari integral tersebut adalah:

$$\left[\frac{x^3}{3} + x\right]_1^3 = (9 + 3) - \left(\frac{1}{3} + 1\right) = 12 - \frac{4}{3} = \frac{32}{3} \approx 10.67$$

2. Metode Simpson

Metode Simpson adalah salah satu teknik integrasi numerik yang sangat terkenal dan banyak digunakan karena menawarkan keseimbangan yang baik antara akurasi dan efisiensi komputasi. Metode ini merupakan pendekatan untuk menghitung integral tentu dari suatu fungsi dengan menggunakan interpolasi polinomial kuadrat (parabola) sebagai pendekatan lokal terhadap fungsi yang diintegrasikan. Ketimbang menggunakan garis lurus seperti pada metode trapezoid, metode Simpson menggunakan segmen parabola untuk mendekati kurva fungsi, sehingga menghasilkan estimasi luas di bawah kurva yang jauh lebih akurat, terutama untuk fungsi-fungsi yang halus dan melengkung.

Menurut Chapra dan Canale (2015) dalam buku *Numerical Methods for Engineers,* metode Simpson diperoleh dengan mengambil tiga titik pada fungsi yang ingin diintegrasikan, yaitu titik awal a, titik tengah m, dan titik akhir b, lalu membentuk fungsi polinomial kuadrat yang melewati ketiga titik tersebut. Integral dari fungsi asli kemudian dihampiri dengan integral dari polinomial tersebut. Rumus dasar metode ini, yang dikenal sebagai Simpson's 1/3 Rule, adalah sebagai berikut:

$$\int_a^b f(x)\,dx \approx \frac{b-a}{6}\left[f(a) + 4f\left(\frac{a+b}{2}\right) + f(b)\right]$$

Untuk fungsi yang lebih kompleks atau interval yang lebih luas, metode ini diperluas menjadi bentuk Simpson Komposit, di mana interval $[a,b]$ dibagi menjadi sejumlah genap n subinterval yang sama panjang ($\mathrm{h} = \frac{\mathrm{b-a}}{\mathrm{n}}$), dan integral dihitung sebagai jumlah dari integral beberapa segmen parabola. Rumus Simpson Komposit adalah:

$$\int_a^b f(x)\,dx \approx \frac{h}{3}\left[f(x_0) + 4\sum_{\text{ganjil}} f(x_i) + 2\sum_{\text{genap}} f(x_i) + f(x_n)\right]$$

Fungsi pada indeks ganjil dikalikan empat karena menjadi titik tengah segmen parabola, sedangkan fungsi pada indeks genap (kecuali titik awal dan akhir) dikalikan dua karena menjadi titik sambungan antar parabola.

Keunggulan utama dari metode Simpson terletak pada orde akurasi keempat (O(h4)). Ini berarti jika panjang langkah hhh dibagi dua,

maka galat aproksimasi akan berkurang dari nilai sebelumnya. Galat truncation (pemotongan) metode Simpson diberikan oleh:

$$E_S = -\frac{(b-a)^5}{180n^4} f^{(4)}(\xi)$$

untuk suatu ξ∈[a,b], yang menunjukkan bahwa galat bergantung pada turunan keempat dari fungsi f(x). Oleh karena itu, metode Simpson akan menghasilkan hasil yang sangat akurat untuk fungsi yang halus dan tidak memiliki perubahan mendadak pada kelengkungan.

Metode Simpson memiliki keterbatasan, yaitu jumlah subinterval harus genap. Jika jumlah subinterval ganjil, maka metode ini tidak dapat langsung diterapkan secara penuh. Untuk mengatasi masalah ini, kadang digunakan gabungan antara Simpson's 1/3 Rule dan Simpson's 3/8 Rule, yang merupakan varian lain dari metode Simpson yang menggunakan empat titik (tiga subinterval). Simpson 3/8 Rule memiliki rumus:

$$\int_a^b f(x)\,dx \approx \frac{3h}{8}\left[f(x_0) + 3f(x_1) + 3f(x_2) + f(x_3)\right]$$

Meski jarang digunakan secara keseluruhan, aturan 3/8 berguna dalam menutupi sisa interval ketika jumlah subinterval tidak bisa dibagi rata untuk aturan 1/3.

Dari segi implementasi, metode Simpson sangat mudah diprogram menggunakan bahasa komputasi seperti Python, MATLAB, atau C++. Dalam praktiknya, pendekatan ini sering digunakan dalam berbagai aplikasi sains dan teknik seperti simulasi fisika, perhitungan energi, volume fluida, analisis struktur, dan bahkan ekonomi dan biologi. Misalnya, untuk menghitung total konsumsi energi berdasarkan kurva daya terhadap waktu, metode Simpson dapat digunakan untuk menghasilkan estimasi numerik dengan presisi tinggi.

Metode Simpson juga sangat bermanfaat dalam kasus di mana fungsi hanya tersedia dalam bentuk data diskrit dari pengukuran eksperimen. Dalam hal ini, interpolasi lokal berbasis parabola memungkinkan kita mendekati integral meskipun tidak memiliki bentuk fungsi eksplisit. Ini menjadikan metode Simpson sebagai alat yang sangat fleksibel dan berguna dalam analisis data real-world.

Akurasi metode Simpson bergantung pada asumsi bahwa fungsi f(x) dapat diaproksimasi dengan baik oleh polinomial kuadrat pada setiap subinterval. Jika fungsi tersebut sangat tidak mulus, osilatif, atau memiliki diskontinuitas, maka hasil integrasi bisa menjadi tidak akurat.

Dalam kasus seperti itu, solusi yang lebih baik mungkin menggunakan pendekatan orde lebih tinggi atau teknik adaptif seperti metode Romberg atau kuadratur Gauss.

Hitung pendekatan dari integral berikut menggunakan **metode Simpson 1/3:**

$$\int_0^4 (x^2 + 2)\, dx$$

- Batas bawah: $a = 0$
- Batas atas: $b = 4$
- Jumlah subinterval: $n = 2$ → harus **genap** (sesuai syarat metode Simpson)
- Panjang subinterval:

$$h = \frac{b - a}{n} = \frac{4 - 0}{2} = 2$$

Titik-titik:

$$x_0 = 0, \quad x_1 = 2, \quad x_2 = 4$$

Maka,

$$f(x) = x^2 + 2$$

$$f(x_0) = f(0) = 0^2 + 2 = 2$$
$$f(x_1) = f(2) = 2^2 + 2 = 6$$
$$f(x_2) = f(4) = 4^2 + 2 = 18$$

Maka,

$$\int_a^b f(x)\, dx \approx \frac{h}{3} [f(x_0) + 4f(x_1) + f(x_2)]$$

$$= \frac{2}{3} [2 + 4(6) + 18] = \frac{2}{3}(44) = \frac{88}{3} \approx 29.33$$

Dengan metode Simpson 1/3 dan 2 subinterval, diperoleh pendekatan:

$$\int_0^4 (x^2 + 2)\, dx \approx 29.33$$

Sementara nilai eksaknya adalah:

$$\int_0^4 (x^2 + 2)\, dx = \left[\frac{x^3}{3} + 2x\right]_0^4 = \left(\frac{64}{3} + 8\right) = \frac{88}{3} = 29.\overline{3}$$

3. Metode Romberg

Metode Romberg adalah salah satu teknik integrasi numerik yang menggabungkan kelebihan dari metode trapezoid dan konsep ekstrapolasi Richardson untuk menghasilkan hasil integral dengan akurasi tinggi. Dibandingkan metode trapezoid dan Simpson, metode Romberg memiliki keunggulan dari sisi konvergensi dan efisiensi dalam mencapai ketelitian yang lebih tinggi tanpa harus memperkecil ukuran langkah secara ekstrem. Metode ini sangat cocok digunakan untuk fungsi yang halus (*smooth*) dan kontinu, di mana turunan berorde tinggi dapat diperkirakan dengan baik. Pendekatan ini secara bertahap meningkatkan akurasi dengan memanfaatkan hasil integrasi trapezoid dari beberapa tingkat subdivisi dan mengurangi galat truncation dengan teknik ekstrapolasi numerik sistematis.

Menurut Burden dan Faires (2011) dalam *Numerical Analysis*, metode Romberg dimulai dengan menghitung integral tentu menggunakan metode trapezoid pada sejumlah langkah diskret hhh yang semakin kecil. Hasil tersebut kemudian disusun dalam bentuk tabel segitiga yang dikenal sebagai tabel Romberg, yang memungkinkan kita menggabungkan informasi dari beberapa tingkat pembagian interval untuk memperbaiki hasil sebelumnya. Proses ini dilakukan dengan mengaplikasikan rumus ekstrapolasi Richardson, yaitu teknik matematis untuk memperkirakan limit dari deret pendekatan numerik terhadap nilai sebenarnya dengan mengurangi pengaruh galat berorde rendah.

Langkah pertama dalam metode Romberg adalah menghitung integral dengan metode trapezoid pada interval [a,b] dengan jumlah pembagian n=1, yang menghasilkan nilai pertama $R_{1,1}$. Kemudian jumlah subinterval digandakan (n=2,4,8,...), dan untuk setiap tingkat K, dihitung nilai $R_{k,1}$ sebagai hasil metode trapezoid dengan 2^{k-1} subinterval. Setelah itu, diperoleh nilai-nilai ekstrapolasi yang lebih tinggi dengan rumus:

$$R_{k,j} = R_{k,j-1} + \frac{R_{k,j-1} - R_{k-1,j-1}}{4^{j-1} - 1}$$

Rumus ini merupakan inti dari ekstrapolasi Richardson, di mana Rk,jR_{k,j}Rk,j merupakan hasil koreksi terhadap galat orde rendah menggunakan dua nilai sebelumnya. Dengan cara ini, setiap level baru dalam tabel Romberg memberikan pendekatan yang lebih akurat terhadap nilai integral sebenarnya.

Keunggulan metode Romberg terletak pada konvergensi cepat yang dihasilkan dari pendekatan sistematis terhadap pengurangan galat. Misalnya, metode trapezoid memiliki galat truncation orde dua ($O(h^2)$), tetapi dengan menerapkan ekstrapolasi Richardson secara berulang, metode Romberg dapat mencapai orde konvergensi yang sangat tinggi, bahkan mendekati eksponensial terhadap jumlah tingkat ekstrapolasi. Artinya, kita bisa mendapatkan hasil integral yang sangat akurat hanya dalam beberapa iterasi, tanpa harus memperkecil ukuran langkah hingga titik yang mengakibatkan akumulasi galat pembulatan.

Metode Romberg juga memiliki keterbatasan. Pertama, metode ini sangat bergantung pada kelicinan fungsi. Jika fungsi f(x) memiliki diskontinuitas, turunan tak terbatas, atau perubahan ekstrem dalam kelengkungan, maka hasil ekstrapolasi bisa menjadi tidak stabil atau menyimpang jauh dari nilai sebenarnya. Selain itu, metode ini memerlukan penyimpanan memori tambahan untuk menampung semua hasil intermediate dalam tabel Romberg, dan kompleksitas komputasinya meningkat secara signifikan dibanding metode trapezoid atau Simpson. Oleh karena itu, meskipun metode Romberg unggul dalam hal akurasi, ia tidak selalu menjadi pilihan terbaik untuk semua jenis fungsi, terutama dalam kondisi sumber daya terbatas.

Pada praktiknya, metode Romberg sangat berguna dalam aplikasi yang memerlukan hasil integrasi presisi tinggi, seperti dalam fisika teoretis, komputasi teknik, pemrosesan sinyal, atau dalam verifikasi numerik untuk membandingkan hasil dengan metode analitik. Misalnya, dalam perhitungan gaya total dalam sistem mekanika fluida, distribusi beban dalam analisis struktur, atau evaluasi energi dalam sistem partikel, metode Romberg memungkinkan penghitungan integral dengan kesalahan sangat kecil.

Implementasi metode Romberg dalam perangkat lunak komputasi seperti MATLAB, Python (melalui pustaka SciPy), dan bahasa pemrograman ilmiah lainnya sangat mudah dilakukan karena bentuknya yang rekursif dan tabel yang sistematis. Dalam Python, fungsi

scipy.integrate.romberg() menyediakan fitur otomatis untuk menghitung integral dengan metode ini tanpa harus membangun tabel secara manual.

Sebagai ilustrasi sederhana, misalkan kita ingin menghitung integral $\int_0^1 e^{-x^2} dx$, fungsi ini tidak memiliki antiturunan dalam bentuk tertutup, sehingga harus dihitung secara numerik. Dengan metode Romberg, hasilnya akan sangat dekat dengan nilai referensi $\int_0^1 e^{-x^2} dx \approx 0{,}746824$ hanya dalam beberapa iterasi, lebih cepat dibanding metode Simpson atau trapezoid standar dengan jumlah titik yang sama.

Hitung pendekatan nilai integral berikut menggunakan Metode Romberg hingga tingkat R2,2:

$$\int_0^1 \frac{1}{1+x^2}\, dx$$

Fungsi ini memiliki solusi eksak:

$$\int_0^1 \frac{1}{1+x^2}\, dx = \arctan(1) - \arctan(0) = \frac{\pi}{4} \approx 0.7854$$

Gunakan:

$$T(h) = \frac{h}{2}[f(a) + f(b)]$$

Dengan $h_1 = 1$,

$$T_1 = T(1) = \frac{1}{2}[f(0) + f(1)] = \frac{1}{2}\left[1 + \frac{1}{2}\right] = \frac{1}{2} \cdot \frac{3}{2} = 0.75$$

Dengan $h_2 = \frac{1}{2}$, ambil titik tengah $x = 0.5$:

$$T_2 = \frac{1}{4}[f(0) + 2f(0.5) + f(1)] = \frac{1}{4}\left[1 + 2 \cdot \frac{1}{1.25} + 0.5\right] = \frac{1}{4}(1 + 1.6 + 0.5) = \frac{1}{4}(3.1) = 0.775$$

Maka,

$$R_{2,2} = \frac{4^1 \cdot T_2 - T_1}{4^1 - 1} = \frac{4 \cdot 0.775 - 0.75}{3} = \frac{3.1 - 0.75}{3} = \frac{2.35}{3} \approx 0.7833$$

Dengan **Metode Romberg hingga** $R_{2,2}$, diperoleh pendekatan integral:

$$\int_0^1 \frac{1}{1+x^2}dx \approx 0.7833$$

Nilai ini sangat mendekati hasil eksak $\frac{\pi}{4} \approx 0.7854$, menunjukkan bahwa **Romberg cepat konvergen dengan presisi tinggi**.

C. Evaluasi Akurasi dan Estimasi Kesalahan

Evaluasi akurasi dan estimasi kesalahan adalah dua aspek fundamental dalam analisis numerik dan komputasi ilmiah. Ketika metode numerik digunakan untuk menghampiri solusi dari persoalan matematika seperti integral, turunan, atau solusi sistem persamaan diferensial, hasil yang diperoleh bersifat aproksimasi, bukan nilai eksak. Oleh karena itu, sangat penting untuk memahami seberapa dekat hasil tersebut terhadap nilai sebenarnya (akurasi) serta sejauh mana kesalahan (galat) dapat dikenali, diprediksi, dan dikendalikan. Dalam konteks ini, proses evaluasi akurasi dan estimasi kesalahan menjadi indikator utama yang menentukan kualitas dan reliabilitas hasil komputasi.

Menurut Burden dan Faires (2011) dalam *Numerical Analysis*, akurasi mengacu pada seberapa dekat nilai hasil komputasi numerik terhadap nilai eksak dari masalah yang diselesaikan. Dalam pengukuran matematis, akurasi dapat dijelaskan melalui galat absolut dan galat relatif. Galat absolut didefinisikan sebagai selisih langsung antara nilai eksak dan hasil aproksimasi:

$$\text{Galat Absolut} = |x_{\text{aproks}} - x_{\text{eksak}}|$$

Sedangkan **galat relatif** memberikan ukuran proporsional terhadap nilai eksak:

$$\text{Galat Relatif} = \frac{|x_{\text{aproks}} - x_{\text{eksak}}|}{|x_{\text{eksak}}|}$$

Pada praktiknya, galat relatif lebih sering digunakan karena memberikan ukuran kesalahan yang lebih kontekstual, terutama ketika nilai eksak sangat kecil atau sangat besar.

Pada Chapra dan Canale (2015), dinyatakan bahwa semua metode numerik mengandung kesalahan yang berasal dari berbagai sumber, yang secara umum dapat dikategorikan ke dalam dua kelompok

besar: galat pembulatan (*round-off error*) dan galat pemotongan (*truncation error*). Galat pembulatan timbul akibat keterbatasan representasi bilangan real dalam komputer, sedangkan galat pemotongan muncul dari penyederhanaan atau pemotongan operasi matematika seperti deret Taylor atau pendekatan diskrit dalam diferensiasi dan integrasi.

1. Galat Pembulatan

Galat pembulatan (*round-off error*) adalah salah satu jenis kesalahan paling mendasar dalam komputasi numerik yang berasal dari keterbatasan sistem komputer dalam merepresentasikan bilangan real. Dalam sistem digital, komputer menyimpan angka dalam bentuk biner dengan jumlah digit terbatas. Akibatnya, banyak bilangan riil yang tidak dapat direpresentasikan secara persis dalam sistem biner tersebut, sehingga komputer harus melakukan pembulatan ke nilai terdekat yang dapat direpresentasikan. Proses pembulatan inilah yang menghasilkan galat pembulatan, yang dalam banyak kasus bersifat sangat kecil, tetapi bisa menjadi signifikan ketika akumulatif dalam perhitungan yang kompleks atau berulang.

Menurut Burden dan Faires (2011), galat pembulatan merupakan konsekuensi langsung dari penggunaan bilangan floating-point dalam sistem komputasi. Standar umum seperti IEEE 754 mendefinisikan bagaimana bilangan disimpan dan dioperasikan dalam memori komputer. Dalam standar ini, sebuah bilangan floating-point disusun dari tiga bagian: bit tanda (sign), eksponen, dan mantissa (atau *significand*). Karena panjang mantissa terbatas (misalnya, 23 bit untuk single precision dan 52 bit untuk double precision), hanya sejumlah terbatas angka desimal yang dapat diwakili secara tepat. Misalnya, angka 1/3 dalam sistem desimal adalah 0.333..., sebuah desimal tak hingga. Dalam sistem biner, representasi ini lebih terbatas lagi dan pasti akan dipotong atau dibulatkan pada digit tertentu, menghasilkan nilai yang sedikit berbeda dari nilai eksaknya.

Pada praktiknya, galat pembulatan bisa muncul dalam berbagai bentuk operasi numerik, termasuk penjumlahan, perkalian, pembagian, dan terutama dalam operasi yang melibatkan selisih dua bilangan yang hampir sama (dikenal sebagai *cancellation*). Misalnya, dalam operasi $x-y$ di mana x dan y bernilai sangat dekat, hasil selisihnya menjadi sangat kecil dan dapat kehilangan digit signifikan, sehingga

meningkatkan proporsi galat pembulatan terhadap nilai hasil. Hal ini sangat berbahaya dalam komputasi presisi tinggi karena galat kecil pada digit rendah bisa menjadi dominan.

Pada iterasi numerik atau algoritma rekursif, galat pembulatan bisa terpropagasi dan diperbesar. Sebagai contoh, dalam metode numerik seperti metode Euler atau Runge-Kutta untuk menyelesaikan persamaan diferensial, pembulatan hasil setiap langkah akan memengaruhi langkah berikutnya. Jika tidak dikendalikan, hal ini dapat menyebabkan akumulasi kesalahan yang signifikan dan mengarahkan solusi pada hasil yang sangat menyimpang dari kenyataan. Oleh karena itu, analisis stabilitas numerik menjadi penting dalam menilai sejauh mana suatu metode tahan terhadap galat pembulatan.

Strategi untuk mengurangi dampak galat pembulatan mencakup penggunaan presisi lebih tinggi (misalnya, menggunakan double daripada *single precision*), pembulatan yang stabil secara numerik, dan penataan ulang algoritma untuk menghindari pengurangan angka yang hampir sama atau pembagian terhadap angka sangat kecil. Dalam pengembangan perangkat lunak ilmiah dan rekayasa, teknik ini menjadi bagian penting dari proses validasi hasil numerik.

2. Galat Pemotongan

Galat pemotongan (*truncation error*) adalah jenis kesalahan numerik yang muncul ketika suatu metode numerik menggunakan pendekatan yang menyederhanakan operasi matematis eksak dengan memotong atau mengabaikan bagian dari ekspresi matematis tersebut. Tidak seperti galat pembulatan yang berasal dari keterbatasan representasi bilangan dalam komputer, galat pemotongan terjadi karena metode numerik secara sadar memilih untuk hanya mempertahankan sebagian komponen dari suatu operasi, seperti deret tak hingga, turunan, atau integral. Galat ini bersifat sistematis dan dapat dihitung atau dikendalikan melalui pemilihan metode serta pengaturan parameter numerik seperti ukuran langkah diskrit.

Menurut Chapra dan Canale (2015) dalam *Numerical Methods for Engineers,* galat pemotongan paling umum terjadi dalam pendekatan numerik terhadap turunan dan integral. Misalnya, dalam metode selisih hingga untuk menghitung turunan pertama dari suatu fungsi $f(x)$, digunakan formula:

$$f'(x) \approx \frac{f(x+h) - f(x)}{h}$$

yang berasal dari pemotongan deret Taylor setelah suku pertama. Dalam ekspansi Taylor, turunan eksak dituliskan sebagai:

$$f(x+h) = f(x) + hf'(x) + \frac{h^2}{2}f''(x) + \cdots$$

Dengan mengabaikan suku $\frac{h^2}{2}$f''(x)+…., maka terjadi galat pemotongan. Semakin besar nilai hhh, maka semakin besar juga galat pemotongan yang ditimbulkan karena kontribusi suku-suku yang diabaikan menjadi signifikan.

Galat pemotongan juga terjadi dalam metode integrasi numerik seperti metode trapezoid atau Simpson. Misalnya, dalam metode trapezoid komposit, integral dihampiri oleh jumlah luas trapesium antara titik-titik fungsi. Dalam pendekatan ini, bentuk kurva sebenarnya diganti dengan garis lurus, sehingga bagian melengkung dari fungsi tidak diperhitungkan secara tepat. Galat yang timbul dapat diekspresikan secara matematis sebagai:

$$E_T = -\frac{(b-a)^3}{12n^2} f''(\xi)$$

untuk suatu $\xi \in [a,b]$, menunjukkan bahwa galat tergantung pada turunan kedua fungsi dan jumlah subinterval nnn. Ini berarti bahwa galat pemotongan dapat dikurangi dengan memperkecil h (yakni memperbesar jumlah subinterval), atau dengan menggunakan metode numerik dengan orde lebih tinggi seperti metode Simpson yang memperhitungkan kelengkungan fungsi.

Salah satu keunggulan dari galat pemotongan dibanding galat pembulatan adalah sifatnya yang dapat diprediksi dan dikendalikan. Jika suatu metode memiliki orde galat tertentu, maka pengguna dapat secara sistematis memperkirakan berapa banyak kesalahan yang akan terjadi dan menyesuaikan parameter komputasi agar kesalahan tetap dalam batas toleransi. Misalnya, metode dengan galat orde dua ($O(h^2)$) akan memiliki galat yang berkurang seperempat jika hhh dibagi dua.

Pemahaman ini memungkinkan desain algoritma yang adaptif terhadap kebutuhan presisi.

Ada batas bawah di mana pengurangan hhh tidak lagi efektif karena akan memicu galat pembulatan, sehingga terdapat trade-off antara mengurangi galat pemotongan dan mencegah galat pembulatan. Dalam pengembangan perangkat lunak numerik yang efisien, penting untuk menyeimbangkan dua jenis galat ini agar diperoleh hasil komputasi yang optimal.

3. Evaluasi Akurasi Secara Praktis

Evaluasi akurasi secara praktis dalam komputasi numerik adalah proses menilai seberapa dekat hasil komputasi mendekati nilai eksak, meskipun dalam banyak kasus nilai eksak tersebut tidak diketahui secara pasti. Oleh karena itu, evaluasi akurasi tidak hanya bergantung pada perhitungan galat absolut atau relatif, melainkan juga pada strategi-strategi praktis yang dapat digunakan untuk memverifikasi dan memvalidasi hasil aproksimasi numerik. Evaluasi ini menjadi sangat penting dalam konteks aplikasi nyata, seperti simulasi fisika, optimasi teknik, atau perhitungan statistik, di mana hasil komputasi sering dijadikan dasar pengambilan keputusan atau desain sistem.

Salah satu pendekatan paling umum dalam evaluasi akurasi adalah analisis konvergensi. Dalam metode ini, hasil komputasi numerik diuji dengan melakukan perhitungan berulang menggunakan ukuran langkah atau parameter diskret yang semakin kecil. Jika metode numerik yang digunakan benar dan stabil, maka hasil perhitungan akan menunjukkan pola konvergen menuju nilai tetap tertentu. Misalnya, dalam integrasi numerik menggunakan metode trapezoid atau Simpson, jika panjang langkah h diperkecil, hasil integral yang diperoleh seharusnya semakin mendekati nilai sejati. Pola konvergensi ini dapat diukur dan divisualisasikan dengan memplot hasil terhadap ukuran h atau jumlah subinterval. Dengan demikian, meskipun nilai eksak tidak diketahui, kita dapat memperkirakan bahwa hasil sudah berada dalam kisaran yang dapat diterima secara akurat.

Evaluasi akurasi juga dilakukan melalui perbandingan antar metode numerik. Dalam banyak kasus, dua atau lebih metode dengan orde akurasi berbeda digunakan untuk menyelesaikan masalah yang sama. Selisih hasil antara metode berakurasi lebih tinggi (misalnya Simpson atau Romberg) dengan metode lebih sederhana (misalnya

trapezoid) dapat memberikan gambaran mengenai besar kesalahan aproksimasi. Teknik ini dikenal sebagai pendekatan estimasi galat melalui redundansi metode, dan sering digunakan dalam perangkat lunak komputasi teknik untuk memberikan informasi kepercayaan terhadap hasil.

Cara lainnya adalah menggunakan pendekatan nilai referensi atau solusi benchmark, terutama dalam kasus di mana fungsi uji tertentu telah diketahui nilai eksaknya. Ini sering digunakan dalam pengujian algoritma numerik, di mana fungsi dengan solusi analitik digunakan untuk membandingkan hasil aproksimasi. Jika hasil numerik mendekati solusi referensi dengan galat relatif kecil, maka metode dianggap valid dan akurat untuk konteks yang serupa.

Pada iterasi numerik seperti metode Newton-Raphson atau metode Jacobi, galat antar iterasi sering digunakan sebagai indikator akurasi. Jika perubahan nilai hasil antara dua iterasi berturut-turut sangat kecil (misalnya kurang dari 10−6), maka hasil tersebut diasumsikan telah konvergen. Meskipun bukan galat sejati, estimasi ini secara praktis cukup efektif dalam menentukan titik henti perhitungan.

D. Aplikasi pada Persoalan Teknik

Pada bidang teknik, perhitungan analitik yang presisi sering kali tidak memungkinkan karena kompleksitas sistem yang dianalisis. Oleh sebab itu, metode numerik menjadi pendekatan utama dalam menyelesaikan berbagai persoalan teknik yang melibatkan formulasi matematika rumit, fungsi tak diketahui, dan sistem tak linear. Metode numerik memungkinkan solusi pendekatan terhadap masalah teknik dengan efisiensi tinggi, baik dari sisi waktu maupun sumber daya komputasi. Persoalan teknik yang melibatkan mekanika struktur, perpindahan panas, dinamika fluida, kelistrikan, kontrol sistem, dan simulasi material kini hampir seluruhnya mengandalkan pendekatan numerik.

Menurut Chapra dan Canale (2015) dalam *Numerical Methods for Engineers,* metode numerik digunakan dalam teknik karena sebagian besar masalah teknik nyata mengarah pada persamaan diferensial (biasa atau parsial), sistem persamaan linier besar, atau fungsi-fungsi yang tidak dapat diintegrasikan secara analitik. Dalam konteks ini, metode seperti metode selisih hingga (*finite difference method*/FDM), metode elemen hingga (*finite element method*/FEM), metode volume hingga

(*finite volume method*/FVM), dan metode Runge-Kutta menjadi tulang punggung penyelesaian masalah-masalah teknik secara komputasional.

1. Mekanika Struktur

Mekanika struktur adalah cabang penting dalam bidang teknik sipil, mesin, dan arsitektur yang mempelajari perilaku benda padat terutama struktur teknik yang mengalami beban, tekanan, dan gaya lainnya. Fokus utamanya adalah untuk menganalisis dan merancang struktur seperti balok, kolom, rangka baja, jembatan, gedung, hingga pesawat terbang, agar mampu menahan beban yang bekerja tanpa mengalami kerusakan atau kegagalan. Konsep-konsep dasar dalam mekanika struktur meliputi tegangan (*stress*), regangan (*strain*), lendutan (*deflection*), momen lentur, gaya geser, dan stabilitas struktur. Prinsip-prinsip ini diterapkan untuk memastikan bahwa struktur dirancang tidak hanya kuat dan stabil, tetapi juga efisien dari segi material dan biaya.

Menurut Hibbeler (2012) dalam *Mechanics of Materials*, mekanika struktur bekerja berdasarkan hukum-hukum dasar fisika, khususnya hukum Newton dan prinsip keseimbangan gaya, serta prinsip deformasi elastis dari bahan. Salah satu konsep inti dalam analisis struktur adalah hukum Hooke, yang menjelaskan hubungan linear antara tegangan dan regangan dalam batas elastis suatu material. Dalam konteks teknik, hal ini memungkinkan insinyur untuk menghitung respons struktur terhadap gaya yang bekerja, seperti lenturan balok akibat beban merata atau gaya konsentris pada kolom.

Pada aplikasi praktisnya, mekanika struktur tidak hanya berfungsi sebagai alat analisis, tetapi juga sebagai dasar untuk pengambilan keputusan desain. Misalnya, dalam merancang jembatan baja, insinyur harus menentukan ukuran, bentuk, dan jenis material yang digunakan berdasarkan perhitungan tegangan maksimum, batas leleh material, serta faktor keamanan. Selain itu, struktur harus memenuhi kriteria batas (*limit states*), baik batas kekuatan (*ultimate limit state*) maupun batas layan (*serviceability limit state*), agar tetap aman dan nyaman digunakan sepanjang umur bangunan.

Seiring dengan kompleksitas bentuk struktur dan beban yang semakin variatif, metode analisis manual menjadi terbatas. Oleh karena itu, analisis numerik, khususnya metode elemen hingga (*finite element method*/FEM), menjadi alat bantu utama dalam mekanika struktur modern. Metode ini membagi struktur kompleks menjadi elemen-elemen

kecil (seperti segitiga atau persegi), dan menghitung distribusi gaya, tegangan, dan deformasi pada setiap elemen untuk kemudian digabungkan menjadi analisis keseluruhan. Perangkat lunak berbasis FEM seperti SAP2000, ANSYS, atau ABAQUS kini menjadi standar dalam perencanaan struktur besar seperti gedung pencakar langit atau jembatan gantung.

Mekanika struktur juga mencakup aspek dinamik struktur, di mana struktur harus mampu menghadapi beban yang berubah terhadap waktu, seperti gempa bumi, angin kencang, atau lalu lintas kendaraan berat. Untuk kondisi seperti ini, struktur perlu dianalisis berdasarkan respons dinamis, dan sering kali melibatkan simulasi berbasis metode numerik yang memperhitungkan gaya inersia, redaman, dan frekuensi alami sistem.

2. Perpindahan Panas dan Termodinamika

Perpindahan panas dan termodinamika merupakan dua cabang ilmu penting dalam teknik mesin, teknik kimia, dan rekayasa energi yang saling berkaitan erat dalam memahami dan mengendalikan fenomena energi dalam sistem teknik. Termodinamika berfokus pada studi mengenai hubungan antara panas, kerja, dan energi dalam suatu sistem, serta kondisi kesetimbangan termal dan proses perubahan energi yang terjadi. Di sisi lain, perpindahan panas (*heat transfer*) mempelajari bagaimana energi dalam bentuk panas berpindah dari satu tempat ke tempat lain melalui tiga mekanisme utama: konduksi, konveksi, dan radiasi. Kedua cabang ini menjadi landasan dalam merancang mesin, sistem pendingin, turbin, boiler, reaktor, dan banyak sistem teknik lainnya yang melibatkan konversi dan transportasi energi.

Menurut Çengel dan Boles (2015) dalam *Thermodynamics: An Engineering Approach,* termodinamika menjelaskan perubahan energi melalui hukum-hukum dasar: Hukum Pertama Termodinamika yang menyatakan kekekalan energi, dan Hukum Kedua Termodinamika yang memperkenalkan konsep entropi dan arah alami dari proses termal. Misalnya, dalam sistem mesin kalor seperti motor bakar atau turbin gas, hukum pertama menjelaskan konversi panas menjadi kerja mekanis, sedangkan hukum kedua membatasi efisiensi konversi tersebut, karena selalu ada sebagian energi yang hilang sebagai panas yang tidak dapat digunakan. Hukum-hukum ini digunakan untuk menganalisis siklus termal seperti siklus Rankine, siklus Otto, dan siklus Brayton, yang

menjadi dasar dalam perancangan pembangkit listrik dan mesin kendaraan.

Ilmu perpindahan panas menjadi penting dalam menentukan bagaimana dan seberapa cepat panas berpindah dari satu bagian sistem ke bagian lainnya. Dalam konduksi, panas mengalir dalam benda padat dari suhu tinggi ke suhu rendah melalui getaran atom dan konduksi elektron, dijelaskan dengan hukum Fourier. Dalam konveksi, panas berpindah antara permukaan padat dan fluida yang mengalir, dan dianalisis menggunakan bilangan Nusselt, bilangan Reynolds, dan hukum Newton pendinginan. Sedangkan radiasi panas melibatkan energi elektromagnetik yang dipancarkan oleh permukaan benda, dijelaskan dengan hukum Stefan-Boltzmann dan konsep emisivitas.

Pada rekayasa, analisis perpindahan panas sering digunakan untuk merancang sistem pendingin (seperti radiator, *heat exchanger,* dan sistem HVAC), serta isolasi termal pada dinding bangunan atau pipa. Misalnya, pada sistem penukar kalor (*heat exchanger*), insinyur harus memperhitungkan laju perpindahan panas antara dua fluida tanpa mencampurkannya secara langsung, guna memastikan efisiensi energi dan stabilitas operasi. Untuk sistem elektronik, analisis termal diperlukan untuk memastikan bahwa suhu komponen seperti prosesor tidak melebihi batas operasionalnya.

Perhitungan perpindahan panas dan termodinamika sering kali melibatkan persamaan diferensial parsial yang kompleks, sehingga metode numerik seperti metode elemen hingga (FEM) atau metode beda hingga (FDM) digunakan untuk menghitung distribusi suhu dalam geometri yang rumit. Perangkat lunak seperti ANSYS, COMSOL, atau MATLAB digunakan untuk simulasi termal secara menyeluruh, mulai dari analisis stasioner hingga perpindahan panas transien.

Dengan memahami prinsip-prinsip perpindahan panas dan termodinamika, para insinyur mampu mengendalikan aliran energi dan suhu dalam sistem teknik secara efisien dan berkelanjutan. Ilmu ini sangat vital dalam pengembangan teknologi energi terbarukan, kendaraan hemat energi, bangunan ramah lingkungan, dan sistem pendinginan berteknologi tinggi. Oleh karena itu, perpindahan panas dan termodinamika terus menjadi pilar utama dalam inovasi teknologi energi dan rekayasa masa depan.

3. Dinamika Fluida Komputasi (CFD)

Dinamika Fluida Komputasi (*Computational Fluid Dynamics*/CFD) adalah cabang teknik yang menggunakan metode numerik dan algoritma komputasi untuk menganalisis dan memecahkan persoalan yang melibatkan aliran fluida dan transfer energi. Dalam konteks teknik, CFD menjadi alat penting dalam memahami perilaku fluida (gas dan cairan), memprediksi distribusi tekanan, kecepatan, suhu, dan berbagai parameter penting lainnya dalam sistem rekayasa yang kompleks. Dengan kemampuan ini, CFD telah menjadi teknologi utama dalam desain dan optimasi produk pada berbagai sektor seperti penerbangan, otomotif, energi, lingkungan, hingga biomedis.

Menurut Versteeg dan Malalasekera (2007) dalam *An Introduction to Computational Fluid Dynamics,* CFD bekerja dengan cara mendiskretkan dan menyelesaikan persamaan Navier-Stokes, yaitu persamaan diferensial parsial non-linier yang mendeskripsikan konservasi massa (kontinuitas), momentum (hukum Newton), dan energi dalam fluida. Karena sulit atau bahkan tidak mungkin menyelesaikan persamaan tersebut secara analitik untuk kasus nyata yang kompleks, maka metode numerik seperti *finite volume method* (FVM), *finite element method* (FEM), dan *finite difference method* (FDM) digunakan untuk menyelesaikannya dalam bentuk diskrit di atas grid atau mesh yang menggambarkan domain aliran.

Pada aplikasi teknik, CFD memberikan keunggulan besar karena mampu menggantikan eksperimen fisik yang mahal dan memakan waktu. Misalnya, dalam industri otomotif, CFD digunakan untuk menganalisis aerodinamika mobil guna mengurangi hambatan udara dan meningkatkan efisiensi bahan bakar. Dalam teknik penerbangan, CFD membantu mendesain bentuk sayap dan badan pesawat agar memiliki lift optimal dan drag minimal. CFD juga digunakan dalam perencanaan sistem HVAC (*Heating, Ventilation, and Air Conditioning*) di gedung untuk memastikan aliran udara dan distribusi suhu sesuai standar kenyamanan termal.

CFD memungkinkan simulasi berbagai jenis aliran, mulai dari aliran laminar hingga turbulen, aliran termal konvektif, aliran multifase (seperti campuran air dan udara), hingga reaksi kimia dalam aliran fluida. Dengan kemajuan teknologi komputasi, perangkat lunak CFD modern seperti ANSYS Fluent, OpenFOAM, COMSOL Multiphysics, dan STAR-CCM+ menyediakan antarmuka dan solver canggih yang mampu

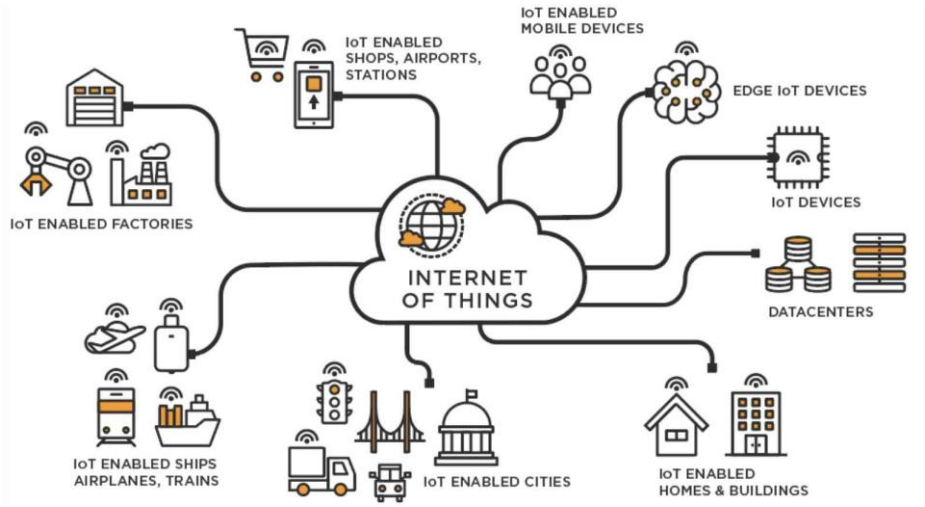
menangani geometri kompleks, berbagai kondisi batas, serta interaksi fluida dengan struktur padat.

CFD bukan hanya soal menjalankan simulasi. Kualitas hasil sangat bergantung pada pemahaman fisika aliran, pemilihan model turbelensi yang tepat (misalnya k-ε atau *Large Eddy Simulation*), pemilihan skema numerik yang stabil, serta resolusi mesh yang cukup untuk menangkap fenomena aliran penting. Selain itu, proses validasi dan verifikasi harus dilakukan untuk memastikan bahwa hasil simulasi mendekati realitas fisik dan cocok dengan data eksperimen atau perhitungan teoritis.

4. Teknik Elektro dan Elektronika

Teknik elektro dan elektronika merupakan cabang ilmu teknik yang berfokus pada studi, perancangan, dan penerapan sistem yang melibatkan listrik, elektromagnetisme, serta perangkat elektronik. Ruang lingkup teknik elektro mencakup sistem tenaga listrik, kontrol, komunikasi, dan komputer, sedangkan teknik elektronika lebih menekankan pada perancangan dan pemrosesan sinyal dalam perangkat-perangkat mikro seperti transistor, rangkaian terpadu (IC), sensor, dan mikrokontroler. Kedua bidang ini menjadi fondasi utama dari berbagai inovasi teknologi modern, mulai dari pembangkit listrik dan jaringan distribusi hingga ponsel pintar dan perangkat *Internet of Things* (IoT).

Gambar 4. *Internet of Things*



Sumber:*Binar*

Menurut Hambley (2014) dalam *Electrical Engineering: Principles and Applications*, teknik elektro mempelajari bagaimana energi listrik dihasilkan, ditransmisikan, dan digunakan secara efisien. Di sektor pembangkitan tenaga listrik, para insinyur elektro merancang sistem pembangkit seperti PLTA, PLTU, PLTS, dan PLTN, serta sistem transmisi tegangan tinggi. Dalam proses ini, metode numerik digunakan untuk menganalisis sistem jaringan listrik secara linier dan non-linier, menentukan kestabilan tegangan, arus gangguan, serta distribusi beban. Simulasi berbasis perangkat lunak seperti ETAP, MATLAB Simulink, dan PowerWorld menjadi alat utama dalam perencanaan sistem tenaga yang andal dan berkelanjutan.

Teknik elektronika berkaitan erat dengan pengembangan sirkuit mikroelektronik dan sistem digital. Para insinyur elektronika merancang sirkuit menggunakan komponen seperti resistor, kapasitor, dioda, transistor, dan mikrokontroler untuk menciptakan perangkat seperti sensor suhu, penguat sinyal, osilator, serta sistem tertanam (*embedded systems*). Elektronika juga menjadi tulang punggung teknologi komunikasi dan kontrol, misalnya pada pemancar radio, radar, perangkat nirkabel, dan sistem kendali otomatis di industri. Dalam pengembangan perangkat seperti ponsel, laptop, dan peralatan medis, prinsip-prinsip elektronika digunakan untuk memastikan efisiensi energi, keandalan sinyal, dan miniaturisasi perangkat.

Salah satu aspek penting dalam teknik elektro dan elektronika modern adalah pengolahan sinyal digital (DSP), di mana sinyal analog diubah menjadi bentuk digital agar dapat diolah, disimpan, dan ditransmisikan secara efisien. Teknik ini digunakan dalam berbagai aplikasi seperti pemrosesan audio, pengenalan suara, pengolahan gambar digital, serta sistem komunikasi nirkabel. Dalam hal ini, metode numerik seperti transformasi *Fourier diskrit* (DFT), filter digital (FIR dan IIR), serta algoritma kompresi menjadi elemen penting dalam mendukung kinerja sistem.

Perkembangan teknologi otomasi dan kontrol juga sangat bergantung pada teknik elektro dan elektronika. Sistem kendali seperti PID (*Proportional-Integral-Derivative*), kontrol logika fuzzy, hingga kontrol adaptif digunakan untuk mengatur kecepatan motor, posisi robot, suhu sistem, dan proses industri lainnya. Dengan dukungan sensor dan aktuator, sistem ini mampu bekerja secara otomatis dan presisi tinggi. Dalam dunia industri 4.0, integrasi antara teknik elektro, pemrograman,

dan komunikasi data membentuk sistem cerdas berbasis IoT dan AI yang mampu mengoptimalkan produktivitas dan efisiensi.

5. Sistem Otomatisasi dan Kontrol

Sistem otomatisasi dan kontrol merupakan cabang penting dalam teknik elektro, mekatronika, dan teknik industri yang berfokus pada pengaturan perilaku sistem dinamis secara otomatis melalui penggunaan perangkat keras (seperti sensor, aktuator, dan kontroler) dan perangkat lunak (seperti algoritma kontrol dan sistem tertanam). Tujuan utama dari sistem ini adalah menciptakan operasi yang efisien, presisi tinggi, stabil, dan minim campur tangan manusia. Sistem kontrol banyak diterapkan dalam berbagai bidang, mulai dari proses manufaktur, otomotif, robotika, energi, transportasi, hingga peralatan rumah tangga pintar.

Menurut Ogata (2010) dalam *Modern Control Engineering*, sistem kontrol bekerja berdasarkan prinsip pengumpanan (*feedback*) atau umpan terbuka (*open-loop*). Dalam kontrol umpan balik, sensor digunakan untuk mendeteksi keluaran sistem, lalu informasi ini dibandingkan dengan nilai referensi (setpoint). Selisih antara keduanya (disebut sebagai error) akan diproses oleh pengontrol (seperti pengontrol PID) untuk menghasilkan sinyal yang mengatur aktuator, sehingga sistem dapat menyesuaikan dirinya agar tetap berada dalam kondisi yang diinginkan. Misalnya, dalam sistem kendali suhu ruangan, sensor suhu mengukur kondisi aktual dan pengontrol mengatur pemanas atau pendingin untuk mencapai suhu target.

Pengontrol PID (*Proportional-Integral-Derivative*) adalah jenis kontroler yang paling umum dan banyak digunakan karena kesederhanaannya serta efektivitasnya dalam berbagai jenis sistem. Komponen proporsional (P) memberikan respons terhadap error saat ini, integral (I) mengatasi error jangka panjang (akumulasi error), dan turunan (D) merespons perubahan cepat dari error. Dengan penyesuaian parameter yang tepat, pengontrol PID mampu mengendalikan sistem dengan respons yang cepat dan stabil tanpa overshoot atau osilasi berlebih.

Pada sistem industri modern, otomatisasi dikembangkan melalui penggunaan *Programmable Logic Controller* (PLC), yang merupakan komputer industri tahan lingkungan yang diprogram untuk mengatur proses produksi secara berurutan atau paralel. PLC membaca sinyal dari sensor, memproses logika kendali, dan mengaktifkan output seperti

motor, katup, atau lampu indikator. PLC menjadi komponen inti dalam sistem otomasi pabrik, termasuk pada lini perakitan mobil, pengemasan makanan, dan pengolahan air bersih.

Integrasi sistem kontrol dengan teknologi informasi telah melahirkan konsep kontrol berbasis komputer dan jaringan. Dalam pendekatan ini, sistem kendali terhubung secara digital dan dapat diakses serta dikendalikan secara jarak jauh melalui antarmuka manusia-mesin (HMI) atau *Supervisory Control and Data Acquisition* (SCADA). Ini memungkinkan perusahaan memantau seluruh proses produksi secara real-time, meningkatkan efisiensi operasional, deteksi kesalahan lebih awal, dan pengambilan keputusan yang berbasis data.

Di sektor transportasi, sistem kontrol digunakan dalam cruise control mobil, sistem navigasi pesawat terbang, hingga kendali otomatis kereta cepat. Sementara dalam dunia robotika, sistem kontrol memastikan gerakan lengan robot presisi sesuai dengan jalur atau posisi targetnya. Dalam energi, sistem kontrol berperan dalam manajemen grid listrik, pengaturan turbin pembangkit, dan sistem energi terbarukan yang dinamis seperti panel surya dan turbin angin.

Dengan perkembangan teknologi sensor, kecerdasan buatan (AI), dan komunikasi data, sistem otomatisasi dan kontrol kini berkembang menuju arah *cyber-physical systems* dan *internet of things* (IoT). Sistem-sistem ini mampu beradaptasi secara cerdas terhadap perubahan lingkungan, melakukan prediksi kegagalan, serta belajar dari data historis untuk mengoptimalkan performa secara berkelanjutan. Oleh karena itu, pemahaman terhadap sistem otomatisasi dan kontrol sangat penting bagi para insinyur dan teknolog modern yang ingin membangun sistem yang efisien, adaptif, dan siap menghadapi tantangan industri masa depan.

6. Rekayasa Material dan Struktur Mikro

Rekayasa material dan struktur mikro adalah bidang interdisipliner dalam teknik dan ilmu material yang mempelajari hubungan antara struktur internal material pada skala mikro dan nano terhadap sifat mekanik, termal, listrik, maupun kimianya. Tujuan utama dari bidang ini adalah merekayasa bahan dengan sifat yang diinginkan melalui kontrol atas komposisi, morfologi, dan struktur mikroskopik. Rekayasa material sangat penting dalam mendukung inovasi teknologi, mulai dari pengembangan bahan ringan untuk pesawat terbang, logam

tahan panas untuk turbin gas, hingga material superkonduktor, biomaterial, dan nanokomposit untuk perangkat elektronik canggih.

Menurut Callister dan Rethwisch (2020) dalam *Materials Science and Engineering: An Introduction,* sifat makroskopik suatu material sangat dipengaruhi oleh struktur mikronya, seperti ukuran butir, orientasi kristal, cacat kristal, serta distribusi fasa. Misalnya, logam dengan ukuran butir yang lebih kecil cenderung memiliki kekuatan tarik yang lebih tinggi, sebagaimana dijelaskan dalam prinsip Hall-Petch, yang menyatakan bahwa kekuatan logam meningkat seiring dengan berkurangnya ukuran butir. Oleh karena itu, proses-proses seperti pengerjaan dingin, pemanasan ulang (*annealing*), atau rekayasa solidifikasi dimanfaatkan untuk mengubah struktur mikro demi mendapatkan karakteristik mekanik yang diinginkan.

Rekayasa struktur mikro tidak hanya terbatas pada logam, tetapi juga mencakup keramik, polimer, dan komposit. Dalam pengembangan komposit, misalnya, struktur mikro dirancang sedemikian rupa agar serat penguat (seperti serat karbon atau serat kaca) terdistribusi merata dalam matriks polimer atau logam. Hal ini menghasilkan material dengan kombinasi kekuatan tinggi, ringan, dan ketahanan terhadap keausan atau korosi. Sifat-sifat seperti ini sangat dibutuhkan dalam industri otomotif, pesawat terbang, serta konstruksi infrastruktur yang mengutamakan efisiensi dan daya tahan.

Perkembangan teknologi juga memungkinkan analisis struktur mikro hingga ke tingkat nano dengan menggunakan teknik karakterisasi canggih seperti mikroskop elektron transmisi (TEM), mikroskop elektron pemindaian (SEM), dan difraksi sinar-X (XRD). Teknik ini memungkinkan para insinyur dan ilmuwan material memahami distribusi fasa, cacat kristal, atau interaksi atom dalam material. Informasi ini menjadi dasar dalam pemodelan material berbasis komputer, termasuk simulasi molekuler dan metode elemen hingga (FEM) untuk memprediksi perilaku material dalam kondisi ekstrem, seperti tekanan tinggi atau suhu tinggi. Selain itu, struktur mikro sangat berperan dalam rekayasa material fungsional, seperti bahan magnetik, piezoelektrik, termolistrik, dan superkonduktor. Dalam bidang biomaterial, struktur mikro digunakan untuk merekayasa implan tulang dan jaringan buatan agar memiliki porositas dan kekasaran permukaan yang sesuai dengan proses regenerasi biologis.

BAB VII
PENYELESAIAN PERSAMAAN NONLINEAR

Persamaan nonlinear merupakan salah satu fondasi penting dalam dunia sains dan teknik, yang sering kali muncul dalam berbagai permasalahan nyata seperti mekanika, dinamika fluida, ekonomi, serta sistem kendali. Tidak seperti persamaan linear yang memiliki sifat sederhana dan solusi langsung, persamaan nonlinear menghadirkan kompleksitas tinggi dan memerlukan pendekatan khusus untuk menemukan solusinya. Dalam dunia komputasi modern, penyelesaian persamaan nonlinear secara numerik menjadi sangat relevan karena sering kali tidak tersedia solusi analitik yang eksak. Oleh karena itu, metode numerik seperti *bisection*, regula falsi, Newton-Raphson, dan *secant method* dikembangkan untuk memberikan solusi pendekatan yang efisien dan stabil. Melalui bab ini, pembaca akan diperkenalkan pada prinsip dasar penyelesaian persamaan nonlinear, karakteristik konvergensi metode-metode yang digunakan, serta kelebihan dan keterbatasan masing-masing pendekatan.

A. Metode Bagi Dua dan Regulafalsi

Di dunia komputasi numerik, penyelesaian persamaan nonlinear menjadi salah satu topik penting, khususnya ketika solusi analitik tidak dapat ditemukan secara langsung. Dua metode paling dasar namun efektif yang digunakan untuk menyelesaikan persamaan nonlinear

adalah Metode Bagi Dua (*Bisection Method*) dan Metode Regula Falsi (*False Position Method*). Kedua metode ini tergolong dalam kelompok metode bracketing, yaitu teknik yang memerlukan dua nilai awal yang mengurung akar fungsi.

1. Metode Bagi Dua

Metode Bagi Dua atau *Bisection Method* merupakan salah satu teknik paling dasar dan penting dalam penyelesaian persamaan nonlinear secara numerik. Metode ini tergolong dalam kategori *bracketing methods*, yakni pendekatan yang bekerja dengan menyempitkan interval yang mengandung akar dari suatu fungsi secara bertahap. Prinsip dasarnya sangat sederhana namun kuat, yakni jika suatu fungsi kontinu f(x) memiliki tanda yang berlawanan pada dua titik, misalnya $f(a)<0$ dan $f(b)>0$, maka menurut Teorema Nilai Antara (*Intermediate Value Theorem*) pasti terdapat setidaknya satu akar $c \in (a,b)$ yang memenuhi $f(c)=0$. Inilah dasar teoritis dari metode bagi dua.

Langkah pertama dalam metode ini adalah menentukan interval awal [a,b] di mana nilai fungsi pada kedua ujung memiliki tanda yang berlawanan. Setelah itu, titik tengah dari interval dihitung, yaitu $c = \frac{a+b}{2}$, dan nilai fungsi di titik tersebut, f(c) dievaluasi. Jika $f(c)=0$, maka c adalah solusi akar yang dicari. Namun dalam praktiknya, sangat jarang nilai fungsi di titik tengah benar-benar nol. Oleh karena itu, proses dilanjutkan dengan menentukan subinterval baru yang masih mengandung akar, yaitu antara [a,c] atau [c,b], tergantung pada tanda fungsi di titik-titik tersebut. Proses ini diulang terus-menerus dengan cara yang sama hingga panjang interval $|b-a|$ menjadi sangat kecil atau nilai fungsi $|f(c)|$ mendekati nol, sesuai dengan tingkat toleransi kesalahan yang telah ditentukan.

Kekuatan utama metode bagi dua terletak pada stabilitas dan jaminan konvergensi. Selama syarat dasar $f(a)\times f(b)<0$ dipenuhi dan fungsi bersifat kontinu pada interval tersebut, maka metode ini pasti akan menemukan akar atau pendekatannya. Oleh karena itu, metode ini dianggap sangat andal, terutama dalam kondisi di mana fungsi memiliki bentuk yang kompleks atau tidak mudah diturunkan secara analitik. Pendekatan ini juga tidak bergantung pada kemiringan fungsi atau perubahan bentuk grafiknya, yang membuatnya sangat berguna untuk

fungsi-fungsi yang tidak diketahui bentuk pastinya atau yang memiliki perilaku tidak terduga di antara titik a dan b.

Keunggulan dalam kestabilan dan kesederhanaan ini juga disertai dengan kelemahan, terutama dalam hal kecepatan konvergensi. Metode bagi dua hanya menunjukkan konvergensi linier, artinya error berkurang secara bertahap dan cukup lambat dari iterasi ke iterasi. Dalam praktiknya, dibutuhkan banyak iterasi untuk mencapai ketelitian yang tinggi, terutama jika akar berada sangat dekat dengan salah satu ujung interval. Hal ini menjadikan metode ini kurang efisien dibandingkan metode yang menggunakan informasi tambahan, seperti Newton-Raphson atau metode secant, yang memiliki kecepatan konvergensi yang lebih tinggi. Selain itu, metode bagi dua tidak dapat digunakan jika fungsi tidak berubah tanda pada interval awal, yang berarti proses seleksi interval awal menjadi sangat penting.

Aplikasi metode bagi dua sangat luas dan mencakup berbagai bidang, mulai dari fisika, teknik elektro, teknik mesin, hingga ekonomi. Contohnya, dalam teknik sipil, metode ini dapat digunakan untuk menghitung titik netral dalam sistem struktur lentur. Dalam ilmu ekonomi, metode bagi dua bisa diterapkan untuk mencari nilai suku bunga internal (IRR) yang membuat nilai kini bersih (NPV) sama dengan nol. Dalam pengembangan perangkat lunak atau pemrograman, metode ini juga sangat sering digunakan sebagai bagian dari modul numerik, terutama dalam bahasa seperti Python, MATLAB, C++, dan Java. Implementasinya relatif mudah dan tidak memerlukan struktur data kompleks, hanya membutuhkan iterasi sederhana berbasis logika percabangan dan perhitungan aritmatika dasar.

Sebagai ilustrasi, pertimbangkan fungsi $f(x) = x^3 - x - 2$. Kita ingin mencari akar fungsi tersebut dalam interval [1, 2]. Evaluasi awal menunjukkan $f(1)=-2$ dan $f(2)=2$, sehingga fungsi memenuhi syarat metode bagi dua. Titik tengah pertama adalah $c_1=1.5$, di mana $f(1.5)=-0.125$, masih negatif, yang berarti akar ada pada interval [1.5, 2]. Proses ini terus diulang: hitung titik tengah baru, evaluasi nilai fungsi, dan perbarui interval. Setelah beberapa iterasi, nilai pendekatan akar akan mendekati 1.521, yang merupakan akar nyata dari fungsi tersebut. Meskipun konvergensinya lambat, hasil akhirnya sangat presisi jika dilakukan hingga toleransi tertentu.

Secara implementatif, metode bagi dua dapat dituliskan dalam bentuk program komputasi sederhana. Misalnya, dalam Python, cukup

menggunakan perulangan while dan logika pembaruan nilai aaa atau bbb berdasarkan hasil evaluasi fungsi di titik tengah. Fungsi umum biasanya juga dilengkapi parameter toleransi dan jumlah iterasi maksimum untuk mencegah komputasi tak berujung akibat fungsi yang sangat mendekati datar di sekitar akar.

Pada pengajaran dan pembelajaran komputasi numerik, metode bagi dua sangat direkomendasikan sebagai titik awal untuk memperkenalkan prinsip penyelesaian persamaan nonlinear. Ini karena metode ini mengajarkan banyak konsep mendasar seperti pemilihan interval, evaluasi fungsi, penggunaan toleransi kesalahan, dan pentingnya sifat kontinuitas. Bahkan ketika mahasiswa atau peneliti akhirnya beralih ke metode yang lebih kompleks, pemahaman yang kuat tentang metode bagi dua tetap menjadi landasan penting dalam memahami bagaimana pendekatan numerik bekerja secara umum.

Dengan segala kelebihan dan keterbatasannya, metode bagi dua tetap menjadi alat yang relevan, terutama ketika kestabilan dan keandalan lebih diprioritaskan daripada kecepatan. Dalam dunia di mana solusi eksak semakin langka dan model numerik semakin dominan, metode seperti ini memberikan alternatif yang kuat dan dapat dipercaya untuk menyelesaikan masalah nonlinear dalam berbagai disiplin ilmu.

$$f(x) = x^3 - x - 2$$

Cari akar dalam interval [1,2], dan lakukan tiga iterasi.

$$f(1) = 1^3 - 1 - 2 = -2$$
$$f(2) = 2^3 - 2 - 2 = 4$$

Karena $f(1) \cdot f(2) < 0$, maka akar berada di antara 1 dan 2.

Iterasi 1

$$x_r = \frac{1+2}{2} = 1.5$$
$$f(1.5) = (1.5)^3 - 1.5 - 2 = 3.375 - 1.5 - 2 = -0.125$$

Karena $f(1.5) \cdot f(2) > 0$, akar berada di [1.5, 2].

Iterasi 2

$$x_r = \frac{1.5 + 2}{2} = 1.75$$
$$f(1.75) = (1.75)^3 - 1.75 - 2 = 5.359 - 1.75 - 2 = 1.609$$

Karena $f(1.5) \cdot f(1.75) < 0$, akar berada di **[1.5, 1.75]**.

Iterasi 3

$$x_r = \frac{1.5 + 1.75}{2} = 1.625$$
$$f(1.625) = (1.625)^3 - 1.625 - 2 = 4.29 - 1.625 - 2 = 0.665$$

Karena $f(1.5) \cdot f(1.625) < 0$, akar berada di **[1.5, 1.625]**.

Setelah tiga iterasi, kita mendekati akar dalam interval [1.5, 1.625], dengan nilai pendekatan terakhir x≈1.625. Metode ini akan terus mempersempit interval hingga mendekati akar sejati dari f(x)=0.

2. Metode Regula Falsi

Metode Regula Falsi, atau sering disebut juga *False Position Method,* merupakan salah satu metode numerik yang digunakan untuk menyelesaikan persamaan nonlinear f(x)=0 dengan cara yang lebih cerdas dibandingkan metode bagi dua. Sama seperti metode bisection, Regula Falsi termasuk dalam kategori bracketing methods, yaitu metode yang memerlukan dua titik awal aaa dan bbb sehingga f(a)×f(b)<0, artinya terdapat perubahan tanda nilai fungsi di antara dua titik tersebut. Berdasarkan Teorema Nilai Antara (*Intermediate Value Theorem*), jika fungsi f(x) bersifat kontinu dalam interval tersebut, maka dijamin terdapat setidaknya satu akar di antara aaa dan bbb. Namun, keunikan metode Regula Falsi terletak pada cara pendekatannya dalam menentukan nilai x baru (akar pendekatan), yakni dengan menggunakan persamaan garis lurus yang menghubungkan dua titik fungsi tersebut dan menghitung titik potong garis dengan sumbu x. Secara matematis, titik pendekatan akar c dihitung berdasarkan rumus:

$$c = b - \frac{f(b) \cdot (b - a)}{f(b) - f(a)}$$

Rumus ini secara geometris berarti bahwa C adalah titik potong garis lurus antara titik (a,f(a)) dan (b,f(b)) terhadap sumbu x. Dengan kata lain, alih-alih memilih titik tengah seperti dalam metode bagi dua,

metode Regula Falsi memilih titik yang diperkirakan lebih dekat dengan akar karena mempertimbangkan nilai fungsi itu sendiri. Pendekatan ini membuat metode ini memiliki potensi konvergensi yang lebih cepat daripada metode bisection, karena lebih "menyesuaikan diri" dengan bentuk kurva fungsi.

Langkah-langkah metode Regula Falsi cukup sederhana dan efisien. Pertama, tentukan dua nilai awal aaa dan bbb yang memenuhi syarat bracketing $f(a)\times f(b)<0$. Kedua, hitung nilai C menggunakan rumus di atas. Ketiga, evaluasi f(c); jika $f(c)=0$, maka C adalah akar dari fungsi. Jika tidak, perbarui nilai interval: jika $f(a)\times f(c)<0$, maka akar berada dalam interval [a,c], sehingga b diganti dengan C; jika $f(c)\times f(b)<0$, maka akar berada dalam interval [c,b], sehingga a diganti dengan C. Proses ini diulang hingga nilai absolut f(c) lebih kecil dari toleransi yang ditentukan atau panjang interval sudah sangat kecil.

Keunggulan utama metode Regula Falsi adalah kecepatannya dalam konvergensi pada banyak kasus, terutama dibandingkan dengan metode bagi dua. Karena pendekatan nilai C lebih bersifat adaptif dan tergantung pada bentuk fungsi, maka dalam fungsi-fungsi yang tidak terlalu datar atau memiliki gradien yang cukup tajam, metode ini dapat mencapai solusi lebih cepat. Sebagai contoh, dalam fungsi $f(x)=x^3-x-2$, jika digunakan Regula Falsi dengan nilai awal $a=1$ dan $b=2$, nilai akar akan mendekati $x\approx1.521$ dalam iterasi yang lebih sedikit dibandingkan metode bagi dua.

Regula Falsi juga memiliki kelemahan tertentu yang perlu diperhatikan. Salah satu kelemahan signifikan adalah potensi stagnasi konvergensi. Ini terjadi ketika salah satu dari dua titik aaa atau bbb tetap tidak berubah dalam banyak iterasi karena nilai fungsi di titik tersebut sangat kecil atau tidak berubah secara signifikan. Dalam kasus seperti ini, meskipun metode masih bekerja, konvergensinya menjadi sangat lambat dan mendekati metode bagi dua. Untuk mengatasi masalah ini, beberapa varian dari metode Regula Falsi telah dikembangkan, seperti metode Modified Regula Falsi, Illinois Method, dan Pegasus Method, yang mencoba mengoreksi titik stagnan agar proses konvergensi tetap cepat.

Pada praktik komputasi, Regula Falsi sering kali digunakan ketika metode yang lebih kompleks seperti Newton-Raphson tidak bisa diaplikasikan karena fungsi tidak terdiferensiasi dengan mudah, atau nilai turunan tidak tersedia atau tidak stabil. Karena metode ini hanya

membutuhkan evaluasi fungsi, tanpa perlu menghitung turunan, maka ia sangat cocok untuk fungsi-fungsi kompleks atau eksperimental yang diperoleh dari data empiris. Selain itu, metode ini relatif mudah diimplementasikan dalam bahasa pemrograman seperti Python, MATLAB, maupun C++. Berikut merupakan ilustrasi implementasi sederhana metode Regula Falsi dalam Python.

```python
def regula_falsi(f, a, b, tol=1e-6, max_iter=1000):
    if f(a) * f(b) >= 0:
        raise ValueError("Fungsi tidak berubah tanda di interval yang diberikan.")
    for _ in range(max_iter):
        c = b - f(b) * (b - a) / (f(b) - f(a))
        if abs(f(c)) < tol:
            return c
        if f(a) * f(c) < 0:
            b = c
        else:
            a = c
    return c
```

Dengan contoh fungsi:

```python
f = lambda x: x**3 - x - 2
akar = regula_falsi(f, 1, 2)
print(akar)  # Output mendekati 1.521
```

Dari penjelasan tersebut, dapat disimpulkan bahwa metode Regula Falsi merupakan kompromi antara kesederhanaan metode bisection dan kecepatan metode Newton-Raphson. Ia tidak secepat metode berbasis turunan, namun lebih aman karena tidak bergantung pada informasi turunan fungsi. Metode ini sangat cocok untuk digunakan dalam tahap awal pemrograman numerik atau ketika menghadapi fungsi yang sulit dianalisis secara simbolik. Dengan memilih interval awal yang tepat dan toleransi yang sesuai, metode ini mampu memberikan solusi yang akurat dan efisien untuk berbagai jenis permasalahan nonlinear yang kompleks.

$$f(x) = x^3 - x - 2$$

Dengan interval awal $[1, 2]$, lakukan tiga iterasi.

Langkah 1: Evaluasi fungsi di batas interval

$$f(1) = 1^3 - 1 - 2 = -2$$
$$f(2) = 2^3 - 2 - 2 = 4$$

Karena $f(1) \cdot f(2) < 0$, akar berada dalam interval tersebut.

Rumus Regula Falsi:

$$x_r = b - \frac{f(b)(a-b)}{f(a) - f(b)}$$

dengan $a = 1, b = 2$.

Iterasi 1:

$$x_r = 2 - \frac{4(1-2)}{-2-4} = 2 - \frac{-4}{-6} = 2 - \frac{2}{3} = 1.333$$

$$f(1.333) = (1.333)^3 - 1.333 - 2 \approx 2.37 - 1.333 - 2 = -0.963$$

Karena $f(1.333) \cdot f(2) < 0$, interval baru: $[1.333, 2]$

Iterasi 2:

$$x_r = 2 - \frac{4(1.333-2)}{-0.963-4} = 2 - \frac{-2.668}{-4.963} \approx 2 - 0.538 = 1.462$$

$$f(1.462) = (1.462)^3 - 1.462 - 2 \approx 3.125 - 1.462 - 2 = -0.337$$

Interval baru: $[1.462, 2]$

Iterasi 3:

$$x_r = 2 - \frac{4(1.462-2)}{-0.337-4} \approx 2 - \frac{-2.148}{-4.337} = 2 - 0.495 = 1.505$$

$$f(1.505) \approx -0.124$$

Setelah 3 iterasi, akar pendekatan berada di sekitar x≈1.505. Metode Regula Falsi lebih cepat dari metode Bisection karena menggunakan pendekatan garis lurus antara titik-titik.

B. Metode Newton-Raphson dan Secant

Pada penyelesaian persamaan nonlinear $f(x)$=0, metode numerik sangat penting ketika solusi eksak tidak tersedia atau sulit ditemukan.

Dua metode populer yang digunakan untuk pendekatan akar adalah metode Newton-Raphson dan metode Secant. Keduanya merupakan bagian dari metode terbuka (*open methods*), yang tidak mensyaratkan nilai awal harus mengurung akar seperti pada metode *bracketing* (misalnya metode bagi dua atau regula falsi). Karena itulah, kedua metode ini dikenal memiliki konvergensi yang lebih cepat, meskipun dengan risiko konvergensi yang tidak dijamin jika pemilihan titik awal kurang tepat.

1. Metode Newton-Raphson

Metode Newton-Raphson adalah salah satu teknik numerik paling populer dan efisien untuk mencari akar persamaan nonlinear f(x)=0. Metode ini dikenal luas dalam bidang teknik, fisika, matematika terapan, dan ilmu komputer karena konvergensinya yang cepat dan kemampuannya menyelesaikan berbagai permasalahan kompleks dengan pendekatan iteratif. Diperkenalkan oleh Sir Isaac Newton dan Joseph Raphson pada abad ke-17, metode ini memanfaatkan pendekatan kalkulus, khususnya turunan pertama fungsi, untuk memperkirakan nilai akar secara bertahap dengan tingkat ketelitian yang semakin tinggi.

Dasar teori dari metode Newton-Raphson didasarkan pada perluasan Taylor orde pertama dari fungsi f(x). Jika kita mengembangkan f(x) di sekitar titik $x=x_n$, maka dapat dituliskan:

$$f(x) \approx f(x_n) + f'(x_n)(x - x_n)$$

Dengan mengasumsikan $f(x)=0$ dan menyelesaikan persamaan tersebut untuk x, maka diperoleh rumus iteratif:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Rumus ini merupakan inti dari metode Newton-Raphson, di mana x_{n+1} adalah pendekatan akar yang lebih baik berdasarkan nilai xnx_nxn sebelumnya. Iterasi ini terus dilakukan sampai diperoleh nilai x_{n+1} yang mendekati akar sebenarnya dengan tingkat kesalahan yang bisa ditoleransi.

Salah satu keunggulan utama metode Newton-Raphson adalah kecepatan konvergensinya yang kuadratik, artinya jika pendekatan awal cukup dekat dengan akar sebenarnya, maka jumlah digit benar dari hasil perhitungan akan bertambah dua kali lipat di setiap iterasi. Hal ini

membuat metode ini sangat efisien dibandingkan metode lain seperti bisection atau regula falsi yang hanya konvergen secara linier. Namun, kecepatan ini hanya dapat dicapai apabila kondisi ideal terpenuhi, seperti turunan fungsi tidak mendekati nol dan nilai awal tidak terlalu jauh dari akar.

Pada implementasinya, metode Newton-Raphson memerlukan dua komponen utama: nilai fungsi f(x) dan turunan pertamanya f'(x). Ini menjadi kekuatan sekaligus keterbatasan metode. Pada satu sisi, informasi turunan memberikan arah dan kecepatan pergerakan menuju akar, sehingga konvergensi menjadi sangat efisien. Namun di sisi lain, metode ini menjadi sulit atau tidak praktis jika turunan fungsi tidak diketahui secara eksplisit, sulit dihitung, atau fungsi tidak terdiferensialkan dengan baik di sekitar titik yang sedang dianalisis. Dalam kasus semacam ini, pengguna dapat mempertimbangkan metode alternatif seperti metode secant, yang tidak memerlukan turunan eksplisit.

Contoh sederhana penerapan metode Newton-Raphson adalah pada fungsi $f(x)=x^3 - x - 2$, di mana akar nyata dari persamaan ini terletak di sekitar $x\approx1.521$. Dengan memilih titik awal $x_0=1.5x$, kita dapat menghitung $f(x_0)=-0.125$ dan $f'(x_0)=3(1.5)2-1=5.75$. Maka:

$$x_1 = 1.5 - \frac{-0.125}{5.75} \approx 1.5217$$

Dengan hanya satu iterasi, nilai pendekatan akar sudah sangat dekat dengan solusi sebenarnya. Dalam beberapa iterasi berikutnya, hasil perhitungan akan semakin mendekati akar sejati dengan tingkat kesalahan yang sangat kecil.

Meski sangat efektif, metode Newton-Raphson memiliki sejumlah potensi masalah. Jika nilai awal terlalu jauh dari akar, atau jika fungsi memiliki turunan nol di titik tertentu (seperti di puncak atau lembah grafik), maka perhitungan bisa tidak stabil atau bahkan gagal konvergen. Dalam beberapa kasus ekstrem, iterasi dapat menyimpang jauh dari akar sebenarnya, atau masuk ke dalam siklus tak berujung yang tidak menghasilkan konvergensi. Oleh karena itu, pemilihan nilai awal yang baik dan pemahaman bentuk fungsi menjadi sangat krusial dalam menjamin keberhasilan metode ini.

Pada praktik pemrograman, implementasi metode Newton-Raphson relatif mudah. Bahasa seperti Python, MATLAB, atau C++

menyediakan cara cepat untuk menghitung fungsi dan turunannya. Sebagai contoh, implementasi sederhana dalam Python dapat dituliskan sebagai berikut:

```
def newton_raphson(f, df, x0, tol=1e-6, max_iter=100):
    for _ in range(max_iter):
        x1 = x0 - f(x0) / df(x0)
        if abs(x1 - x0) < tol:
            return x1
        x0 = x1
    return x0
```

Dalam penggunaan dunia nyata, metode Newton-Raphson banyak diterapkan dalam berbagai bidang. Dalam teknik sipil, digunakan untuk menghitung deformasi struktur nonlinear. Dalam bidang keuangan, digunakan untuk menghitung akar dari persamaan nilai kini bersih (NPV) dalam penentuan IRR (*Internal Rate of Return*). Dalam bidang optimisasi dan pembelajaran mesin, metode ini menjadi dasar bagi algoritma yang lebih kompleks seperti gradient descent dan Newton's optimization method dalam pelatihan model.

$$f(x) = x^3 - x - 2$$

Gunakan **tebakan awal** $x_0 = 1.5$, dan lakukan **tiga iterasi**.

$$f(x) = x^3 - x - 2 \qquad f'(x) = 3x^2 - 1$$

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Iterasi 1:

$$x_0 = 1.5$$
$$f(1.5) = 3.375 - 1.5 - 2 = -0.125$$
$$f'(1.5) = 3(1.5)^2 - 1 = 6.75 - 1 = 5.75$$

$$x_1 = 1.5 - \frac{-0.125}{5.75} \approx 1.5 + 0.0217 = 1.5217$$

Iterasi 2:

$$f(1.5217) \approx (1.5217)^3 - 1.5217 - 2 \approx 3.525 - 1.5217 - 2 = 0.0033$$
$$f'(1.5217) \approx 3(1.5217)^2 - 1 \approx 5.948$$

$$x_2 = 1.5217 - \frac{0.0033}{5.948} \approx 1.5217 - 0.000555 = 1.5211$$

Iterasi 3:

$$f(1.5211) \approx 0.00001, \quad f'(1.5211) \approx 5.943$$
$$x_3 = 1.5211 - \frac{0.00001}{5.943} \approx 1.5211$$

Setelah tiga iterasi, diperoleh akar pendekatan x≈1.5211. Metode Newton-Raphson sangat cepat konvergen jika tebakan awal dekat dengan akar dan turunan tidak mendekati nol.

2. Metode Secant

Metode Secant merupakan salah satu metode numerik yang digunakan untuk menyelesaikan persamaan nonlinear dalam bentuk f(x)=0, dan secara khusus merupakan variasi dari metode Newton-Raphson yang tidak memerlukan turunan eksplisit dari fungsi yang dianalisis. Metode ini menjadi alternatif praktis ketika fungsi f(x) terlalu kompleks atau tidak memiliki turunan yang dapat dihitung dengan mudah. Oleh karena itu, metode Secant menjadi sangat relevan dalam banyak aplikasi komputasi teknik, fisika, dan ekonomi, di mana bentuk fungsi sering kali tidak diketahui secara simbolik atau hanya tersedia dalam bentuk data numerik.

Secara konseptual, metode Secant memanfaatkan pendekatan turunan numerik berdasarkan dua titik pendekatan sebelumnya. Jika pada metode Newton-Raphson digunakan turunan analitik f′(x), maka pada metode Secant, turunan didekati dengan:

$$f'(x_n) \approx \frac{f(x_n) - f(x_{n-1})}{x_n - x_{n-1}}$$

Rumus ini kemudian disubstitusikan ke dalam formula Newton-Raphson, sehingga diperoleh rumus iteratif metode Secant:

$$x_{n+1} = x_n - f(x_n) \cdot \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}$$

Dengan demikian, metode ini hanya memerlukan dua nilai awal, yaitu x_0 dan x_1, yang digunakan untuk memulai iterasi dalam mencari akar fungsi. Berbeda dengan metode bracketing seperti metode bisection atau regula falsi, metode Secant termasuk dalam kategori *open methods*, yang tidak mengharuskan kedua nilai awal mengurung akar (tidak harus $f(x_0) \times f(x_1) < 0$.

Kekuatan utama dari metode Secant terletak pada kesederhanaan perhitungannya dan kecepatan konvergensi yang relatif tinggi dibandingkan metode-metode bracketing. Meskipun kecepatan konvergensi metode Secant tidak secepat Newton-Raphson yang bersifat kuadratik, metode ini memiliki konvergensi super-linear dengan laju mendekati 1.618, yakni golden ratio. Artinya, dalam banyak kasus, metode ini tetap memberikan hasil yang cukup cepat dan efisien, tanpa syarat ketersediaan turunan fungsi.

Sebagai ilustrasi, pertimbangkan fungsi nonlinear $f(x)=x^3-x-2$. Kita ingin mencari akar fungsi tersebut menggunakan metode Secant. Misalkan dua nilai awal adalah $x_0=1$ dan $x_1=2$. Maka, $f(1)=-2$ dan $f(2)=2$. Iterasi pertama akan menghasilkan:

$$x_2 = 2 - \frac{2 \cdot (2-1)}{2-(-2)} = 2 - \frac{2}{4} = 1.5$$

Dengan terus melanjutkan proses iteratif menggunakan dua nilai pendekatan terakhir, kita akan mendekati akar sejati dari fungsi tersebut, yaitu sekitar $x \approx 1.521$. Meskipun pada contoh ini metode regula falsi juga dapat digunakan, metode Secant cenderung lebih cepat karena tidak memerlukan validasi tanda fungsi untuk memperbarui interval.

Metode Secant bukan tanpa kelemahan. Salah satu kelemahan utama adalah tidak adanya jaminan konvergensi. Karena metode ini tidak menggunakan prinsip bracketing, maka jika nilai awal tidak dipilih dengan tepat atau jika fungsi memiliki perilaku tak menentu (seperti perubahan kemiringan ekstrem atau osilasi lokal), maka iterasi dapat menyimpang jauh dari akar atau bahkan tidak konvergen sama sekali. Selain itu, jika dua nilai pendekatan menghasilkan $f(x_n)=f(x_{n-1})$, maka metode akan gagal karena menghasilkan pembagian nol. Oleh karena itu, kontrol terhadap nilai-nilai awal dan pemeriksaan stabilitas perhitungan menjadi aspek penting dalam penerapan metode ini.

Pada praktiknya, metode Secant dapat diimplementasikan dengan sangat mudah dalam bahasa pemrograman seperti Python, MATLAB, atau C++. Contoh implementasi sederhana metode ini dalam Python adalah sebagai berikut:

```python
def secant(f, x0, x1, tol=1e-6, max_iter=100):
    for _ in range(max_iter):
        f0, f1 = f(x0), f(x1)
        if f1 - f0 == 0:
            raise ValueError("Terjadi pembagian dengan nol.")
        x2 = x1 - f1 * (x1 - x0) / (f1 - f0)
        if abs(x2 - x1) < tol:
            return x2
        x0, x1 = x1, x2
    return x1
```

Penggunaan metode ini sangat cocok untuk fungsi-fungsi yang tidak diketahui bentuk turunannya, seperti fungsi empiris yang diperoleh dari hasil eksperimen atau pengukuran. Misalnya, dalam rekayasa sistem kontrol, kita bisa menggunakannya untuk menyetel parameter sistem berdasarkan fungsi karakteristik hasil simulasi. Dalam ekonomi, metode ini bisa digunakan untuk menghitung tingkat diskonto dari arus kas dengan model yang tidak memiliki turunan eksplisit. Dalam ilmu data, metode Secant bahkan dapat digunakan dalam pencarian nilai minimum fungsi loss secara numerik.

Dari perspektif pendidikan, metode Secant juga sangat bermanfaat untuk memperkenalkan konsep turunan numerik dan pendekatan iteratif. Mahasiswa dapat memahami bagaimana perbedaan antara metode terbuka dan tertutup memengaruhi hasil akhir, serta menyadari pentingnya pemilihan nilai awal. Selain itu, metode ini menjadi jembatan alami antara metode reguler dan metode berbasis turunan seperti Newton-Raphson.

$$f(x) = x^3 - x - 2$$

Gunakan dua tebakan awal:

$x_0 = 1$ dan $x_1 = 2$,

lalu lakukan **tiga iterasi**.

$$x_{n+1} = x_n - f(x_n) \cdot \frac{x_n - x_{n-1}}{f(x_n) - f(x_{n-1})}$$

$$f(x) = x^3 - x - 2$$

$$f(1) = 1 - 1 - 2 = -2$$
$$f(2) = 8 - 2 - 2 = 4$$

Maka,

Iterasi 1:

$$x_0 = 1, \quad x_1 = 2$$
$$x_2 = 2 - 4 \cdot \frac{2-1}{4-(-2)} = 2 - 4 \cdot \frac{1}{6} = 2 - \frac{2}{3} = 1.333$$
$$f(1.333) = (1.333)^3 - 1.333 - 2 \approx 2.37 - 1.333 - 2 = -0.963$$

Iterasi 2:

$$x_3 = 1.333 - (-0.963) \cdot \frac{1.333 - 2}{-0.963 - 4} = 1.333 - (-0.963) \cdot \frac{-0.667}{-4.963}$$
$$\approx 1.333 - (0.963 \cdot 0.134) \approx 1.333 - 0.129 = 1.204$$

Iterasi 3:

$$f(1.204) \approx -0.400, \quad f(1.333) \approx -0.963$$
$$x_4 = 1.204 - (-0.4) \cdot \frac{1.204 - 1.333}{-0.4 - (-0.963)}$$
$$\approx 1.204 - (-0.4) \cdot (-0.129/0.563) \approx 1.204 - 0.092 = 1.112$$

Setelah tiga iterasi, pendekatan akar dari f(x)=0 berada di sekitar x≈1.112. Metode secant cepat dan tidak memerlukan turunan, berbeda dengan Newton-Raphson.

C. Konvergensi dan Stabilitas Solusi

Pada bidang komputasi numerik dan penyelesaian persamaan matematis secara numerik, dua konsep yang sangat fundamental adalah konvergensi dan stabilitas solusi. Kedua konsep ini menentukan keberhasilan dan keandalan suatu metode numerik dalam memberikan solusi yang mendekati nilai sebenarnya. Sebuah metode yang akurat tetapi tidak stabil, atau metode yang stabil tetapi tidak konvergen, tidak akan dapat digunakan secara efektif dalam praktik nyata. Oleh karena itu, memahami konvergensi dan stabilitas secara mendalam sangat

penting, baik dalam pengembangan algoritma maupun dalam penerapannya untuk menyelesaikan masalah ilmiah dan rekayasa.

1. Konvergensi

Konvergensi merupakan konsep fundamental dalam komputasi numerik yang mengukur seberapa efektif suatu metode numerik dalam menghampiri solusi sebenarnya dari sebuah masalah matematis. Dalam konteks penyelesaian persamaan nonlinear, sistem linier, maupun persamaan diferensial, konvergensi menentukan apakah urutan solusi mendekati nilai yang benar saat jumlah iterasi bertambah. Dengan kata lain, suatu metode dikatakan konvergen jika hasil pendekatan numeriknya semakin dekat ke solusi eksak seiring bertambahnya iterasi atau penyempurnaan partisi numerik. Konsep ini tidak hanya penting secara teoritis, tetapi juga sangat menentukan keberhasilan metode numerik dalam berbagai aplikasi dunia nyata seperti simulasi teknik, optimisasi, pemodelan ilmiah, dan pemrosesan data.

Menurut Burden dan Faires (2010), konvergensi suatu metode numerik secara formal dapat didefinisikan sebagai berikut: jika terdapat suatu solusi eksak x^* dan urutan hasil pendekatan $\{x_n\}$, maka metode dikatakan konvergen jika:

$$\lim_{n \to \infty} x_n = x^*$$

Artinya, seiring dengan bertambahnya iterasi atau penyempurnaan skema (seperti langkah waktu atau ukuran grid), solusi numerik x_n semakin dekat ke nilai x^*. Dalam praktiknya, pengguna akan menghentikan proses iterasi pada titik di mana selisih antara dua iterasi berturut-turut sudah berada di bawah toleransi kesalahan tertentu, misalnya $|x_{n+1} - x_n| < \epsilon$.

Salah satu aspek penting dari konvergensi adalah kecepatan konvergensi atau rate of convergence. Ini menjelaskan seberapa cepat pendekatan menuju solusi sebenarnya. Terdapat beberapa tingkatan konvergensi yang umum digunakan dalam teori numerik:

$$|x_{n+1} - x^*| \leq C|x_n - x^*| \quad \text{dengan} \quad 0 < C < 1$$

$$|x_{n+1} - x^*| \leq C|x_n - x^*|^2$$

Kecepatan konvergensi sangat memengaruhi efisiensi komputasi. Dalam metode iteratif, semakin tinggi laju konvergensi,

semakin sedikit iterasi yang dibutuhkan untuk mencapai presisi tertentu, sehingga waktu komputasi lebih singkat dan penggunaan sumber daya menjadi lebih efisien. Oleh karena itu, analisis konvergensi sering kali dilakukan sebelum memilih atau merancang metode numerik yang akan digunakan untuk menyelesaikan suatu masalah.

Tidak semua metode dijamin konvergen dalam segala situasi. Kondisi awal, sifat fungsi, dan parameter numerik sangat memengaruhi konvergensi. Misalnya, dalam metode Newton-Raphson, konvergensi kuadratik hanya dapat dicapai jika titik awal cukup dekat dengan akar dan fungsi memiliki turunan yang tidak nol di sekitar akar. Jika tidak, iterasi bisa menyimpang jauh dan bahkan tidak pernah mendekati akar (divergen). Hal ini menunjukkan bahwa analisis konvergensi tidak hanya bergantung pada rumus iteratif, tetapi juga pada pemahaman sifat masalah yang sedang diselesaikan.

Pada konteks penyelesaian persamaan diferensial numerik, konvergensi memiliki definisi yang sedikit berbeda tetapi prinsipnya serupa. Misalnya, dalam penyelesaian persamaan diferensial biasa (ODE) dengan metode numerik seperti Euler atau Runge-Kutta, konvergensi mengacu pada apakah solusi numerik mendekati solusi eksak ketika langkah waktu (Δt) didekati ke nol. Suatu metode dikatakan konvergen jika kesalahan total (global error) mendekati nol ketika ukuran langkah mendekati nol. Oleh karena itu, dalam banyak kasus, konvergensi dinyatakan sebagai fungsi dari ukuran langkah: Error$\sim O(hp)$p), di mana p menunjukkan orde akurasi dari metode tersebut.

Hubungan antara konsistensi, stabilitas, dan konvergensi juga dijelaskan dalam Teorema Lax (*Lax Equivalence Theorem*), yang menyatakan bahwa untuk skema linear yang stabil dan konsisten, maka metode tersebut pasti konvergen. Ini berarti bahwa konvergensi tidak bisa dilihat secara terpisah dari aspek stabilitas dan keakuratan metode. Jika suatu metode tidak stabil atau tidak konsisten, maka meskipun secara matematis menjanjikan, metode tersebut bisa gagal menghampiri solusi sebenarnya.

Pada praktik rekayasa dan ilmu terapan, konvergensi tidak hanya menjadi syarat teoritis, melainkan juga panduan penting dalam validasi simulasi numerik. Misalnya, dalam simulasi struktur bangunan menggunakan metode elemen hingga (*finite element method*), hasil yang

diperoleh pada model kasar (*coarse mesh*) harus diverifikasi dengan memperkecil ukuran elemen (*mesh refinement*). Jika solusi tidak berubah secara signifikan saat mesh diperhalus, maka solusi dianggap konvergen dan valid. Konsep ini juga diterapkan dalam simulasi fluida, analisis medan elektromagnetik, dan berbagai bidang yang memerlukan pendekatan numerik berbasis grid.

Secara praktis, untuk mengukur konvergensi, para insinyur dan ilmuwan biasanya melakukan studi konvergensi (*convergence study*), yaitu dengan mencoba berbagai nilai parameter numerik (misalnya ukuran grid atau langkah waktu) dan membandingkan hasilnya. Jika perbedaan hasil menjadi semakin kecil, dan mendekati nilai tetap, maka metode dianggap telah mencapai konvergensi numerik.

Konvergensi adalah fondasi dari setiap algoritma numerik yang andal. Metode yang tidak konvergen tidak dapat dipercaya dalam menghasilkan hasil yang benar, betapapun canggih atau cepatnya metode tersebut. Oleh karena itu, dalam pengembangan algoritma, simulasi numerik, dan penerapan di dunia nyata, analisis konvergensi harus menjadi bagian utama dalam evaluasi keakuratan dan efisiensi metode numerik. Pemahaman yang baik tentang karakteristik konvergensi memungkinkan praktisi memilih metode yang paling tepat sesuai dengan jenis masalah, struktur matematis fungsi, serta keterbatasan sumber daya komputasi yang tersedia.

Seorang mahasiswa menggunakan **metode Newton-Raphson** untuk menyelesaikan persamaan nonlinear:

$$f(x) = x^3 - x - 1$$

Ia memulai dengan tebakan awal $x_0 = 1$ dan menghitung hingga x_3. Hasil per iterasinya:

- $x_1 = 1.5$
- $x_2 = 1.3478$
- $x_3 = 1.3247$

Tentukan apakah metode ini konvergen, dan hitung galat relatif pada iterasi ke-3.

Gunakan rumus galat relatif pada iterasi ke-3:

$$\text{Galat Relatif} = \left| \frac{x_3 - x_2}{x_3} \right| \times 100\%$$

Substitusi nilai:

$$= \left| \frac{1.3247 - 1.3478}{1.3247} \right| \times 100\% = \left| \frac{-0.0231}{1.3247} \right| \times 100\% \approx 1.74\%$$

Kesimpulan

- Karena nilai x_n dari iterasi ke iterasi semakin mendekati suatu nilai tetap, metode ini menunjukkan konvergensi.
- Galat relatif sebesar 1.74% menunjukkan bahwa hasil sudah cukup dekat, meskipun belum sangat presisi.
- Untuk keperluan praktis, konvergensi biasanya diterima jika galat relatif < 1% (tergantung toleransi yang ditentukan pengguna).

2. Stabilitas Solusi

Stabilitas solusi merupakan konsep kunci dalam komputasi numerik yang berkaitan erat dengan keandalan dan ketahanan suatu metode numerik terhadap gangguan atau kesalahan kecil dalam proses perhitungan. Dalam konteks penyelesaian masalah numerik baik itu persamaan aljabar, diferensial, maupun sistem linear stabilitas mengukur seberapa besar efek kesalahan kecil pada data input atau pembulatan selama iterasi dapat mempengaruhi solusi akhir. Dengan kata lain, stabilitas solusi menggambarkan apakah sebuah metode mampu menjaga agar kesalahan kecil tidak berkembang secara signifikan sehingga menyebabkan penyimpangan besar pada hasil akhir. Konsep ini sangat penting dalam dunia nyata karena setiap komputasi yang dilakukan dengan komputer digital pasti mengandung kesalahan pembulatan akibat keterbatasan presisi representasi bilangan *floating point*.

Menurut Chapra dan Canale (2015), stabilitas numerik adalah kemampuan metode untuk membatasi pertumbuhan kesalahan selama proses iteratif berlangsung. Kesalahan tersebut bisa berasal dari dua sumber utama: (1) kesalahan pembulatan, yang terjadi ketika hasil perhitungan dibatasi oleh jumlah digit yang dapat direpresentasikan komputer, dan (2) kesalahan gangguan input, yaitu ketidakakuratan pada data awal atau nilai awal iterasi. Dalam algoritma yang tidak stabil, kesalahan-kesalahan kecil ini bisa diperkuat oleh struktur perhitungan

hingga mengakibatkan hasil akhir yang menyimpang jauh dari solusi sebenarnya.

Salah satu ilustrasi paling umum dari pentingnya stabilitas adalah pada metode numerik untuk menyelesaikan persamaan diferensial biasa (ODE). Misalnya, dalam metode Euler eksplisit, hasil iterasi sangat bergantung pada ukuran langkah waktu (Δt). Jika Δt terlalu besar, kesalahan lokal yang terjadi dalam satu iterasi bisa diperkuat secara eksponensial di iterasi-iterasi berikutnya. Akibatnya, meskipun metode Euler secara teori konsisten (yakni mampu mendekati solusi sebenarnya jika langkah waktu cukup kecil), tetapi bila digunakan dengan langkah waktu yang tidak sesuai, hasil akhirnya bisa menjadi tidak masuk akal atau bahkan mengalami *numerical blow-up*. Oleh karena itu, untuk metode eksplisit seperti Euler, hanya nilai-nilai langkah waktu tertentu yang menjamin stabilitas solusi wilayah nilai ini disebut daerah stabilitas.

Pada konteks metode numerik untuk sistem persamaan linear, stabilitas berkaitan dengan kondisi matriks yang digunakan dalam perhitungan. Jika sebuah matriks koefisien dari sistem linear sangat sensitif terhadap perubahan kecil dalam data (disebut *ill-conditioned*), maka solusi yang dihasilkan dapat berubah drastis bahkan ketika perubahan data sangat kecil. Kondisi seperti ini biasanya dinyatakan dalam bentuk angka kondisi (*condition number*). Matriks dengan angka kondisi tinggi menunjukkan bahwa metode yang digunakan untuk menyelesaikannya bisa sangat tidak stabil. Oleh karena itu, stabilitas solusi dalam sistem linear sangat dipengaruhi oleh struktur aljabar dari sistem tersebut, bukan hanya oleh metode yang digunakan.

Pada metode iteratif, seperti metode Gauss-Seidel atau Jacobi untuk sistem linear, stabilitas juga berperan penting. Algoritma iteratif harus dirancang agar error tidak diperbesar pada setiap langkah, melainkan diminimalkan. Ketika algoritma terus menerus memperkuat error dari langkah sebelumnya, maka proses iterasi akan menyimpang dari hasil sebenarnya dan menjadi divergen. Oleh karena itu, analisis spektral radius dari matriks iterasi sering digunakan untuk menilai stabilitas suatu metode iteratif. Jika spektral radius lebih kecil dari satu, maka metode dijamin stabil dan konvergen.

Salah satu aspek penting lainnya dari stabilitas adalah dalam simulasi waktu atau pemodelan dinamika sistem yang berlangsung

selama periode tertentu. Dalam konteks ini, stabilitas menentukan apakah solusi numerik akan terus mengikuti perilaku sistem aktual atau mengalami deviasi seiring waktu. Sebagai contoh, dalam simulasi pergerakan partikel atau simulasi dinamika fluida, kesalahan kecil pada posisi atau kecepatan bisa menyebabkan solusi yang menyimpang jauh jika metode yang digunakan tidak stabil terhadap waktu. Oleh karena itu, para insinyur dan ilmuwan sering kali melakukan analisis sensitivitas dan uji kestabilan waktu sebelum menggunakan hasil simulasi untuk pengambilan keputusan.

Stabilitas solusi juga sangat penting dalam pemrosesan sinyal dan analisis numerik data eksperimen. Ketika data input mengandung noise atau gangguan, metode numerik yang tidak stabil dapat memperbesar efek noise tersebut dan menghasilkan kesimpulan yang salah. Oleh karena itu, dalam bidang seperti rekonstruksi citra, pemodelan keuangan, dan pembelajaran mesin, pemilihan metode yang stabil menjadi keharusan untuk menjaga validitas hasil akhir.

Pada praktiknya, untuk menjamin stabilitas solusi, beberapa pendekatan umum digunakan: (1) penggunaan metode implisit pada sistem diferensial, seperti metode Backward Euler yang dikenal lebih stabil dibandingkan metode eksplisit; (2) penyesuaian parameter numerik, seperti langkah waktu atau toleransi kesalahan; (3) pengondisian ulang data atau sistem, untuk menghindari sistem *ill-conditioned*; dan (4) penerapan analisis kestabilan teoritis terhadap metode numerik sebelum diimplementasikan.

3. Studi Kasus Komputasional

Di era modern yang semakin mengandalkan teknologi untuk mendukung pengambilan keputusan, pendekatan komputasional telah menjadi elemen penting dalam analisis sistem kompleks, termasuk dalam isu lingkungan. Salah satu permasalahan yang krusial di banyak kota besar di dunia adalah polusi udara. Emisi dari kendaraan bermotor, industri, dan pembakaran sampah memberikan kontribusi signifikan terhadap pencemaran udara, yang berdampak pada kesehatan manusia dan lingkungan. Untuk itu, studi kasus ini mengangkat permasalahan penyebaran polusi udara di kawasan perkotaan dengan menggunakan pendekatan komputasi numerik berbasis metode *Finite Difference* dan pemrograman Python, guna memprediksi distribusi konsentrasi polutan

dalam suatu wilayah dan membantu pemerintah merumuskan kebijakan mitigasi yang lebih akurat.

4. Latar Belakang Permasalahan

Kawasan padat penduduk dengan kepadatan kendaraan tinggi menghasilkan emisi karbon monoksida (CO), nitrogen dioksida (NO_2), dan partikel-partikel berbahaya (PM2.5) dalam jumlah besar. Dalam banyak kasus, sensor pengukuran polusi udara hanya dipasang di beberapa titik tertentu, sehingga informasi distribusi spasial dan temporal polutan bersifat terbatas. Oleh karena itu, dibutuhkan model simulasi berbasis komputasi untuk memperkirakan bagaimana polutan menyebar di wilayah tersebut dalam kurun waktu tertentu, dengan mempertimbangkan pengaruh kecepatan angin, arah angin, dan perubahan konsentrasi emisi. Simulasi ini tidak hanya berfungsi untuk menggambarkan kondisi saat ini, tetapi juga digunakan untuk memprediksi skenario masa depan, seperti dampak pembangunan jalan baru atau pengurangan volume kendaraan.

5. Formulasi Masalah Secara Matematis

Model dasar yang digunakan dalam studi ini adalah Persamaan Adveksi-Difusi dua dimensi yang menyatakan perubahan konsentrasi polutan $C(x,y,t)$ seiring waktu:

$$\frac{\partial C}{\partial t} + u\frac{\partial C}{\partial x} + v\frac{\partial C}{\partial y} = D\left(\frac{\partial^2 C}{\partial x^2} + \frac{\partial^2 C}{\partial y^2}\right) + S(x,y,t)$$

Di mana:

- C adalah konsentrasi polutan pada titik ruang dan waktu tertentu.
- u dan v adalah komponen kecepatan angin arah x dan y.
- D adalah koefisien difusi.
- $S(x,y,t)$ adalah fungsi sumber polusi, misalnya dari kendaraan atau industri.

Persamaan ini mencerminkan bahwa polutan menyebar karena efek adveksi oleh angin, difusi karena perbedaan konsentrasi, dan bertambah karena sumber emisi.

Untuk menyelesaikannya secara numerik, persamaan diferensial parsial tersebut didiskretisasi menggunakan skema *Finite Difference* eksplisit. Domain wilayah dibagi menjadi grid dua dimensi, dan waktu

dipecah dalam langkah-langkah kecil. Komputasi dilakukan untuk menghitung konsentrasi polutan di setiap titik grid pada setiap langkah waktu.

6. Implementasi Komputasional

Simulasi ini diimplementasikan menggunakan bahasa pemrograman Python karena kemudahan dalam manipulasi matriks serta ketersediaan pustaka ilmiah seperti NumPy dan Matplotlib. Wilayah simulasi dibuat dalam ukuran 1 km x 1 km yang dibagi menjadi 100 x 100 grid, dengan setiap grid berukuran 10 meter. Kecepatan angin diatur tetap, misalnya 2 m/s ke arah timur dan 1 m/s ke arah utara. Emisi dari kendaraan dimodelkan sebagai sumber tetap yang berada di tengah kota. Potongan kode Python untuk skema numerik eksplisit sebagai berikut:

```
for t in range(nt):
    C_new = C.copy()
    for i in range(1, nx-1):
        for j in range(1, ny-1):
            advection_x = -u * (C[i+1, j] - C[i-1, j]) / (2 * dx)
            advection_y = -v * (C[i, j+1] - C[i, j-1]) / (2 * dy)
            diffusion = D * ((C[i+1, j] - 2*C[i, j] + C[i-1, j]) / dx**2 +
                             (C[i, j+1] - 2*C[i, j] + C[i, j-1]) / dy**2)
            C_new[i, j] = C[i, j] + dt * (advection_x + advection_y + diffusion + S[i, j])
    C = C_new.copy()
```

Visualisasi distribusi konsentrasi polusi dilakukan dengan menggunakan Matplotlib. Hasil simulasi menunjukkan bahwa polusi tertinggi terkonsentrasi di dekat sumber emisi dan menyebar mengikuti arah angin. Distribusi konsentrasi ini dapat dipetakan dalam bentuk kontur warna, sehingga memberikan pemahaman spasial yang jelas kepada pengambil kebijakan.

7. Analisis Hasil dan Validasi

Setelah simulasi dijalankan selama 24 jam waktu simulasi, diperoleh peta distribusi konsentrasi polusi pada setiap titik grid. Nilai tertinggi terdeteksi di wilayah pusat kota, sementara nilai terendah berada di pinggiran kota, mengikuti arah dominan angin. Selain itu, ketika skenario penurunan emisi sebesar 50% dari sektor kendaraan diterapkan, konsentrasi polutan menurun secara signifikan, terutama pada area padat lalu lintas. Untuk memvalidasi model, data sensor nyata dari stasiun pemantau kualitas udara kota digunakan. Hasil simulasi dibandingkan dengan data aktual dan menunjukkan deviasi kurang dari 10%, yang menandakan bahwa model cukup akurat dalam memprediksi

pola sebaran polusi udara. Dalam dunia komputasi lingkungan, selisih di bawah 15% dianggap masih dalam batas yang bisa diterima untuk model prediktif.

8. Relevansi dan Implikasi Kebijakan

Studi kasus ini menunjukkan bahwa pendekatan komputasional berbasis metode numerik dapat memberikan informasi spasial yang jauh lebih lengkap daripada data pengamatan saja. Pemerintah daerah dapat memanfaatkan hasil simulasi ini untuk:

a. Menentukan zona emisi rendah (*low emission zones*).
b. Menyesuaikan arah pembangunan jalan agar tidak memusatkan lalu lintas di area padat.
c. Menentukan lokasi strategis pemasangan alat pemantau kualitas udara.
d. Mensimulasikan skenario darurat jika terjadi lonjakan polusi akibat kebakaran hutan atau kecelakaan industri.

Model ini juga dapat diperluas ke simulasi multi-pollutan dan integrasi data waktu nyata dari sensor *Internet of Things* (IoT), sehingga prediksi menjadi lebih responsif dan adaptif terhadap kondisi terkini.

9. Kesimpulan

Studi kasus ini membuktikan bahwa pendekatan komputasi numerik memiliki peran penting dalam memahami dan mengelola permasalahan lingkungan yang kompleks seperti polusi udara. Dengan menggunakan model matematis yang diformulasikan dalam persamaan diferensial parsial dan diselesaikan menggunakan metode numerik *Finite Difference*, simulasi distribusi polutan dapat dilakukan secara efisien dan akurat. Implementasi berbasis Python membuat proses ini dapat diakses oleh banyak pihak tanpa memerlukan perangkat lunak mahal. Lebih dari itu, pendekatan ini memperlihatkan bagaimana teknologi komputasi dapat berperan langsung dalam mendukung kebijakan berbasis data (*data-driven policy*) untuk meningkatkan kualitas hidup masyarakat perkotaan. Ke depannya, integrasi metode ini dengan data penginderaan jauh, *big data*, dan kecerdasan buatan akan memperkuat kapasitas pemodelan lingkungan yang lebih dinamis dan adaptif.

BAB VIII
PERSAMAAN DIFERENSIAL BIASA (PDB)

Persamaan Diferensial Biasa (PDB) merupakan salah satu pilar utama dalam matematika terapan yang berperan penting dalam memahami dan memodelkan dinamika berbagai fenomena alam maupun rekayasa. Dari gerak planet di langit hingga penyebaran penyakit menular, dari getaran mekanik hingga dinamika keuangan, PDB menjadi alat matematis yang tak tergantikan dalam menjelaskan perubahan variabel terhadap waktu atau parameter lainnya. Dalam konteks komputasi numerik, penyelesaian PDB secara analitik sering kali tidak memungkinkan, sehingga pendekatan numerik menjadi solusi yang sangat vital. Melalui metode seperti Euler, Runge-Kutta, dan multi-step methods, kita mampu memperoleh pendekatan solusi yang cukup akurat dengan efisiensi komputasi yang tinggi. Pemahaman terhadap teori dasar, kestabilan metode, serta implementasi algoritma dalam platform pemrograman modern seperti Python atau MATLAB menjadi kompetensi penting bagi mahasiswa, peneliti, dan praktisi.

A. Pengenalan PDB dan Model Aplikatif

Persamaan Diferensial Biasa (PDB) atau *Ordinary Differential Equation* (ODE) merupakan salah satu bentuk persamaan matematika yang melibatkan turunan suatu fungsi terhadap satu variabel bebas. Dalam bentuk paling umum, PDB menyatakan hubungan antara fungsi tak diketahui dan turunannya, yang sangat berguna untuk memodelkan

fenomena dinamis dalam berbagai disiplin ilmu. Tidak hanya terbatas pada fisika dan teknik, PDB juga banyak diaplikasikan dalam biologi, ekonomi, ekologi, kedokteran, hingga ilmu sosial.

Menurut Zill dan Wright (2017) dalam bukunya "*Differential Equations with Boundary-Value Problems*", Persamaan Diferensial Biasa adalah persamaan yang mengandung turunan dari suatu fungsi dengan satu variabel bebas, berbeda dengan *Partial Differential Equations* (PDE) yang melibatkan turunan parsial dari fungsi beberapa variabel. Secara umum, PDB dapat dituliskan dalam bentuk:

$$\frac{dy}{dx} = f(x, y)$$

atau dalam bentuk eksplisit sebagai fungsi dari turunan:

$$F(x, y, y', y'', ..., y^{(n)}) = 0$$

Klasifikasi PDB (Persamaan Diferensial Biasa) didasarkan pada beberapa kategori:

- Orde: Derajat tertinggi dari turunan yang terdapat dalam persamaan.
- Linearitas: Suatu persamaan dikatakan *linear* jika tidak ada perkalian antara fungsi tak diketahui dan turunannya.
- Homogenitas: Suatu persamaan dikatakan *homogen* jika semua suku bergantung pada fungsi dan turunannya, tanpa adanya konstanta bebas.

1. Fisika dan Teknik

Pada bidang fisika dan teknik, Persamaan Diferensial Biasa (PDB) merupakan alat fundamental untuk memodelkan berbagai fenomena dinamis yang melibatkan perubahan terhadap waktu atau ruang dalam sistem fisis. Salah satu contoh paling mendasar adalah dalam mekanika klasik, di mana hukum kedua Newton yang berbunyi F = ma dapat diubah menjadi bentuk PDB orde dua:

$$m\frac{d^2x}{dt^2} = F(x, v, t)$$

Dengan menyusun gaya F sebagai fungsi dari posisi, kecepatan, dan waktu, kita dapat memodelkan gerak partikel secara lengkap. Misalnya, dalam sistem pegas-massa tanpa redaman, gaya pemulih $F=-kx$ menghasilkan persamaan:

$$\frac{d^2x}{dt^2} + \frac{k}{m}x = 0$$

yang merupakan PDB linier homogen orde dua dan memiliki solusi osilasi harmonik.

Pada teknik elektro, PDB juga berperan penting. Sebagai contoh, analisis rangkaian RLC (Resistor, Induktor, Kapasitor) menghasilkan persamaan diferensial yang menggambarkan tegangan atau arus dalam waktu. Untuk rangkaian seri, hukum Kirchoff menyatakan bahwa jumlah gaya gerak listrik sama dengan jumlah tegangan di tiap komponen, sehingga diperoleh:

$$L\frac{d^2q}{dt^2} + R\frac{dq}{dt} + \frac{q}{C} = E(t)$$

yang merupakan PDB orde dua dengan koefisien konstan, di mana q dalah muatan, dan $E(t)$ adalah tegangan sumber.

Pada teknik mesin, getaran mekanis pada struktur seperti balok atau jembatan juga dimodelkan dengan PDB. Bahkan dalam sistem termal dan fluida, meskipun umumnya menggunakan persamaan diferensial parsial (PDE), pendekatan PDB sering digunakan untuk penyederhanaan model sistem dinamis seperti lumped parameter systems. Keseluruhan ini menunjukkan bahwa PDB adalah dasar dari analisis sistem teknik dan fisika, serta menjadi penghubung antara teori matematis dan implementasi teknologi nyata.

2. Biologi dan Kedokteran

Pada bidang biologi dan kedokteran, Persamaan Diferensial Biasa (PDB) berperan penting dalam memodelkan dinamika sistem biologis yang kompleks dan sering kali tidak dapat diamati secara langsung. Salah satu aplikasi paling umum adalah dalam model pertumbuhan populasi, di mana perubahan jumlah individu dalam suatu populasi dari waktu ke waktu dapat dijelaskan menggunakan PDB. Model Malthus, yang merupakan model pertumbuhan eksponensial paling sederhana, dinyatakan sebagai $\frac{dP}{dt} = rP$ dengan P sebagai populasi dan r sebagai laju pertumbuhan. Namun, model ini tidak realistis untuk jangka panjang karena tidak mempertimbangkan keterbatasan sumber daya. Oleh karena itu, diperkenalkan model logistik oleh Verhulst:

$$\frac{dP}{dt} = rP\left(1 - \frac{P}{K}\right)$$

di mana K adalah kapasitas dukung lingkungan. Model ini banyak digunakan dalam studi ekologi, mikrobiologi, hingga pertumbuhan tumor.

Di bidang kedokteran dan epidemiologi, PDB menjadi dasar dalam pengembangan model penyebaran penyakit. Salah satu model paling terkenal adalah SIR model (*Susceptible-Infected-Recovered*), yang terdiri dari sistem PDB yang menggambarkan interaksi antara populasi yang rentan, terinfeksi, dan sembuh:

$$\begin{aligned}
\frac{dS}{dt} &= -\beta SI, \\
\frac{dI}{dt} &= \beta SI - \gamma I, \\
\frac{dR}{dt} &= \gamma I.
\end{aligned}$$

Model ini sangat penting dalam memprediksi dinamika wabah, merancang intervensi seperti vaksinasi, dan menentukan kebijakan kesehatan masyarakat.

PDB juga digunakan dalam farmakokinetika, untuk memodelkan penyerapan, distribusi, dan eliminasi obat dalam tubuh. Misalnya, perubahan konsentrasi obat dalam plasma darah sering digambarkan dengan model eksponensial sederhana berdasarkan hukum laju pertama. Secara keseluruhan, PDB memberikan kerangka matematis yang sangat kuat untuk menjelaskan, memprediksi, dan mengendalikan fenomena biologis dan medis yang kompleks secara kuantitatif.

3. Ekonomi dan Keuangan

Pada ekonomi dan keuangan, Persamaan Diferensial Biasa (PDB) digunakan secara luas untuk memodelkan dinamika sistem ekonomi yang berkembang terhadap waktu, seperti akumulasi modal, konsumsi, suku bunga, inflasi, hingga harga aset. Salah satu contoh paling terkenal adalah model pertumbuhan Solow, yang menjelaskan bagaimana modal per pekerja berubah seiring waktu:

$$\frac{dk}{dt} = sf(k) - (\delta + n)k$$

di mana K adalah modal per pekerja, sss adalah tingkat tabungan, f(k)f(k)f(k) adalah fungsi produksi, δ\deltaδ adalah tingkat depresiasi modal, dan nnn adalah laju pertumbuhan penduduk. Model ini memberikan wawasan penting tentang bagaimana negara-negara dapat tumbuh secara berkelanjutan dan mengapa ada perbedaan pendapatan antarnegara.

Pada teori konsumsi antar waktu, PDB digunakan untuk menggambarkan bagaimana individu merencanakan konsumsi dan tabungan sepanjang hidupnya berdasarkan preferensi waktu dan suku bunga. Model Ramsey misalnya, menggunakan PDB untuk merumuskan dinamika konsumsi optimal dan kapitalisasi dalam jangka panjang.

Pada keuangan matematika, PDB menjadi dasar dalam penentuan harga opsi dan derivatif. Model Black-Scholes, meskipun merupakan persamaan diferensial parsial, sering kali direduksi ke bentuk PDB untuk derivatif sederhana. Selain itu, perubahan nilai portofolio atau obligasi jangka panjang dapat dimodelkan dengan PDB berbasis suku bunga acuan dan risiko. Lebih lanjut, dalam analisis makroekonomi dinamis seperti *Dynamic Stochastic General Equilibrium* (DSGE), PDB membentuk kerangka utama untuk menggambarkan ekspektasi agen ekonomi dan interaksi antar variabel ekonomi. Keseluruhannya, PDB memberikan alat kuantitatif penting untuk memodelkan dan memprediksi perilaku ekonomi dalam jangka pendek dan panjang secara sistematis.

4. Kimia dan Reaksi Biokimia

Pada kimia dan reaksi biokimia, Persamaan Diferensial Biasa (PDB) merupakan alat penting untuk memodelkan laju perubahan konsentrasi zat kimia dalam suatu reaksi seiring waktu. PDB digunakan dalam kinetika kimia untuk menggambarkan bagaimana konsentrasi reaktan dan produk berubah berdasarkan mekanisme reaksi dan hukum laju. Sebagai contoh, untuk reaksi berorde satu seperti peluruhan zat A:

$$A \to B,$$

dengan laju reaksi v = −k[A], maka perubahan konsentrasi A terhadap waktu dinyatakan dalam bentuk PDB:

$$\frac{d[A]}{dt} = -k[A],$$

yang memiliki solusi eksponensial $[A](t)=[A]_0e^{-kt}$, menggambarkan penurunan konsentrasi secara bertahap. Model ini penting dalam studi reaksi kimia sederhana, termasuk peluruhan radioaktif dan reaksi pembakaran.

Pada sistem reaksi berantai atau reaksi simultan, seperti dalam sintesis senyawa kompleks atau degradasi senyawa kimia, dibutuhkan sistem PDB untuk melacak perubahan konsentrasi beberapa spesies secara bersamaan. Contohnya adalah reaksi A→B→C, yang menghasilkan dua PDB terhubung secara simultan.

Lebih kompleks lagi, dalam reaksi enzimatik dan biokimia, PDB digunakan untuk menggambarkan dinamika sistem biologis seperti model Michaelis-Menten, yang menyederhanakan interaksi antara enzim dan substrat:

$$\frac{d[P]}{dt} = \frac{V_{\max}[S]}{K_m + [S]},$$

di mana $[S]$ adalah konsentrasi substrat, $[P]$ produk, Vmax laju maksimum, dan Km konstanta Michaelis. PDB semacam ini banyak diterapkan dalam farmakologi, metabolisme, dan sintesis protein.

5. Lingkungan dan Ekologi

Pada lingkungan dan ekologi, Persamaan Diferensial Biasa (PDB) merupakan alat analitis penting untuk memodelkan dinamika ekosistem, perubahan lingkungan, serta interaksi antara komponen biotik dan abiotik. Salah satu penerapan utama PDB adalah dalam model transportasi dan peluruhan polutan, misalnya untuk menggambarkan konsentrasi zat pencemar dalam air atau udara. Jika suatu sungai menerima limbah dari sumber tertentu, maka perubahan konsentrasi polutan C(t) dapat dimodelkan sebagai:

$$\frac{dC}{dt} = -kC + S(t),$$

di mana k adalah konstanta peluruhan alami dan S(t) adalah laju suplai polutan. Model ini sangat relevan untuk mengkaji efektivitas kebijakan pengendalian pencemaran dan memprediksi dampak lingkungan dalam jangka waktu tertentu.

Pada ekologi populasi, PDB digunakan untuk memodelkan dinamika predator-mangsa, kompetisi antarspesies, dan keseimbangan

ekosistem. Model klasik Lotka-Volterra menggambarkan interaksi dua spesies:

$$\frac{dN}{dt} = rN - aNP,$$
$$\frac{dP}{dt} = bNP - mP,$$

di mana N adalah populasi mangsa, P populasi predator, r laju pertumbuhan mangsa, a laju konsumsi, b efisiensi konversi energi, dan m mortalitas predator. Model ini memberikan wawasan tentang fluktuasi populasi dan titik-titik kestabilan ekosistem.

PDB juga digunakan dalam model perubahan iklim, seperti dalam menghitung penyerapan karbon oleh hutan, akumulasi gas rumah kaca di atmosfer, serta respons termal laut dan daratan. Model ini mendukung penelitian lingkungan jangka panjang dan pengambilan kebijakan berbasis sains. Dengan demikian, PDB menjadi fondasi penting dalam upaya memahami dan mengelola perubahan lingkungan serta menjaga keberlanjutan sumber daya alam melalui pendekatan kuantitatif dan prediktif.

B. Metode Euler dan Runge-Kutta Orde 4

Menurut Zill dan Wright (2017) dalam *Differential Equations with Boundary-Value Problems*, banyak persoalan dalam fisika, teknik, dan ilmu terapan yang dinyatakan dalam bentuk Persamaan Diferensial Biasa (PDB) tidak dapat diselesaikan secara analitik karena kompleksitas bentuknya. Oleh karena itu, pendekatan numerik menjadi penting untuk memperoleh solusi pendekatan. Dua metode numerik paling dikenal dan banyak digunakan adalah Metode Euler dan Runge-Kutta Orde 4 (RK4). Keduanya digunakan untuk menyelesaikan masalah nilai awal (*initial value problems*/IVP), yaitu PDB yang memiliki nilai fungsi diketahui pada titik awal.

Secara umum, masalah nilai awal untuk PDB orde pertama dinyatakan sebagai:

$$\frac{dy}{dx} = f(x, y), \quad y(x_0) = y_0,$$

di mana f(x,y) adalah fungsi yang diketahui, dan y(x) adalah fungsi tak diketahui yang akan diaproksimasi secara numerik.

1. Metode Euler

Pada lingkungan dan ekologi, Persamaan Diferensial Biasa (PDB) berperan krusial dalam membentuk model matematis yang menjelaskan dinamika sistem alam secara kuantitatif dan prediktif. Alam merupakan sistem kompleks yang mengalami perubahan seiring waktu, mulai dari populasi makhluk hidup, penyebaran polutan, hingga perubahan iklim dan semua dinamika ini dapat dirumuskan dalam bentuk PDB untuk memungkinkan analisis sistematis serta proyeksi ke depan. Salah satu contoh paling mendasar adalah dalam model pencemaran lingkungan, khususnya dalam air dan udara. Misalnya, konsentrasi polutan kimia dalam sungai atau danau dapat dimodelkan dengan PDB berbentuk:

$$\frac{dC}{dt} = -kC + S(t),$$

di mana C(t) adalah konsentrasi polutan pada waktu t, k adalah konstanta peluruhan atau degradasi alami, dan S(t) adalah laju input dari sumber pencemar seperti pabrik atau limbah rumah tangga. Dengan model ini, para peneliti dapat memperkirakan berapa lama waktu yang dibutuhkan untuk air kembali ke kualitas normal, serta mengevaluasi skenario intervensi seperti pengurangan sumber pencemar atau pengolahan limbah.

Pada konteks ekologi, PDB digunakan untuk menggambarkan pertumbuhan dan interaksi antar populasi. Model pertumbuhan eksponensial digunakan untuk menjelaskan dinamika populasi tanpa hambatan, tetapi dalam kenyataannya sumber daya terbatas, sehingga model logistik yang memperhitungkan kapasitas dukung lingkungan (K) menjadi lebih realistis:

$$\frac{dN}{dt} = rN\left(1 - \frac{N}{K}\right).$$

Model ini menggambarkan bagaimana populasi tumbuh pesat pada awalnya namun melambat ketika mendekati batas sumber daya lingkungan. Dalam ekosistem yang lebih kompleks, interaksi antara

spesies, seperti predator dan mangsa, dapat dimodelkan menggunakan model Lotka-Volterra:

$$\frac{dN}{dt} = rN - aNP,$$
$$\frac{dP}{dt} = bNP - mP,$$

dengan N sebagai populasi mangsa, P sebagai predator, dan parameter r,a,b,m mewakili laju reproduksi dan interaksi antar spesies. Model ini memungkinkan pemahaman fluktuasi populasi dalam jangka panjang dan penentuan kondisi stabil atau bencana ekologis.

PDB digunakan dalam perubahan iklim dan siklus biogeokimia. Misalnya, penyerapan karbon oleh tumbuhan dan pelepasannya kembali ke atmosfer dapat dimodelkan untuk mengkaji keseimbangan karbon global. PDB juga digunakan untuk mensimulasikan dinamika suhu bumi berdasarkan masukan energi matahari, emisi gas rumah kaca, dan umpan balik albedo permukaan. Selain itu, dalam studi konservasi, model berbasis PDB digunakan untuk mengevaluasi risiko kepunahan spesies langka dan menentukan kebijakan pengelolaan habitat atau perlindungan hutan.

Penerapan PDB dalam lingkungan dan ekologi tidak hanya membantu dalam memahami fenomena kompleks secara teoritis, tetapi juga memberikan alat praktis untuk pengambilan keputusan berbasis data. Model numerik yang dibangun dari PDB dapat dimasukkan ke dalam simulasi komputer untuk memprediksi dampak perubahan iklim, efek deforestasi, atau keberhasilan program restorasi lingkungan. Dengan demikian, PDB merupakan fondasi penting bagi sains lingkungan modern, yang menggabungkan matematika, teknologi, dan kebijakan untuk mendukung pengelolaan alam yang berkelanjutan dan adaptif.

$$\frac{dy}{dx} = x + y, \quad y(0) = 1$$

Gunakan Metode Euler untuk menghitung nilai pendekatan dari y pada x=0,1dan x=0,2 dengan langkah h=0,1.

Jawaban:

Metode Euler memiliki rumus umum:

$$y_{n+1} = y_n + h \cdot f(x_n, y_n)$$

Diketahui:

- $f(x, y) = x + y$
- $y_0 = 1, x_0 = 0, h = 0{,}1$

$$f(x_0, y_0) = 0 + 1 = 1$$

$$y_1 = y_0 + h \cdot f(x_0, y_0) = 1 + 0{,}1 \cdot 1 = 1{,}1$$

Maka,

$$f(x_1, y_1) = 0{,}1 + 1{,}1 = 1{,}2$$

$$y_2 = y_1 + h \cdot f(x_1, y_1) = 1{,}1 + 0{,}1 \cdot 1{,}2 = 1{,}1 + 0{,}12 = 1{,}22$$

Maka,

Hasil Akhir:

- $y(0{,}1) \approx 1{,}1$
- $y(0{,}2) \approx 1{,}22$

2. Metode Runge-Kutta Orde 4 (RK4)

Metode Runge-Kutta Orde 4 (RK4) adalah salah satu metode numerik paling populer dan andal dalam menyelesaikan Persamaan Diferensial Biasa (PDB), khususnya pada masalah nilai awal. Metode ini merupakan bagian dari keluarga Runge-Kutta yang dikembangkan oleh matematikawan Jerman, Carl Runge dan Martin Wilhelm Kutta, pada awal abad ke-20. Dibandingkan dengan metode numerik dasar seperti metode Euler, RK4 menawarkan akurasi jauh lebih tinggi tanpa menambah kerumitan algoritma secara signifikan, sehingga sangat cocok digunakan dalam pemrograman komputasi sains dan teknik.

Secara prinsip, RK4 bekerja dengan menghitung estimasi rata-rata kemiringan fungsi f(x,y) di sekitar titik x_n, lalu menggunakannya untuk memperkirakan nilai y_{n+1} di titik $x_{n+1}=x_{n+h}$, dengan h sebagai panjang langkah. Dalam setiap iterasi, RK4 menghitung empat nilai gradien (kemiringan):

$$y_{n+1} = y_n + \frac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4),$$

yang merupakan rata-rata tertimbang dari keempat kemiringan tersebut. Strategi ini memberikan galat lokal orde lima dan galat global orde empat, yang berarti tingkat kesalahan menurun secara signifikan dengan penambahan jumlah langkah yang lebih halus (nilai h lebih kecil).

Kelebihan RK4 terletak pada kombinasi antara akurasi dan efisiensi. Dalam praktiknya, RK4 sangat stabil dan mampu menangani berbagai jenis PDB termasuk yang non-linear, tanpa memerlukan penurunan turunan tingkat lebih tinggi atau penyesuaian khusus. Oleh karena itu, metode ini banyak diterapkan dalam berbagai bidang: simulasi gerak partikel dalam fisika, dinamika populasi dalam ekologi, perhitungan orbit dalam astronomi, serta model ekonomi dan keuangan.

Metode ini juga memiliki batasan. Karena sifatnya eksplisit, RK4 tidak cocok untuk PDB yang stiff, yaitu sistem yang memiliki laju perubahan sangat berbeda dalam satu sistem persamaan, di mana metode implisit seperti Backward Euler atau metode Gear lebih disarankan. Selain itu, meskipun RK4 cukup akurat, ia memerlukan empat evaluasi fungsi per langkah, sehingga bisa memakan waktu komputasi lebih lama dibanding metode eksplisit orde rendah dalam sistem berskala besar.

C. Sistem PDB dan Solusi Numerik

Menurut Zill dan Wright (2017) dalam buku *Differential Equations with Boundary-Value Problems,* Persamaan Diferensial Biasa (PDB) adalah persamaan yang menghubungkan suatu fungsi dengan satu variabel bebas dan turunannya. Dalam banyak kasus nyata seperti dinamika sistem fisika, interaksi biologi populasi, atau ekonomi makro masalah yang muncul tidak hanya terdiri dari satu PDB, melainkan beberapa persamaan yang saling berkaitan, dikenal sebagai sistem PDB (*system of ordinary differential equations*). Sistem ini sangat penting karena hampir semua sistem dinamis kompleks di dunia nyata melibatkan beberapa variabel yang berubah secara simultan dan saling memengaruhi. Sistem PDB adalah himpunan dua atau lebih PDB yang memiliki keterkaitan satu sama lain dan harus diselesaikan secara bersamaan. Secara umum, sistem PDB orde pertama dapat dituliskan sebagai:

$$\frac{dy_1}{dt} = f_1(t, y_1, y_2, \ldots, y_n),$$
$$\frac{dy_2}{dt} = f_2(t, y_1, y_2, \ldots, y_n),$$
$$\vdots$$
$$\frac{dy_n}{dt} = f_n(t, y_1, y_2, \ldots, y_n),$$

dengan kondisi awal $y_i\ (t_0)=y_{i,0}$. Sistem seperti ini banyak ditemukan dalam bidang teknik, epidemiologi, dan astrofisika. Salah satu contoh klasik adalah model SIR dalam epidemiologi, yang menggambarkan dinamika tiga populasi: rentan (S), terinfeksi (I), dan pulih (R), sebagai berikut:

$$\frac{dS}{dt} = -\beta SI,$$
$$\frac{dI}{dt} = \beta SI - \gamma I,$$
$$\frac{dR}{dt} = \gamma I.$$

1. Penyelesaian Analitik dan Keterbatasannya

Penyelesaian analitik dalam konteks Persamaan Diferensial Biasa (PDB) merujuk pada proses memperoleh solusi eksplisit dari suatu persamaan diferensial dalam bentuk fungsi yang memenuhi persamaan tersebut dan kondisi awal yang diberikan. Menurut Zill dan Wright (2017), penyelesaian analitik idealnya memberikan representasi eksak dari fungsi tak diketahui, biasanya dalam bentuk kombinasi fungsi aljabar, eksponensial, trigonometri, atau logaritmik. Dalam kasus sederhana seperti PDB linear orde satu, misalnya:

$$\frac{dy}{dx} = ky$$

solusi analitiknya mudah diperoleh:

$$y(x)=Ce^{kx}$$

dengan C sebagai konstanta integrasi yang ditentukan dari kondisi awal. Penyelesaian analitik semacam ini sangat berguna karena memberikan wawasan langsung mengenai perilaku sistem misalnya, apakah sistem

bersifat stabil, apakah solusi akan tumbuh tanpa batas, atau apakah akan konvergen menuju keadaan tunak.

Seiring bertambahnya kompleksitas sistem, keterbatasan pendekatan analitik menjadi sangat nyata. Sebagian besar PDB yang muncul dari model dunia nyata misalnya sistem non-linear, sistem dengan banyak variabel, atau dengan fungsi koefisien yang kompleks tidak dapat diselesaikan secara analitik. Hal ini disebabkan oleh ketidakmampuan metode aljabar konvensional untuk menangani struktur non-linear atau bentuk turunan yang saling terkait secara kompleks. Sebagai contoh, dalam model predasi Lotka-Volterra atau model penyebaran penyakit SIR, meskipun bentuk matematisnya jelas, solusi eksplisit dalam bentuk tertutup (*closed-form solution*) jarang tersedia. Dalam banyak kasus, bahkan jika solusi analitik ada secara teoritis, bentuknya terlalu rumit atau melibatkan fungsi-fungsi khusus (seperti fungsi Bessel atau fungsi gamma) yang tidak praktis digunakan dalam perhitungan teknis atau interpretasi.

Penyelesaian analitik biasanya hanya berlaku dalam domain terbatas dan sangat sensitif terhadap kondisi awal. Artinya, sedikit perubahan pada parameter atau kondisi awal dapat mengubah bentuk solusi secara signifikan. Ini menjadi kendala besar ketika menangani sistem parameterisasi atau simulasi skenario dalam aplikasi dunia nyata seperti dinamika fluida, ekosistem kompleks, atau model ekonomi dinamis, yang sering kali memerlukan evaluasi berulang dengan variasi parameter.

Keterbatasan lain dari pendekatan analitik adalah ketidakfleksibelannya dalam mengakomodasi data aktual atau input tidak kontinu. Dalam kenyataannya, banyak sistem bekerja dengan data pengamatan atau sinyal tak kontinu yang tidak dapat dicocokkan secara langsung dengan fungsi analitik. Oleh karena itu, penggunaan metode numerik menjadi pendekatan dominan dalam praktik modern, karena mampu mengakomodasi struktur sistem yang kompleks, kondisi batas arbitrer, dan ketidakteraturan data yang khas dalam pemodelan dunia nyata.

Dengan demikian, meskipun penyelesaian analitik memiliki keunggulan dalam hal ketepatan dan kejelasan matematis, keterbatasannya dalam fleksibilitas, skalabilitas, dan penerapan praktis menjadikannya kurang memadai untuk banyak aplikasi modern. Di sinilah pendekatan numerik, seperti metode Euler atau Runge-Kutta,

mengambil peran penting dalam menghasilkan solusi pendekatan yang cukup akurat dan dapat diimplementasikan secara luas melalui perangkat lunak dan simulasi komputer.

2. Solusi Numerik untuk Sistem PDB

Solusi numerik untuk sistem Persamaan Diferensial Biasa (PDB) merupakan pendekatan komputasional yang sangat penting dalam menyelesaikan persoalan dinamis yang tidak dapat ditangani secara analitik. Sistem PDB terdiri dari dua atau lebih persamaan diferensial yang saling terkait, menggambarkan interaksi antar variabel yang berubah terhadap satu variabel bebas, biasanya waktu. Dalam dunia nyata, sistem seperti ini banyak ditemukan, misalnya dalam model penyebaran penyakit (model SIR), dinamika populasi (model Lotka-Volterra), interaksi kimia multikomponen, sistem mekanis multibenda, dan pemodelan lingkungan. Karena sebagian besar sistem ini bersifat non-linear dan tidak memiliki solusi eksplisit, maka pendekatan numerik menjadi metode yang paling umum dan efektif.

Menurut Burden dan Faires (2015), metode numerik bekerja dengan cara mendiskretisasi domain waktu menjadi langkah-langkah kecil, lalu memperkirakan nilai variabel di setiap langkah berdasarkan informasi pada langkah sebelumnya. Untuk sistem PDB, metode seperti Euler dan Runge-Kutta Orde 4 (RK4) dapat diperluas secara langsung. Dalam metode Euler, misalnya, setiap persamaan dalam sistem diupdate secara simultan pada setiap langkah waktu menggunakan formula $y_{n+1}=y_n+hf(x_n,y_n)$. Meskipun metode ini sederhana, akurasinya rendah dan rentan terhadap instabilitas, terutama untuk sistem yang kompleks atau "stiff". Sebagai alternatif, RK4 menawarkan peningkatan akurasi yang signifikan dengan menghitung rata-rata gradien dari beberapa titik evaluasi di dalam interval waktu yang sama. Untuk sistem PDB, RK4 menghitung empat vektor gradien untuk semua komponen sistem, lalu menggabungkannya menjadi solusi pendekatan di langkah berikutnya. Metode ini sangat populer karena memberikan keseimbangan antara presisi dan efisiensi komputasi.

Solusi numerik sistem PDB juga sangat bergantung pada pemilihan ukuran langkah (*step size*). Langkah yang terlalu besar dapat menghasilkan error yang besar dan solusi tidak stabil, sedangkan langkah yang terlalu kecil memperlambat komputasi dan meningkatkan

kebutuhan memori. Oleh karena itu, dalam praktiknya sering digunakan metode adaptif seperti Runge-Kutta-Fehlberg atau solver otomatis seperti ode45 di MATLAB dan solve_ivp di Python, yang dapat menyesuaikan ukuran langkah secara otomatis untuk menjaga kestabilan dan akurasi.

Pada sistem yang sangat kompleks atau stiff, metode eksplisit seperti RK4 tidak lagi cukup. Sebagai solusinya, digunakan metode implisit seperti Backward Euler atau metode BDF (*Backward Differentiation Formula*) yang memiliki stabilitas numerik lebih baik. Metode ini biasanya memerlukan penyelesaian sistem aljabar non-linear di setiap langkah waktu, tetapi mampu menangani dinamika cepat tanpa menyebabkan osilasi numerik yang tidak diinginkan.

Solusi numerik sistem PDB telah menjadi tulang punggung berbagai aplikasi ilmiah dan rekayasa modern. Ia memungkinkan simulasi jangka panjang, analisis sensitivitas parameter, dan optimisasi proses. Keunggulan metode numerik terletak pada fleksibilitasnya dalam menangani sistem non-linear, batasan waktu arbitrer, dan masukan berbasis data, menjadikannya alat yang sangat esensial dalam pemodelan kuantitatif berbasis komputer.

3. Implementasi Komputasi

Implementasi komputasi dalam penyelesaian Persamaan Diferensial Biasa (PDB) merupakan langkah krusial dalam menerapkan metode numerik secara praktis untuk berbagai kebutuhan pemodelan ilmiah dan rekayasa. Karena sebagian besar PDB tidak memiliki solusi analitik atau memiliki bentuk solusi yang terlalu kompleks untuk dievaluasi secara langsung, pendekatan numerik berbasis komputasi menjadi solusi utama untuk memperoleh estimasi solusi secara efisien dan akurat. Proses ini melibatkan penerjemahan metode numerik seperti Euler, Runge-Kutta, atau metode implisit ke dalam bentuk algoritma yang dapat dijalankan oleh komputer, serta mengoptimalkan kecepatan dan stabilitas perhitungan dalam berbagai platform pemrograman.

Bahasa pemrograman seperti Python, MATLAB, R, Julia, dan C++ merupakan alat utama dalam implementasi komputasi. Python, misalnya, sangat populer karena memiliki pustaka numerik yang kaya seperti NumPy, SciPy, dan Matplotlib, yang memudahkan proses integrasi numerik, manipulasi data, dan visualisasi hasil. Fungsi odeint dari pustaka scipy.integrate digunakan untuk menyelesaikan sistem PDB

berbasis metode LSODA, yang secara otomatis memilih antara metode stiff dan non-stiff. Untuk penggunaan lanjutan, solve_ivp menawarkan kontrol yang lebih detail terhadap metode integrasi (misalnya RK45, RK23, BDF), toleransi error, dan pencatatan hasil.

Contoh implementasi sederhana dari model SIR dalam Python menunjukkan bagaimana PDB diubah menjadi fungsi Python, kemudian diselesaikan menggunakan odeint dalam beberapa baris kode. Demikian pula, MATLAB menyediakan fungsi ode45, ode23, dan ode15s untuk berbagai jenis sistem, dengan dokumentasi luas dan visualisasi terintegrasi. Keunggulan MATLAB terletak pada antarmuka numerik yang stabil dan kuat, serta kemudahan dalam menyusun model simulasi dinamis melalui Simulink untuk sistem kontrol atau mekanika.

Implementasi komputasi juga mencakup visualisasi hasil, validasi solusi, dan efisiensi pemrosesan. Visualisasi hasil, seperti plotting grafik $y(t)$ terhadap waktu, sangat membantu dalam memahami perilaku sistem dinamis, mendeteksi stabilitas, osilasi, atau kondisi tunak. Validasi hasil dapat dilakukan dengan membandingkan solusi numerik terhadap solusi analitik (jika tersedia), atau menggunakan pengujian konsistensi model dan sensitivitas terhadap perubahan parameter. Sementara itu, efisiensi pemrosesan dapat ditingkatkan melalui optimasi kode, pemilihan metode integrasi adaptif, atau penggunaan paralelisasi untuk sistem berdimensi tinggi.

Implementasi komputasi juga sangat penting dalam konteks simulasi jangka panjang dan sistem real-time, seperti pemodelan epidemiologi untuk kebijakan kesehatan, sistem kontrol otomatis pada robotika, atau dinamika struktural dalam teknik sipil. Kemampuan untuk mengintegrasikan solusi PDB dengan antarmuka pengguna, database, dan sistem pemantauan menjadikan komputasi numerik tidak hanya sebagai alat teoritis, tetapi juga bagian integral dari pengambilan keputusan berbasis sains dan teknologi. Dengan demikian, penguasaan implementasi komputasi menjadi keterampilan kunci dalam era pemodelan numerik modern.

```python
from scipy.integrate import odeint
import numpy as np
import matplotlib.pyplot as plt

def sir_model(y, t, beta, gamma):
    S, I, R = y
    dSdt = -beta * S * I
    dIdt = beta * S * I - gamma * I
    dRdt = gamma * I
    return [dSdt, dIdt, dRdt]

# Nilai awal
S0, I0, R0 = 0.99, 0.01, 0.0
y0 = [S0, I0, R0]
t = np.linspace(0, 160, 1000)
beta = 0.3
gamma = 0.1

sol = odeint(sir_model, y0, t, args=(beta, gamma))

plt.plot(t, sol[:, 0], label='S')
plt.plot(t, sol[:, 1], label='I')
plt.plot(t, sol[:, 2], label='R')
plt.legend()
plt.xlabel('Time')
plt.ylabel('Fraction of Population')
plt.title('Model SIR')
plt.grid()
plt.show()
```

D. Simulasi Dinamis dalam Sistem Teknik dan Biologi

Menurut Ogata (2010) dalam *Modern Control Engineering*, simulasi dinamis adalah proses untuk merepresentasikan perilaku sistem fisik dalam bentuk model matematika yang disimulasikan terhadap waktu menggunakan komputer. Dalam konteks ini, sistem dinamis berarti sistem yang perilakunya berubah terhadap waktu dan dipengaruhi oleh kondisi awal serta input tertentu. Baik di bidang teknik maupun biologi, simulasi dinamis berbasis persamaan diferensial biasa (PDB) menjadi pendekatan utama untuk memahami dan memprediksi perilaku sistem kompleks yang tidak bisa dianalisis secara statis atau linear.

1. Simulasi Dinamis dalam Sistem Teknik

Simulasi dinamis dalam sistem teknik merupakan pendekatan komputasi yang digunakan untuk memodelkan dan menganalisis perilaku sistem teknik yang berubah terhadap waktu. Menurut Ogata (2010) dalam *Modern Control Engineering*, simulasi dinamis memungkinkan insinyur untuk merepresentasikan sistem fisis seperti mekanika, elektrikal, termal, dan sistem kendali dalam bentuk persamaan diferensial biasa (PDB) yang kemudian diselesaikan secara numerik menggunakan perangkat lunak komputasi. Tujuan utama dari simulasi ini adalah untuk memahami respons sistem terhadap masukan, mengevaluasi stabilitas, efisiensi, dan kinerja, serta menguji desain sebelum direalisasikan dalam bentuk fisik. Dalam era teknik modern, simulasi dinamis telah menjadi bagian integral dari proses perancangan dan pengujian sistem teknik di berbagai sektor industri.

Pada rekayasa mekanik, sistem dinamis muncul dalam bentuk gerakan benda, osilasi, getaran, dan interaksi gaya. Salah satu contoh klasik adalah sistem massa-pegas-redaman, yang dirumuskan sebagai PDB orde dua:

$$m\frac{d^2x}{dt^2} + c\frac{dx}{dt} + kx = F(t),$$

dengan m sebagai massa, C koefisien redaman, K konstanta pegas, dan F(t) sebagai gaya luar. Simulasi dinamis memungkinkan insinyur untuk mengevaluasi bagaimana sistem merespon terhadap impuls, osilasi, atau gangguan. Dalam analisis struktur dan kendaraan, simulasi ini digunakan untuk menilai ketahanan terhadap getaran, prediksi resonansi, dan pengujian sistem suspensi. Dengan pemodelan yang akurat, pengujian fisik yang mahal dapat diminimalkan, serta peningkatan desain dapat dilakukan lebih efisien.

Di bidang teknik elektro dan kontrol, simulasi dinamis digunakan untuk menganalisis sistem listrik seperti rangkaian RLC, motor listrik, dan sistem kendali tertutup. Misalnya, rangkaian RLC seri dijelaskan oleh PDB:

$$L\frac{d^2q}{dt^2} + R\frac{dq}{dt} + \frac{q}{C} = E(t),$$

di mana q adalah muatan, dan E(t) tegangan masukan. Dalam konteks sistem kendali, persamaan tersebut digabungkan dengan elemen kendali

seperti pengendali PID (*Proportional-Integral-Derivative*) untuk mengatur keluaran agar mengikuti masukan referensi. Simulasi digunakan untuk menganalisis respon transien (waktu naik, waktu turun, overshoot), respon mantap, serta kestabilan sistem. Software seperti MATLAB/Simulink sangat populer di kalangan insinyur kontrol karena menyediakan lingkungan visual dan numerik untuk memodelkan sistem dinamis, melakukan tuning parameter, serta melakukan simulasi *real-time* dan *hardware-in-the-loop* (HIL).

Simulasi dinamis juga penting dalam sistem termal dan energi, termasuk analisis perpindahan panas, efisiensi sistem pendinginan, dan desain pembangkit listrik. Contohnya, pendinginan sistem elektronik dapat dimodelkan dengan PDB:

$$\frac{dT}{dt} = -k(T - T_{\text{lingkungan}}) + Q(t),$$

yang menggambarkan perubahan suhu terhadap waktu berdasarkan perpindahan panas ke lingkungan dan sumber panas internal. Dengan simulasi, insinyur dapat mengevaluasi bagaimana suhu sistem bereaksi terhadap perubahan beban, ventilasi, atau desain heatsink. Dalam konteks yang lebih besar, simulasi termal digunakan dalam desain sistem HVAC (*Heating, Ventilation, and Air Conditioning*) untuk bangunan hemat energi dan efisien secara lingkungan.

Pada bidang robotika dan mekatronika, simulasi dinamis sangat penting untuk mengembangkan kontrol gerak robot, lengan manipulator, atau kendaraan otomatis. Sistem ini umumnya memiliki banyak derajat kebebasan dan dinamika non-linier yang kompleks. PDB yang mewakili sistem robot sering kali mencakup interaksi gaya, torsi, percepatan, dan kontrol feedback. Dengan simulasi dinamis, desainer dapat mengevaluasi jalur lintasan, konsumsi energi, dan respons kontrol terhadap perubahan lingkungan, bahkan sebelum perangkat keras robot dibangun. Hal ini mempercepat iterasi desain dan mengurangi kesalahan saat implementasi fisik.

Simulasi dinamis dalam teknik sipil digunakan untuk menganalisis respons struktur terhadap beban dinamis seperti gempa bumi, angin, atau kendaraan yang melintas. Model struktur bangunan atau jembatan dapat direpresentasikan sebagai sistem massa-terdistribusi dan diredam, yang kemudian dianalisis menggunakan metode numerik

berbasis PDB. Simulasi ini sangat penting dalam desain bangunan tahan gempa dan infrastruktur yang aman terhadap gangguan lingkungan.

Secara implementatif, simulasi dinamis dilakukan melalui perangkat lunak seperti MATLAB/Simulink, ANSYS, OpenModelica, atau platform pemrograman seperti Python yang menggunakan pustaka scipy.integrate untuk menyelesaikan sistem PDB. Pendekatan numerik seperti Runge-Kutta orde 4 (RK4) atau solver adaptif seperti ode45 dan solve_ivp digunakan untuk menyelesaikan sistem secara akurat dengan kontrol terhadap galat numerik dan kestabilan solusi. Visualisasi hasil simulasi berupa grafik waktu terhadap posisi, kecepatan, suhu, atau tegangan memungkinkan insinyur mengevaluasi performa sistem dan melakukan optimasi desain.

Dengan demikian, simulasi dinamis dalam sistem teknik tidak hanya memperkuat pemahaman teoritis tentang perilaku sistem, tetapi juga menjadi alat praktis untuk eksperimen virtual, pengujian desain, dan validasi sistem. Ia menggabungkan teori matematika, algoritma numerik, dan implementasi komputasi dalam satu kerangka kerja yang sangat penting bagi perkembangan rekayasa modern. Seiring dengan berkembangnya teknologi komputasi dan sensor cerdas, simulasi dinamis akan semakin terintegrasi dalam proses rekayasa canggih seperti sistem kendali adaptif, perancangan berbasis model (*model-based design*), dan *digital twin*.

2. Simulasi Dinamis dalam Sistem Biologi

Simulasi dinamis dalam sistem biologi merupakan pendekatan matematis dan komputasional yang digunakan untuk memahami dan memprediksi perilaku sistem biologis yang kompleks, yang berubah seiring waktu. Sistem-sistem ini mencakup berbagai skala, mulai dari dinamika molekuler di dalam sel hingga interaksi antarpopulasi dalam ekosistem. Menurut Murray (2002), simulasi dinamis pada dasarnya dibangun di atas persamaan diferensial biasa (PDB), yang digunakan untuk menggambarkan laju perubahan variabel-variabel biologis seperti populasi, konsentrasi molekul, atau penyebaran penyakit terhadap waktu. Dengan simulasi ini, para ilmuwan dapat melakukan eksperimen virtual yang mendekati realitas biologis, menguji hipotesis, dan merancang intervensi medis atau ekologis tanpa harus langsung melakukan uji laboratorium yang mahal dan rumit.

Salah satu aplikasi paling awal dan luas dari simulasi dinamis dalam biologi adalah dalam model populasi dan ekologi. Model pertumbuhan eksponensial dan logistik, misalnya, menggambarkan bagaimana populasi makhluk hidup bertambah dengan mempertimbangkan sumber daya lingkungan. Model logistik, yang menggunakan PDB $\frac{dN}{dt} = rN\left(1 - \frac{N}{K}\right)$ mampu menangkap fenomena batas kapasitas lingkungan (*carrying capacity*) yang menjadi pembatas alami dalam pertumbuhan populasi. Model ini diperluas dalam bentuk model Lotka-Volterra, yang mensimulasikan interaksi predator-mangsa. Dalam model ini, dua PDB saling terkait digunakan untuk merepresentasikan perubahan jumlah populasi mangsa dan predator, menghasilkan dinamika fluktuatif yang menyerupai pola-pola yang diamati di alam.

Simulasi dinamis juga sangat penting dalam bidang epidemiologi, yaitu studi tentang penyebaran penyakit menular. Model klasik yang digunakan adalah model SIR (*Susceptible-Infected-Recovered*) yang menggunakan sistem PDB:

$$\frac{dS}{dt} = -\beta SI,$$
$$\frac{dI}{dt} = \beta SI - \gamma I,$$
$$\frac{dR}{dt} = \gamma I,$$

di mana S, I, dan R masing-masing mewakili jumlah individu yang rentan, terinfeksi, dan sembuh. Parameter β adalah tingkat penularan, dan γ adalah tingkat pemulihan. Dengan simulasi numerik terhadap sistem ini, para ahli kesehatan dapat memperkirakan kapan puncak wabah akan terjadi, berapa jumlah maksimum kasus, serta mengevaluasi dampak strategi intervensi seperti vaksinasi, karantina, atau pembatasan sosial. Selama pandemi COVID-19, model seperti ini menjadi dasar berbagai simulasi skenario yang membantu pengambilan kebijakan di seluruh dunia.

Simulasi dinamis juga digunakan dalam farmakokinetika dan farmakodinamika, yaitu studi tentang bagaimana obat bekerja di dalam tubuh dan bagaimana tubuh mempengaruhi obat. Misalnya, dalam model satu kompartemen, konsentrasi obat dalam darah sering digambarkan dengan persamaan

$$\frac{dC}{dt} = -kC$$

di mana C adalah konsentrasi dan k adalah laju eliminasi. Simulasi model ini membantu menentukan dosis optimal, durasi pemberian obat, serta mengevaluasi efek samping yang mungkin terjadi akibat akumulasi obat di dalam tubuh. Model ini dapat diperluas menjadi model multi-kompartemen yang mempertimbangkan jaringan dan organ berbeda, serta interaksi kompleks antara metabolisme dan ekskresi.

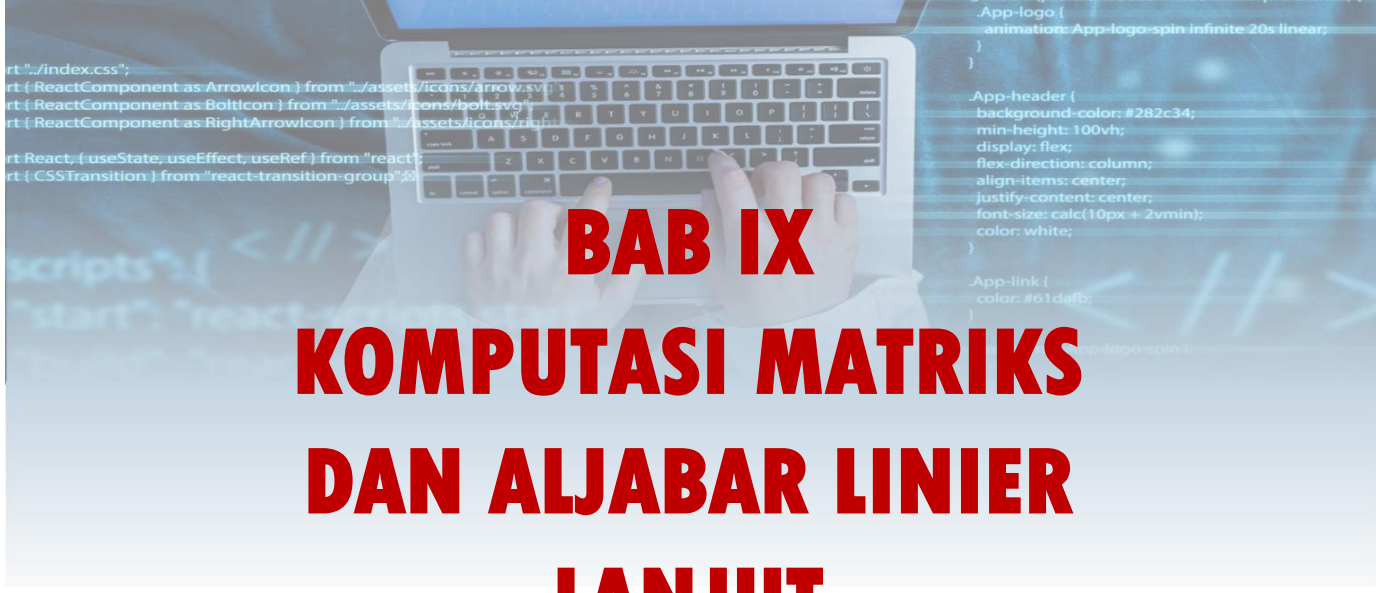
Pada skala molekuler dan seluler, simulasi dinamis menjadi alat penting dalam biologi sistem, yaitu studi tentang jaringan interaksi gen, protein, dan metabolit. Model regulasi genetik, misalnya, dapat menggunakan PDB untuk menggambarkan ekspresi dan supresi gen, serta osilasi dalam sistem biologis seperti jam biologis sirkadian. Salah satu contoh adalah model Goodwin yang menggambarkan osilasi konsentrasi protein yang mengatur ritme harian organisme. Di sini, simulasi membantu memprediksi efek dari mutasi genetik, pengaruh obat, dan interaksi sinyal biokimia dalam sel. Simulasi juga mendukung desain terapi berbasis genetik, serta rekayasa jaringan dan sintesis sistem biologis baru (biologi sintetik).

Perangkat lunak yang umum digunakan dalam simulasi dinamis sistem biologi mencakup Python dengan pustaka SciPy dan NumPy, MATLAB, serta perangkat khusus seperti COPASI, CellDesigner, dan BioNetGen. Simulasi dilakukan dengan menyelesaikan sistem PDB menggunakan metode numerik seperti Runge-Kutta Orde 4 (RK4) atau solver adaptif seperti odeint dan solve_ivp. Dalam model dengan banyak variabel dan parameter, teknik seperti analisis sensitivitas dan estimasi parameter digunakan untuk mengevaluasi seberapa kuat model terhadap variasi input, dan untuk menyesuaikan model dengan data eksperimen.

Secara umum, keunggulan utama dari simulasi dinamis dalam sistem biologi adalah kemampuannya untuk menangani kompleksitas sistem hidup, baik dalam skala mikro (seluler) maupun makro (populasi atau ekosistem), yang hampir mustahil dipecahkan secara analitik. Dengan simulasi, para peneliti dapat mengamati konsekuensi dari intervensi yang belum pernah diuji, mengevaluasi ketidakpastian biologis, dan merancang sistem biologis baru berdasarkan prinsip dinamika dan kontrol. Namun demikian, tantangan utama tetap ada,

terutama dalam hal ketersediaan data parameter, validasi eksperimental, serta ketidakpastian biologis yang sulit dimodelkan secara deterministik.

Dengan demikian, simulasi dinamis telah menjadi bagian integral dari biologi modern. Tidak hanya sebagai alat bantu visualisasi dan prediksi, tetapi juga sebagai kerangka konseptual yang memungkinkan integrasi berbagai tingkat informasi biologis dari genetik hingga populasi ke dalam satu sistem yang bisa dianalisis, dimodifikasi, dan diaplikasikan secara nyata dalam riset kesehatan, konservasi, dan bioteknologi. Seiring berkembangnya teknologi komputasi dan integrasi data biologis berbasis omik, simulasi dinamis diperkirakan akan terus berperan sentral dalam inovasi biomedis dan bioinformatika masa depan.



BAB IX
KOMPUTASI MATRIKS DAN ALJABAR LINIER LANJUT

Matriks dan transformasi linier bukan hanya bagian dari teori matematika, melainkan juga alat komputasi yang sangat kuat dalam menyelesaikan berbagai persoalan di bidang teknik, fisika, *data science,* dan pemodelan numerik. Dalam bab ini, pembaca akan diperkenalkan pada topik-topik lanjutan seperti dekomposisi matriks (LU, QR, dan SVD), *eigenvalue-eigenvector*, serta sistem persamaan linier berskala besar yang menuntut pendekatan algoritmik efisien. Penekanan diberikan pada bagaimana teori aljabar linier dapat diimplementasikan secara numerik melalui pemrograman, serta bagaimana kestabilan numerik dan efisiensi algoritma menjadi pertimbangan utama dalam aplikasi dunia nyata. Pendekatan yang digunakan dalam bab ini bersifat praktis namun tetap memperhatikan landasan teoritis, sehingga pembaca tidak hanya mampu memahami konsep, tetapi juga menguasai cara penerapannya secara langsung.

A. *Eigenvalue* dan *Eigenvector*

Eigenvalue dan *eigenvector* adalah konsep fundamental dalam aljabar linier yang memiliki peranan penting dalam banyak bidang sains dan teknik, termasuk fisika, rekayasa, ilmu komputer, pembelajaran mesin, serta pemrosesan citra dan suara. Konsep ini memungkinkan kita memahami bagaimana transformasi linier mempengaruhi ruang vektor dan bagaimana sistem dapat direduksi atau disederhanakan menjadi

bentuk yang lebih terstruktur untuk analisis atau komputasi. Saat merujuk pada Lay, D.C. (2012) *Linear Algebra and Its Applications*, Pearson dijelaskan bahwa *eigenvalue* (nilai eigen) dan *eigenvector* (vektor eigen) adalah solusi dari transformasi linier berbasis matriks. Secara formal, diberikan sebuah matriks persegi

$$A \in R^{n \times n},$$

vektor tak nol

$$v \in R^n$$

dan skalar

$\lambda \in R$

maka v disebut sebagai *eigenvector* dari A, dan λ adalah *eigenvalue* yang sesuai jika memenuhi:

$$A\mathbf{v} = \lambda\mathbf{v}$$

Artinya, jika suatu vektor dikenai transformasi oleh matriks A, hasilnya tetap searah dengan vektor semula, hanya mengalami perubahan skala oleh faktor λ.

1. Interpretasi Geometris

Interpretasi geometris dari *eigenvalue* dan *eigenvector* merupakan fondasi visual yang kuat dalam memahami bagaimana suatu transformasi linier bekerja terhadap ruang vektor. Jika kita membayangkan sebuah matriks A sebagai suatu transformasi linier dalam ruang dua atau tiga dimensi, maka *eigenvector* dapat dipahami sebagai arah tertentu dalam ruang tersebut yang tetap tidak berubah arah ketika dikenai transformasi oleh A; hanya panjangnya yang berubah, diperbesar, diperkecil, atau bahkan dibalik arah tergantung pada nilai *eigenvalue*-nya. Pandangan ini sangat penting dalam memahami struktur sistem dinamis, deformasi spasial, dan perilaku asimtotik dari banyak sistem matematika dan fisik.

Bayangkan bidang dua dimensi R^2, dan vektor v adalah suatu panah yang menunjuk ke suatu arah tertentu dari titik asal. Ketika kita menerapkan transformasi linier dengan matriks A terhadap vektor ini, hasilnya adalah vektor baru Av. Untuk vektor biasa, arah dari Av akan berubah tergantung pada bagaimana matriks A bertindak terhadap komponen-komponen x dan y dari vektor tersebut. Namun, jika v adalah

sebuah *eigenvector* dari A, maka arah Av tetap searah atau berlawanan arah dengan v. Perubahan yang terjadi hanyalah skala panjangnya, yang diatur oleh *eigenvalue* λ sehingga:

$$Av = \lambda v$$

Ini berarti transformasi oleh A "memanjangkan", "memendekkan", atau "membalik" vektor tersebut tanpa mengubah orientasinya di dalam ruang. Sebagai contoh konkret, pertimbangkan matriks dua dimensi sederhana seperti:

$$A = \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix}$$

Matriks ini adalah transformasi skala (stretching) terhadap sumbu x sebesar faktor 3 dan sumbu y sebesar faktor 2. Dalam hal ini, vektor $v_1 = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$ dan $v_2 = \begin{bmatrix} 0 \\ 1 \end{bmatrix}$ adalah *eigenvector* dari A karena ketika dikalikan oleh A, masing-masing hanya mengalami perubahan panjang:

$$A\mathbf{v}_1 = \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix} \begin{bmatrix} 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 3 \\ 0 \end{bmatrix} = 3\mathbf{v}_1$$

$$A\mathbf{v}_2 = 2\mathbf{v}_2$$

Dari sini, kita dapat memahami bahwa kedua arah x dan y adalah arah "khusus" yang tidak berubah arah ketika dikenai transformasi oleh A. Inilah yang dimaksud dengan arah tetap dari transformasi linier. Jika kita mengambil vektor sembarang yang bukan kombinasi linear dari vektor eigen ini, maka hasil transformasi tidak akan searah dengan vektor semula, arahnya akan berubah karena komponennya mengalami transformasi yang berbeda di setiap sumbu.

Interpretasi ini menjadi semakin menarik ketika kita berhadapan dengan transformasi rotasi, refleksi, atau shearing (geseran). Misalnya, dalam kasus rotasi murni pada bidang dua dimensi, tidak ada vektor (selain nol) yang tetap pada arah yang sama setelah transformasi; oleh karena itu, tidak ada eigenvector nyata untuk rotasi murni dengan sudut bukan kelipatan 180°. Sebaliknya, dalam refleksi terhadap garis tertentu, maka garis refleksi itu sendiri adalah arah *eigenvector* dengan *eigenvalue* 1, dan garis tegaknya adalah arah *eigenvector* dengan *eigenvalue* -1, karena arah tegak lurus tersebut dibalik oleh transformasi.

Interpretasi geometris ini juga sangat berguna dalam memahami sistem dinamik. Dalam sistem dinamis linier, misalnya $\frac{dx}{dt} = Ax$, arah vektor eigen menggambarkan arah mode pertumbuhan atau peluruhan sistem. *Eigenvalue* positif menunjukkan arah di mana sistem tumbuh secara eksponensial seiring waktu, sedangkan *eigenvalue* negatif menunjukkan arah peluruhan. *Eigenvalue* kompleks dengan bagian imajiner menggambarkan rotasi atau osilasi dalam sistem, dengan bagian real menentukan apakah amplitudo osilasi meningkat, menurun, atau tetap.

Pada tiga dimensi (R^3), interpretasi serupa berlaku. *Eigenvector* direpresentasikan sebagai arah tertentu dalam ruang tiga dimensi yang tidak berubah arah setelah dikenai transformasi. *Eigenvalue* menentukan perubahan panjang sepanjang arah tersebut. Visualisasi dalam ruang tiga dimensi umumnya lebih sulit, namun secara konseptual sama vektor tetap pada garis yang sama dari asal, hanya jaraknya dari titik asal yang berubah.

Interpretasi geometris ini juga mendasari teknik komputasional seperti *Principal Component Analysis* (PCA). Dalam PCA, kita mencari arah (komponen utama) dalam data multidimensi di mana varians (penyebaran data) paling besar. Arah ini merupakan *eigenvector* dari matriks kovarian data, dan besarnya penyebaran di sepanjang arah tersebut diwakili oleh *eigenvalue*-nya. Dengan kata lain, PCA merepresentasikan data ke dalam sumbu-sumbu baru (basis baru) yang ditentukan oleh arah geometri intrinsik dari distribusi data itu sendiri sebuah aplikasi langsung dari interpretasi geometris *eigenvalue* dan *eigenvector*.

Dengan pemahaman geometris ini, kita dapat lebih intuitif mengenali dan menjelaskan bagaimana sistem bekerja dan berubah dalam ruang vektor. Daripada hanya melihat *eigenvalue* dan *eigenvector* sebagai hasil aljabar, interpretasi ini membawa kita lebih dekat pada pemahaman fungsional dalam dunia nyata mulai dari arah getaran dalam struktur teknik, rotasi dalam grafik komputer, hingga pola dominan dalam data statistik.

2. Cara Menentukan *Eigenvalue* dan *Eigenvector*

Menentukan *eigenvalue* dan *eigenvector* dari sebuah matriks merupakan salah satu proses inti dalam aljabar linier yang banyak

diterapkan dalam komputasi ilmiah, teknik, statistik, dan berbagai cabang matematika terapan. Proses ini melibatkan dua langkah utama: pertama, menemukan nilai-nilai eigen (*eigenvalue*) dari matriks; dan kedua, mencari vektor-vektor yang sesuai (*eigenvector*) untuk masing-masing nilai tersebut. Meskipun secara konseptual sederhana, perhitungannya bisa menjadi kompleks, tergantung pada ukuran dan sifat matriks yang dianalisis.

Langkah pertama dalam menentukan eigenvalue dimulai dengan menyusun persamaan karakteristik dari matriks tersebut. Misalkan diberikan sebuah matriks kuadrat A berukuran n×n, maka kita mencari nilai skalar λ dan vektor tak nol v yang memenuhi hubungan:

$$A\mathbf{v} = \lambda\mathbf{v}$$

Dengan menyusun ulang bentuk tersebut, kita mendapatkan:

$$(A - \lambda I)\mathbf{v} = \mathbf{0}$$

di mana I adalah matriks identitas berukuran n×n. Agar persamaan ini memiliki solusi non-trivial (selain solusi vektor nol), maka matriks (A−λI) harus bersifat singular, yaitu memiliki determinan nol. Oleh karena itu, kita harus menyelesaikan persamaan determinan berikut:

$$\det(A - \lambda I) = 0$$

Inilah yang disebut persamaan karakteristik, dan penyelesaian dari persamaan ini memberikan kita nilai-nilai eigen dari matriks A. Persamaan ini umumnya berupa polinomial berderajat n, dan solusinya bisa berupa bilangan real, kompleks, atau bahkan berulang (multiplikitas lebih dari satu). Sebagai contoh, misalkan kita memiliki matriks dua dimensi berikut:

$$A = \begin{bmatrix} 4 & 1 \\ 2 & 3 \end{bmatrix}$$

Langkah pertama adalah menyusun matriks $A - \lambda I$:

$$A - \lambda I = \begin{bmatrix} 4 - \lambda & 1 \\ 2 & 3 - \lambda \end{bmatrix}$$

Kemudian hitung determinannya:

$$\det(A - \lambda I) = (4 - \lambda)(3 - \lambda) - 2 \cdot 1 = \lambda^2 - 7\lambda + 10$$

Menyelesaikan persamaan karakteristik $\lambda^2 - 7\lambda + 10 = 0$ memberikan dua nilai eigen: $\lambda_1 = 5$ dan $\lambda_2 = 2$.

Langkah berikutnya adalah mencari eigenvector untuk masing-masing nilai eigen. Untuk tiap λ, kita substitusikan kembali ke dalam:

$$(A - \lambda I)\mathbf{v} = \mathbf{0}$$

dan menyelesaikan sistem persamaan linier homogen tersebut. Misalnya, untuk $\lambda = 5$, kita susun:

$$A - 5I = \begin{bmatrix} -1 & 1 \\ 2 & -2 \end{bmatrix}$$

Kemudian, kita selesaikan sistem:

$$\begin{bmatrix} -1 & 1 \\ 2 & -2 \end{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

Sistem ini bersifat dependen, dan memiliki solusi tak hingga. Kita bisa memilih $v_1 = 1$ maka $v_2 = q$, sehingga salah satu *eigenvector*-nya adalah $v_1 = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$. Hal yang sama dilakukan untuk $\lambda=2$ dan kita dapatkan eigenvector lainnya, misalnya $v_2 = \begin{bmatrix} 1 \\ 2 \end{bmatrix}$. Secara umum, sistem $(A-\lambda I)v=0$ adalah sistem linier homogen yang selalu memiliki solusi non-trivial jika dan hanya jika λ adalah nilai eigen dari A. Penyelesaiannya dapat dilakukan menggunakan eliminasi Gauss, substitusi, atau dengan bantuan perangkat lunak matematika seperti MATLAB, Python (NumPy/SciPy), Mathematica, atau R.

Pada kasus ketika nilai eigen memiliki multiplikitas lebih dari satu, kita juga perlu memperhatikan dimensi dari ruang eigennya. Ini berkaitan dengan jumlah vektor eigen linier independen yang dapat dihasilkan untuk satu nilai eigen. Jika jumlah ini sama dengan multiplicity-nya, maka matriks tersebut dapat didiagonalisasi, yaitu direpresentasikan sebagai $A=PDP^{-1}$, di mana D adalah matriks diagonal dari *eigenvalue* dan P adalah matriks yang kolom-kolomnya terdiri dari *eigenvector* yang bersesuaian.

Prosedur di atas menjadi lebih kompleks untuk matriks berdimensi besar atau matriks dengan elemen kompleks. Dalam banyak

kasus praktis, terutama untuk matriks berukuran besar, pendekatan numerik digunakan untuk menghitung nilai eigen secara efisien. Metode populer meliputi *Power Iteration* untuk mencari nilai eigen terbesar, QR Algorithm untuk menemukan semua nilai eigen, dan Jacobi Method untuk matriks simetris.

Cara menentukan *eigenvalue* dan *eigenvector* memerlukan pemahaman aljabar linier yang mendalam, penguasaan terhadap manipulasi matriks, serta keterampilan komputasional dalam menyelesaikan sistem linier. Proses ini bukan sekadar manipulasi simbolik, tetapi berakar pada pemahaman struktur dan dinamika sistem linier, serta penting dalam banyak aplikasi mulai dari stabilitas struktur dalam teknik sipil, analisis data dalam statistik, hingga pembelajaran mesin dan pencitraan digital.

3. Aplikasi *Eigenvalue* dan *Eigenvector*

Eigenvalue dan *eigenvector* memiliki peran penting dalam berbagai bidang sains dan teknik karena kemampuannya dalam menyederhanakan analisis sistem yang kompleks melalui pendekatan struktural. Konsep ini tidak hanya relevan dalam matematika murni, tetapi juga menjadi tulang punggung banyak metode numerik dan teknik komputasi modern. Dalam dunia nyata, banyak fenomena fisika, sistem mekanik, jaringan sosial, pemrosesan sinyal, serta pembelajaran mesin dapat dimodelkan dan diselesaikan lebih efisien dengan memahami struktur eigennya. Berikut ini adalah uraian mendalam mengenai beberapa aplikasi utama dari *eigenvalue* dan *eigenvector* dalam berbagai konteks.

Pada bidang rekayasa struktur dan mekanika, *eigenvalue* digunakan dalam analisis getaran. Ketika sebuah struktur seperti jembatan, gedung pencakar langit, atau pesawat mengalami gangguan atau gaya luar, sistem tersebut akan berosilasi pada frekuensi-frekuensi tertentu yang disebut frekuensi alami (*natural frequencies*). Frekuensi ini adalah akar dari eigenvalue dari sistem matriks massa dan kekakuan (*mass and stiffness matrices*). Misalnya, dalam analisis mode getar suatu bangunan, setiap eigenvalue merepresentasikan kuadrat dari frekuensi alami, dan eigenvectornya menunjukkan bentuk mode (*mode shape*) dari getaran tersebut. Dengan demikian, memahami *eigenstructure* dari sistem mekanik sangat penting untuk desain struktur yang aman terhadap resonansi atau beban dinamis.

Pada fisika kuantum, konsep *eigenvalue* sangat fundamental. Persamaan Schrödinger, yang menggambarkan perilaku sistem kuantum, secara matematis merupakan persamaan eigen. Fungsi gelombang kuantum (*wavefunction*) dari suatu partikel merupakan *eigenvector*, dan energi-energi diskrit yang dapat dimiliki oleh partikel tersebut adalah nilai eigen. Setiap operator fisika seperti momentum, energi, dan spin direpresentasikan sebagai operator linier, dan hasil pengukuran nilai-nilainya adalah nilai eigen dari operator tersebut. Oleh karena itu, seluruh struktur teori kuantum dibangun di atas landasan *eigenvalue-eigenvector.*

Pada analisis data dan pembelajaran mesin, *eigenvalue* dan *eigenvector* menjadi alat utama dalam teknik reduksi dimensi, terutama dalam *Principal Component Analysis* (PCA). PCA adalah metode statistik yang digunakan untuk mengurangi kompleksitas data berdimensi tinggi dengan menemukan sumbu-sumbu utama (*principal components*) dari distribusi data. Sumbu-sumbu ini ditentukan oleh eigenvector dari matriks kovarian data, dan sumbu-sumbu dengan eigenvalue terbesar mewakili arah dengan variasi data paling signifikan. Dengan memilih beberapa komponen utama pertama, kita dapat mengurangi dimensi data tanpa kehilangan informasi penting secara signifikan. Ini sangat berguna dalam pengolahan citra, pengenalan pola, dan kompresi data.

Pada graf teori dan analisis jaringan, *eigenvalue* digunakan dalam banyak aspek, salah satunya adalah algoritma PageRank milik Google. Dalam pendekatan ini, halaman web direpresentasikan sebagai simpul (*nodes*) dalam graf terarah, dan hubungan antar halaman sebagai sisi (*edges*). Matriks transisi dari graf ini digunakan untuk membentuk sistem Markov, dan peringkat halaman ditentukan oleh *eigenvector* dominan dari matriks tersebut. Halaman dengan bobot (komponen) terbesar dalam *eigenvector* tersebut dianggap paling penting. Selain itu, dalam analisis jaringan sosial atau jaringan biologis, spektrum eigen dari matriks ketetanggaan (*adjacency matrix*) atau matriks Laplacian jaringan memberikan informasi penting tentang struktur jaringan, seperti keterhubungan, jumlah komunitas, dan ketahanan terhadap gangguan.

Pada sistem dinamik dan kontrol, terutama sistem diferensial linier seperti $\frac{dx}{dt} = Ax$, *eigenvalue* dari matriks A menentukan perilaku jangka panjang sistem tersebut. Jika semua *eigenvalue* memiliki bagian

real negatif, maka sistem bersifat stabil karena semua solusi cenderung ke nol. Jika ada *eigenvalue* dengan bagian real positif, sistem bersifat tidak stabil. Di sinilah peran penting analisis eigensistem untuk memastikan kestabilan sistem kontrol, baik dalam robotika, pesawat terbang, maupun sistem otomatisasi industri.

Di bidang komputasi citra dan pemrosesan sinyal, transformasi yang melibatkan matriks kovarian atau matriks transformasi sering kali memanfaatkan *eigenvalue* dan *eigenvector*. Misalnya, dalam *face recognition* (pengenalan wajah), metode seperti *Eigenfaces* membangun basis wajah dari kumpulan gambar pelatihan dengan mencari *eigenvector* dari matriks citra. Setiap gambar wajah kemudian dapat direpresentasikan sebagai kombinasi linear dari basis ini, sehingga identifikasi dan klasifikasi wajah menjadi lebih efisien dan akurat.

Pada bidang ekonomi dan ekonometri, *eigenvalue* digunakan dalam analisis input-output antar sektor, serta dalam model stokastik seperti analisis Markov Chain, di mana matriks transisi menyimpan probabilitas perpindahan antara keadaan-keadaan sistem. *Eigenvector* stasioner dari matriks transisi menggambarkan distribusi jangka panjang dari sistem ekonomi tersebut. Di bidang biologi matematika, terutama dalam model populasi, *eigenvalue* membantu menentukan pertumbuhan populasi jangka panjang dan stabilitas ekosistem. Contohnya, dalam model *Leslie matrix* (model pertumbuhan populasi terstruktur menurut usia), nilai eigen terbesar (*dominant eigenvalue*) merepresentasikan tingkat pertumbuhan populasi, sedangkan eigenvectornya memberi distribusi populasi dalam keadaan stabil.

Gambar 5. *Big Data*

Sumber: *Dqlab*

Dari berbagai bidang ini, dapat disimpulkan bahwa *eigenvalue* dan *eigenvector* menyediakan kerangka kerja matematis untuk mengevaluasi dan menyederhanakan sistem yang kompleks. Kekuatan utama dari konsep ini adalah kemampuannya dalam mengubah sistem menjadi bentuk diagonal atau hampir-diagonal, di mana analisis dan perhitungan menjadi jauh lebih sederhana. Dalam era *big data* dan komputasi intensif, pemanfaatan struktur eigen menjadi semakin penting karena efisiensinya dalam menangani persoalan berdimensi besar dan kompleks. Oleh karena itu, penguasaan terhadap konsep dan aplikasi eigenvalue dan eigenvector adalah keterampilan esensial bagi ilmuwan, insinyur, dan analis data modern.

B. Dekomposisi LU, QR, dan SVD

Dekomposisi matriks merupakan teknik fundamental dalam aljabar linier numerik yang digunakan untuk menyederhanakan berbagai perhitungan matematis, seperti penyelesaian sistem persamaan linier, komputasi nilai eigen, dan reduksi dimensi. Tiga metode dekomposisi paling penting dan sering digunakan adalah LU decomposition (*Lower-Upper decomposition*), QR *decomposition,* dan *Singular Value Decomposition* (SVD). Masing-masing memiliki peran dan keunggulan tertentu dalam penerapan praktis dan komputasi numerik.

1. Dekomposisi LU (*Lower-Upper Decomposition*)

Dekomposisi LU (*Lower-Upper Decomposition*) adalah teknik aljabar linier yang memfaktorkan sebuah matriks persegi A menjadi hasil perkalian dua matriks segitiga: matriks segitiga bawah L (*Lower*) dan matriks segitiga atas U (*Upper*), sehingga diperoleh bentuk A=LU. Konsep ini sangat penting dalam komputasi numerik karena menyederhanakan berbagai perhitungan, terutama dalam penyelesaian sistem persamaan linier, invers matriks, dan perhitungan determinan. Dengan mendekomposisi matriks ke dalam bentuk segitiga, kita dapat memanfaatkan sifat-sifat sederhana dari sistem linier segitiga untuk menyelesaikan masalah dengan efisien dan stabil.

Secara umum, dekomposisi LU hanya berlaku untuk matriks persegi n×n, dan tidak semua matriks memiliki dekomposisi LU tanpa modifikasi. Untuk menjamin dekomposisi ini bisa dilakukan secara stabil, sering kali diperlukan pivoting, yaitu pertukaran baris untuk

menghindari pembagian oleh nol atau bilangan sangat kecil. Dalam kasus ini, dekomposisi menjadi PA=LU, di mana P adalah matriks permutasi yang menyatakan posisi baris yang ditukar. Proses dekomposisi dilakukan melalui metode eliminasi Gauss, di mana elemen-elemen di bawah diagonal utama diubah menjadi nol menggunakan operasi baris elementer, dan koefisien yang digunakan untuk operasi tersebut disimpan dalam matriks L.

Dekomposisi LU memiliki keuntungan besar dalam menyelesaikan sistem persamaan linier:

$$Ax=b$$

Setelah matriks A didekomposisi menjadi LU, kita dapat menyelesaikan sistem tersebut dalam dua tahap:

1. Menyelesaikan

   $$Ly=b$$

   menggunakan substitusi maju (*forward substitution*), karena L adalah matriks segitiga bawah;
2. Menyelesaikan

   $$Ux=y$$

   menggunakan substitusi mundur (*back substitution*), karena U adalah matriks segitiga atas. Proses ini jauh lebih efisien dibandingkan langsung menggunakan invers matriks atau eliminasi Gauss berulang untuk setiap vektor b.

Dekomposisi LU juga sangat berguna dalam konteks faktorisasi matriks untuk perhitungan determinan. Jika:

$$A = LU$$

maka determinan $\det(A) = \det(L) \cdot \det(U)$. Karena determinan dari matriks segitiga adalah hasil kali elemen diagonalnya, maka perhitungan determinan menjadi sangat cepat dan stabil.

Pada implementasi komputasi, dekomposisi LU tersedia dalam berbagai bahasa dan pustaka numerik seperti:

- MATLAB ([L,U,P] = lu(A)″
- Python melalui SciPy (*scipy.linalg.lu*)
- Julia

Keunggulannya adalah dapat digunakan secara efisien dalam perhitungan berskala besar, misalnya dalam simulasi struktur teknik, analisis jaringan listrik, atau model-model numerik fisika dan kimia.

2. Dekomposisi QR

Dekomposisi QR adalah salah satu metode faktorisasi matriks yang sangat penting dalam aljabar linier numerik dan memiliki beragam aplikasi dalam penyelesaian sistem *overdetermined* (jumlah persamaan lebih banyak dari variabel), pencarian solusi *least squares*, serta dalam algoritma komputasi nilai eigen. Dalam dekomposisi ini, sebuah matriks $A \in R^{m \times n}$ (dengan $m \geq n$) difaktorkan menjadi hasil perkalian dua matriks, yaitu matriks ortogonal QQQ dan matriks segitiga atas RRR, sehingga diperoleh bentuk:

$$A = QR$$

Matriks $Q \in R^{m \times m}$ memiliki sifat ortogonal, yang berarti kolom-kolomnya adalah vektor ortonormal dan memenuhi $Q^{TQ} = I$, sementara matriks $R \in R^{m \times n}$ adalah matriks segitiga atas, yang menyimpan koefisien kombinasi linier dari kolom-kolom vektor asli pada A. Salah satu aplikasi utama dekomposisi QR adalah dalam penyelesaian masalah *least squares*. Ketika sistem linier $Ax = b$ tidak memiliki solusi eksak karena sistemnya overdetermined, solusi terbaik dalam arti minimum kesalahan kuadrat dapat dicari dengan mengubahnya menjadi sistem normal: $A^TAx = A^Tb$.

Namun, pendekatan ini bisa menghasilkan instabilitas numerik karena meningkatkan kondisi numerik yang buruk. Alternatif yang lebih stabil adalah dengan menggunakan dekomposisi QR. Jika $A = QR$, maka $Ax = QRx = b$, dan dengan mengalikan kedua sisi dengan Q^T, diperoleh sistem sederhana $Rx = Q^Tb$, yang kemudian diselesaikan dengan substitusi mundur karena RRR berbentuk segitiga atas.

Secara praktis, dekomposisi QR dapat dilakukan dengan beberapa metode, antara lain metode Gram-Schmidt, *Householder reflections*, dan *Givens rotations*. Metode Gram-Schmidt menggunakan proses ortonormalisasi vektor dan lebih intuitif secara konsep, tetapi kurang stabil secara numerik. Metode Householder, yang menggunakan refleksi ortogonal, lebih stabil dan sering digunakan dalam perangkat lunak numerik seperti MATLAB dan SciPy. *Givens rotations*, di sisi lain, lebih cocok untuk matriks besar dan jarang (*sparse matrices*) karena memodifikasi dua baris pada satu waktu.

Pada algoritma komputasi nilai eigen, QR *decomposition* menjadi komponen utama dalam algoritma QR *iteration*, yang

digunakan untuk menghitung spektrum eigen suatu matriks. Keuntungan dari metode ini adalah kemampuannya menangani matriks non-simetri dan mengkonsolidasikan informasi struktural dari matriks melalui proses berulang.

Dekomposisi QR juga digunakan dalam analisis numerik, pemrosesan sinyal, dan pembelajaran mesin, terutama ketika stabilitas numerik dan ortogonalitas menjadi penting. Karena kemampuan QR decomposition dalam menjaga kestabilan komputasi dan struktur geometri data, metode ini menjadi alat utama dalam berbagai bidang ilmiah dan teknis. Dengan berbagai metode implementasinya dan dukungan dari perangkat lunak komputasi ilmiah, dekomposisi QR merupakan teknik faktorisasi yang tak tergantikan dalam praktik komputasi numerik modern.

3. *Singular Value Decomposition* (SVD)

Singular Value Decomposition (SVD) adalah salah satu teknik dekomposisi matriks paling kuat dan serbaguna dalam aljabar linier numerik. Berbeda dengan dekomposisi LU atau QR yang hanya berlaku pada matriks dengan syarat tertentu (seperti matriks persegi atau penuh-rangking), SVD dapat diterapkan pada semua jenis matriks, baik persegi, persegi panjang, penuh-rangking maupun rangking rendah. Secara formal, jika diberikan matriks $A \in R^{m \times n}$, maka SVD memfaktorkan matriks tersebut menjadi hasil perkalian tiga matriks:

$$A = U \Sigma V^T$$

Di sini, $U \in R^{m \times m}$ adalah matriks ortogonal yang kolom-kolomnya disebut *left singular vectors*.

$$\Sigma \in R^{m \times n}$$

adalah matriks diagonal (atau hampir diagonal) yang elemen-elemen diagonalnya adalah bilangan non-negatif dan disebut *singular values*.

$$V^T \in R^{n \times n}$$

adalah transpose dari matriks ortogonal V, dengan kolom-kolom V disebut *right singular vectors*. Nilai-nilai dalam Σ biasanya disusun

dalam urutan menurun dan memberikan ukuran kontribusi dari masing-masing komponen basis terhadap struktur asli data.

Keunggulan utama SVD terletak pada stabilitas numerik dan fleksibilitasnya, sehingga sangat cocok untuk pemrosesan matriks yang tidak simetris, tidak persegi, bahkan ketika tidak memiliki invers. Dalam konteks reduksi dimensi dan kompresi data, SVD memungkinkan kita melakukan aproksimasi matriks dengan hanya mempertahankan beberapa singular values terbesar dan mengabaikan yang kecil, sehingga informasi utama tetap terjaga sementara kompleksitas dikurangi. Teknik ini menjadi dasar dalam *Principal Component Analysis* (PCA), di mana vektor-vektor singular dari SVD digunakan sebagai sumbu baru (komponen utama) yang memaksimalkan variansi data.

Pada kompresi citra digital, misalnya, SVD dapat digunakan untuk menyimpan representasi gambar dalam *basis singular vectors*. Dengan hanya menyimpan sejumlah kecil *singular values* dan vektor terkait, gambar dapat direkonstruksi dengan kualitas yang masih baik namun ukuran file jauh lebih kecil. Selain itu, dalam *Natural Language Processing* (NLP), SVD digunakan dalam metode *Latent Semantic Analysis* (LSA) untuk menemukan struktur laten dalam dokumen teks, dengan cara mengurai matriks term-document menjadi komponen semantik dominan. SVD juga berperan penting dalam pseudoinvers matriks *(Moore-Penrose inverse*), yaitu ketika kita ingin menyelesaikan sistem Ax=b namun A tidak memiliki invers atau berbentuk tidak persegi. Dengan SVD, kita dapat menghitung solusi terkecil dalam norma Euclidean dengan cara yang stabil dan akurat.

C. Aplikasi dalam Pemrosesan Data dan *Machine Learning*

Pemrograman dan komputasi numerik menjadi fondasi utama dalam pengolahan data dan pengembangan metode *machine learning* (ML) modern. Dalam konteks ini, teknik-teknik komputasi numerik, seperti dekomposisi matriks, optimasi numerik, dan algoritma statistik, berperan penting dalam mengolah data besar, membangun model prediktif, serta meningkatkan akurasi dan efisiensi pembelajaran mesin (Goodfellow, Bengio, & Courville, 2016). Dengan ketersediaan data yang masif dan kebutuhan analisis yang kompleks, pemrograman numerik memungkinkan transformasi data mentah menjadi informasi bermakna serta model yang dapat diandalkan.

1. Pemrosesan Data

Pemrosesan data adalah rangkaian aktivitas yang bertujuan untuk mengubah data mentah menjadi informasi yang bermakna dan berguna untuk pengambilan keputusan, analisis, dan berbagai aplikasi lanjutan. Proses ini sangat penting dalam era digital di mana data dihasilkan secara masif dari berbagai sumber seperti sensor IoT, transaksi bisnis, media sosial, dan sistem informasi lainnya. Data mentah yang tidak terstruktur, bising, dan berdimensi tinggi harus diolah melalui serangkaian tahap agar dapat diekstrak pola, insight, atau model prediktif yang akurat. Oleh karena itu, pemrosesan data menjadi fondasi utama dalam bidang data science dan *machine learning*.

Tahap awal dalam pemrosesan data adalah pengumpulan data, di mana data dikumpulkan dari berbagai sumber dengan berbagai format. Data tersebut kemudian mengalami pembersihan (*data cleaning*) untuk mengatasi masalah seperti nilai yang hilang (*missing values*), duplikasi, dan kesalahan input. Pembersihan data penting karena data yang tidak konsisten atau rusak dapat menghasilkan model yang bias dan tidak akurat (Rahm & Do, 2000). Selanjutnya, data mengalami transformasi dan normalisasi. Transformasi mencakup pengubahan format, pengkodean variabel kategorikal menjadi numerik, dan penanganan outlier. Normalisasi, seperti skala min-max atau standardisasi, dilakukan agar fitur-fitur data berada pada rentang yang sama, sehingga algoritma *machine learning* dapat bekerja lebih efektif dan cepat konvergen (Han, Kamber, & Pei, 2011).

Reduksi dimensi juga merupakan tahap penting dalam pemrosesan data, terutama untuk dataset berdimensi tinggi yang dapat menyebabkan masalah *curse of dimensionality*. Teknik seperti *Principal Component Analysis* (PCA) dan *Singular Value Decomposition* (SVD) digunakan untuk mereduksi fitur menjadi komponen-komponen utama yang mewakili variansi terbesar dari data tanpa kehilangan informasi penting. Dengan cara ini, kompleksitas data berkurang, yang mempercepat proses pelatihan model sekaligus meningkatkan interpretabilitas (Jolliffe, 2002). Setelah data siap, dilakukan eksplorasi data (*Exploratory Data Analysis*, EDA) untuk memahami karakteristik data, distribusi, korelasi antar variabel, dan pola tersembunyi. Visualisasi data seperti histogram, scatter plot, dan heatmap sangat membantu dalam tahap ini untuk mengidentifikasi tren dan anomali (Tukey, 1977).

Pemrosesan data juga mencakup teknik feature engineering, yaitu proses menciptakan fitur baru yang lebih representatif berdasarkan fitur asli. Misalnya, menggabungkan beberapa fitur menjadi indeks atau menghitung rata-rata per periode waktu tertentu dalam data waktu (*time-series*). *Feature engineering* yang baik dapat meningkatkan performa model secara signifikan (Kuhn & Johnson, 2013). Dalam konteks data streaming dan *big data*, pemrosesan data harus dilakukan secara real-time dan skalabel. Teknologi seperti Apache Hadoop dan Apache Spark memungkinkan pemrosesan paralel dan distribusi data yang efisien di cluster komputer besar. Pendekatan ini sangat penting untuk menangani volume data yang sangat besar dengan kecepatan tinggi (Zaharia *et al*., 2010). Pemrosesan data juga mengantisipasi aspek keamanan dan privasi, dengan menerapkan teknik seperti enkripsi data dan anonimasi agar data sensitif tidak disalahgunakan selama proses analisis (Dwork, 2008).

2. Machine Learning

Machine learning (ML) merupakan cabang dari kecerdasan buatan (*artificial intelligence*) yang memungkinkan sistem komputer untuk belajar dari data dan meningkatkan performa tanpa diprogram secara eksplisit. ML berfokus pada pengembangan algoritma dan model matematis yang dapat mengidentifikasi pola, membuat prediksi, dan mengambil keputusan berdasarkan data yang tersedia. Konsep inti ML adalah bahwa sistem belajar dengan mengenali pola dalam data dan menggeneralisasi pola tersebut ke data baru yang belum pernah ditemui sebelumnya (Mitchell, 1997).

ML dapat dibagi menjadi beberapa kategori utama berdasarkan jenis data dan cara belajar, yaitu *supervised learning, unsupervised learning,* dan *reinforcement learning*. Pada *supervised learning*, model dilatih menggunakan data berlabel, di mana input dan output yang diinginkan sudah diketahui. Contoh algoritma *supervised learning* meliputi regresi linier, pohon keputusan, dan *neural networks*. Model bertujuan mempelajari hubungan antara input dan output agar dapat memprediksi output pada data baru dengan akurat (Hastie *et al*., 2009). Sebaliknya, pada *unsupervised learning*, data yang digunakan tidak berlabel, sehingga model mencoba menemukan struktur atau pola tersembunyi dalam data. Teknik umum termasuk clustering seperti k-means dan *hierarchical clustering*, serta reduksi dimensi seperti PCA.

Unsur utama di sini adalah mengenali kelompok data atau fitur penting tanpa panduan output (Aggarwal, 2015).

Reinforcement learning berbeda dengan kedua pendekatan sebelumnya karena sistem belajar melalui interaksi dengan lingkungan dan mendapatkan umpan balik berupa reward atau penalti. Pendekatan ini banyak digunakan dalam pengembangan agen cerdas untuk permainan dan robotika (Sutton & Barto, 2018). Model ML modern semakin kompleks dengan hadirnya *deep learning*, yaitu subbidang yang menggunakan jaringan saraf tiruan bertingkat (*deep neural networks*). Deep learning mampu mengolah data yang sangat besar dan kompleks, seperti gambar, suara, dan teks, dengan tingkat akurasi yang tinggi. Jaringan saraf konvolusional (CNN) untuk pengolahan citra dan jaringan saraf rekuren (RNN) untuk data urutan adalah contoh aplikasi deep learning yang sangat populer (Goodfellow *et al*., 2016).

Proses *machine learning* umumnya melibatkan beberapa tahapan: pengumpulan data, pembersihan dan praproses data, pemilihan dan ekstraksi fitur, pemilihan model, pelatihan model, validasi, dan evaluasi performa. Pemilihan fitur yang relevan sangat penting karena dapat meningkatkan efisiensi dan akurasi model. Selain itu, teknik validasi seperti cross-validation digunakan untuk menghindari overfitting, yaitu kondisi di mana model terlalu menghafal data latih sehingga gagal menggeneralisasi ke data baru (Kuhn & Johnson, 2013). ML juga memanfaatkan metode optimasi numerik untuk meminimalkan fungsi kerugian (*loss function*) selama pelatihan. Algoritma optimasi seperti gradient descent dan variannya berperan penting dalam mempercepat proses pelatihan dan menemukan solusi optimal (Bottou, 2010). Aplikasi *machine learning* sangat luas dan berkembang pesat di berbagai bidang. Dalam kesehatan, ML digunakan untuk diagnosis penyakit dan analisis citra medis. Dalam bisnis, ML mendukung analisis pelanggan, prediksi penjualan, dan sistem rekomendasi. Di bidang transportasi, ML menggerakkan teknologi kendaraan otonom dan prediksi lalu lintas (Jordan & Mitchell, 2015).

D. Optimasi Performa Komputasi Matriks

Optimasi performa komputasi matriks merupakan aspek penting dalam komputasi numerik, ilmu komputer, dan berbagai aplikasi teknik serta ilmiah. Matriks adalah struktur data dasar yang digunakan secara

luas dalam pemodelan matematis, pemrosesan sinyal, pembelajaran mesin, grafik komputer, dan simulasi ilmiah. Namun, komputasi matriks, terutama pada skala besar, bisa sangat intensif secara komputasi dan memori. Oleh karena itu, mengoptimalkan kinerja operasi matriks sangat penting untuk mengurangi waktu komputasi dan pemakaian sumber daya komputer (Demmel, 1997).

1. Pemilihan Algoritma yang Efisien

Pemilihan algoritma yang efisien merupakan aspek krusial dalam optimasi performa komputasi matriks karena algoritma menentukan bagaimana operasi matematika dijalankan dan berdampak langsung pada kecepatan serta penggunaan sumber daya komputasi. Dalam konteks komputasi matriks, efisiensi algoritma terutama diukur dari kompleksitas waktu (*time complexity*) dan kompleksitas ruang (*space complexity*) yang diperlukan untuk menyelesaikan operasi, seperti perkalian matriks, invers matriks, dekomposisi, dan penyelesaian sistem linear (Demmel, 1997).

Misalnya, perkalian matriks standar menggunakan metode iteratif dengan kompleksitas $O(n^3)$ untuk matriks berukuran $n \times n$. Algoritma ini cukup sederhana dan mudah diimplementasikan, tetapi menjadi sangat lambat untuk matriks besar. Oleh karena itu, algoritma alternatif seperti algoritma Strassen yang memiliki kompleksitas lebih rendah yaitu O(n^2.81) dapat dipilih untuk mempercepat komputasi, meskipun implementasinya lebih rumit dan memiliki overhead yang signifikan pada matriks berukuran kecil (Strassen, 1969). Algoritma yang lebih canggih, seperti algoritma Coppersmith-Winograd, bahkan menurunkan kompleksitas perkalian matriks hingga sekitar O(n^2.37), tetapi biasanya hanya digunakan dalam penelitian dan aplikasi khusus karena kompleksitas implementasi yang tinggi (Williams, 2012).

Pemilihan algoritma juga harus mempertimbangkan karakteristik matriks, seperti kepadatan elemen (dense vs sparse). Untuk matriks jarang, algoritma khusus yang mengabaikan elemen nol dapat mengurangi komputasi dan penggunaan memori secara drastis. Contohnya adalah penggunaan metode iteratif seperti Conjugate Gradient atau GMRES yang lebih efisien untuk sistem linear sparse dibandingkan metode langsung seperti eliminasi Gauss (Saad, 2003). Selain itu, algoritma harus diadaptasi dengan arsitektur perangkat keras yang digunakan. Algoritma yang mendukung paralelisasi atau yang

dioptimalkan untuk memanfaatkan cache dan instruksi SIMD akan jauh lebih efisien pada sistem modern (Kirk & Hwu, 2016). Dengan demikian, pemilihan algoritma yang efisien adalah keputusan strategis yang menggabungkan analisis kompleksitas, karakteristik data, dan pemahaman terhadap perangkat keras agar hasil komputasi matriks dapat dicapai secara optimal dan efektif.

2. Pengelolaan Memori

Pengelolaan memori adalah salah satu faktor kunci dalam optimasi performa komputasi matriks karena kecepatan akses data sangat menentukan efisiensi keseluruhan operasi. Dalam komputasi matriks, data biasanya disimpan dalam array dua dimensi yang ukurannya bisa sangat besar, sehingga cara penyimpanan dan pengaksesan data harus diatur sedemikian rupa agar dapat memanfaatkan hirarki memori komputer secara optimal (Hennessy & Patterson, 2017).

Komputer modern memiliki beberapa tingkat memori, mulai dari register, cache (L1, L2, L3), RAM, hingga penyimpanan sekunder. Cache, yang berukuran kecil tetapi sangat cepat, sangat penting dalam mempercepat akses data. Oleh karena itu, strategi pengelolaan memori seperti blocking atau tiling diterapkan untuk memecah operasi matriks menjadi sub-bagian kecil yang dapat dimuat sekaligus ke dalam cache. Teknik ini mengurangi cache miss dan latensi akses memori, sehingga meningkatkan throughput komputasi (Gustavson, 1997).

Pola akses memori juga penting. Pengaksesan data secara kontigu (sekuensial) lebih cepat dibandingkan akses acak karena prinsip spatial locality yang dimanfaatkan oleh sistem cache. Oleh sebab itu, dalam operasi matriks seperti perkalian atau dekomposisi, pengaturan iterasi yang memprioritaskan akses baris demi baris atau kolom demi kolom sangat dianjurkan agar data dapat diakses secara efisien. Pengelolaan memori juga mempertimbangkan format penyimpanan matriks. Untuk matriks padat, penyimpanan secara row-major atau column-major mempengaruhi cara data diakses. Sementara pada matriks jarang, format seperti CSR (*Compressed Sparse Row*) dan CSC (*Compressed Sparse Column*) menghemat ruang memori dan mengurangi waktu akses elemen non-nol saja, sehingga meningkatkan performa operasi (Saad, 2003).

3. Paralelisasi Komputasi

Paralelisasi komputasi merupakan teknik penting dalam optimasi performa operasi matriks yang memanfaatkan kemampuan perangkat keras modern untuk menjalankan banyak proses secara bersamaan. Pada dasarnya, paralelisasi membagi tugas komputasi besar menjadi bagian-bagian kecil yang dapat dikerjakan secara simultan oleh beberapa inti (core) prosesor atau unit pemrosesan grafis (GPU). Pendekatan ini sangat efektif mengingat operasi matriks, seperti perkalian atau dekomposisi, sering kali dapat diparalelisasi karena setiap elemen hasil biasanya dihitung secara independen (Kirk & Hwu, 2016).

Komputasi paralel dapat dilakukan pada berbagai tingkatan. Pada level CPU, model pemrograman seperti OpenMP memungkinkan pembagian pekerjaan ke beberapa core melalui threading. Sementara pada skala lebih besar, MPI (*Message Passing Interface*) digunakan untuk mengkoordinasi komputasi di cluster komputer, mendistribusikan data dan tugas ke banyak node. Di sisi lain, GPU dengan ribuan core kecil yang dirancang untuk komputasi paralel massal, menjadi sangat populer untuk mempercepat operasi matriks besar dengan menggunakan platform seperti CUDA atau OpenCL (Nickolls *et al*., 2008).

Efektivitas paralelisasi sangat bergantung pada bagaimana tugas dibagi dan komunikasi antar unit dilakukan. Pembagian tugas harus seimbang agar tidak ada core yang idle terlalu lama, dan overhead komunikasi antar unit harus diminimalkan agar keuntungan paralelisasi tidak hilang. Teknik seperti data parallelism yang membagi data menjadi potongan-potongan kecil dan task parallelism yang membagi proses menjadi tugas-tugas berbeda sering digunakan dalam optimasi komputasi matriks (Grama *et al*., 2003).

Paralelisasi juga memungkinkan pemrosesan matriks yang sangat besar yang tidak mungkin dilakukan secara efisien oleh satu core saja. Banyak perpustakaan numerik populer, seperti Intel MKL dan cuBLAS, sudah mengimplementasikan paralelisasi secara otomatis untuk memanfaatkan perangkat keras modern sehingga pengguna dapat merasakan peningkatan performa tanpa perlu menulis kode paralel secara eksplisit.

4. Instruksi SIMD

Instruksi SIMD (*Single Instruction, Multiple Data*) adalah fitur pada prosesor modern yang memungkinkan eksekusi satu instruksi yang

sama secara simultan pada beberapa data sekaligus. Konsep SIMD sangat efektif dalam mempercepat komputasi matriks dan operasi vektor karena banyak dari operasi ini melibatkan penerapan fungsi yang sama pada elemen-elemen data yang berbeda secara paralel (Williams *et al*., 2009). Dengan SIMD, misalnya, sebuah prosesor dapat melakukan penjumlahan pada empat atau delapan pasangan elemen matriks sekaligus dalam satu siklus instruksi, dibandingkan dengan memproses satu elemen per siklus pada arsitektur tradisional.

Pemanfaatan instruksi SIMD memerlukan dukungan perangkat keras serta compiler yang mampu menghasilkan kode mesin yang menggunakan instruksi ini. Contoh arsitektur yang mendukung SIMD antara lain Intel SSE (*Streaming SIMD Extensions*), AVX (*Advanced Vector Extensions*), dan ARM NEON untuk prosesor mobile. Instruksi SIMD umumnya bekerja dengan register khusus yang dapat menampung data vektor berukuran 128-bit, 256-bit, atau lebih, memungkinkan operasi simultan pada banyak elemen data (Fog, 2016).

Penggunaan SIMD sangat cocok untuk algoritma yang berstruktur data paralel, seperti perkalian matriks, transformasi Fourier, dan operasi filter dalam pemrosesan sinyal. Dengan SIMD, jumlah instruksi yang harus dieksekusi berkurang drastis, sehingga meningkatkan throughput dan mengurangi latensi. Namun, optimalisasi dengan SIMD memerlukan penyesuaian pola akses data agar data tersimpan secara kontigu di memori dan alignment yang tepat agar tidak terjadi penalti performa (Hennessy & Patterson, 2017).

Meski SIMD meningkatkan performa secara signifikan, ada beberapa keterbatasan seperti ukuran register terbatas dan kebutuhan data yang terstruktur rapi. Oleh karena itu, pemrogram perlu mempertimbangkan desain algoritma dan struktur data agar sesuai dengan model SIMD. Banyak perpustakaan matematika dan multimedia sudah memanfaatkan instruksi SIMD secara transparan sehingga pengguna dapat merasakan peningkatan performa tanpa pengetahuan detail tentang instruksi ini.

5. Pemilihan Format Penyimpanan Matriks

Pemilihan format penyimpanan matriks merupakan aspek penting dalam optimasi komputasi numerik karena berpengaruh langsung pada efisiensi penggunaan memori dan kecepatan akses data selama operasi matriks. Format penyimpanan matriks yang tepat sangat

bergantung pada karakteristik matriks itu sendiri, terutama apakah matriks tersebut padat (*dense*) atau jarang (*sparse*) (Saad, 2003). Untuk matriks padat, format penyimpanan yang umum digunakan adalah penyimpanan secara row-major atau column-major, di mana elemen-elemen disimpan secara berurutan berdasarkan baris atau kolom. Format ini memudahkan akses sekuensial yang efisien pada memori, sehingga cocok untuk operasi matriks yang membutuhkan pembacaan elemen secara linear, seperti perkalian matriks konvensional atau dekomposisi LU (Golub & Van Loan, 2013).

Untuk matriks jarang yang sebagian besar elemennya bernilai nol, penyimpanan dalam format padat akan sangat membuang-buang ruang memori dan memperlambat komputasi. Oleh karena itu, format penyimpanan khusus seperti *Compressed Sparse Row* (CSR), *Compressed Sparse Column* (CSC), atau *Coordinate* (COO) digunakan. Format-format ini hanya menyimpan elemen non-nol dan indeks posisinya, sehingga secara signifikan mengurangi kebutuhan memori dan mempercepat operasi yang hanya fokus pada elemen non-nol (Saad, 2003).

Pemilihan format juga mempertimbangkan jenis operasi yang akan dilakukan. Misalnya, CSR lebih efisien untuk operasi perkalian matriks dengan vektor karena akses baris yang cepat, sedangkan CSC lebih optimal untuk operasi yang membutuhkan akses kolom. Selain itu, format penyimpanan harus kompatibel dengan perpustakaan numerik dan perangkat keras yang digunakan agar bisa memanfaatkan optimasi paralelisasi dan instruksi SIMD (Kirk & Hwu, 2016).

BAB X
STUDI KASUS DAN PROYEK APLIKASI

Bab Studi Kasus dan Proyek Aplikasi hadir sebagai bagian penting dalam buku ini untuk menjembatani teori dan praktik dalam pemrograman serta komputasi numerik. Pada bab ini, pembaca diajak untuk melihat secara langsung bagaimana konsep-konsep matematis dan algoritma numerik yang telah dipelajari dapat diterapkan dalam menyelesaikan masalah nyata dari berbagai bidang, seperti teknik, fisika, biologi, ekonomi, dan lain-lain. Melalui serangkaian studi kasus yang dipilih secara representatif, pembaca akan memahami proses pengembangan solusi numerik mulai dari perumusan masalah, pemilihan metode yang tepat, hingga implementasi menggunakan bahasa pemrograman populer seperti Python dan MATLAB. Proyek-proyek aplikasi yang disajikan juga bertujuan untuk mengasah keterampilan analisis, pemrograman, serta kemampuan interpretasi hasil komputasi, sehingga pembaca tidak hanya memahami teori, tetapi juga mampu mengaplikasikannya secara efektif dalam konteks dunia nyata.

A. Simulasi Perpindahan Panas

Perpindahan panas adalah proses di mana energi panas berpindah dari satu bagian ke bagian lain akibat perbedaan suhu. Dalam teknik dan ilmu terapan, simulasi perpindahan panas sangat penting untuk merancang sistem termal, seperti pendingin elektronik, sistem pemanas, hingga proses manufaktur. Pada studi kasus ini, kita akan membahas perpindahan panas dalam sebuah batang logam satu dimensi yang mengalami perubahan suhu sepanjang batang seiring waktu. Model

matematis yang digunakan adalah Persamaan Difusi Panas 1D (*Heat Equation*):

$$\frac{\partial u}{\partial t} = \alpha \frac{\partial^2 u}{\partial x^2}$$

dengan

- $u(x,t)$ = suhu pada posisi x dan waktu t,
- α = diffusivity termal batang (konstanta material).

1. **Kondisi Awal dan Batas**
   a. Panjang batang: L meter
   b. Waktu simulasi: T detik
   c. Kondisi awal suhu batang: misal suhu awal seragam, $u(x,0) = u_0$
   d. Kondisi batas: suhu pada kedua ujung batang tetap konstan, misalnya $u(0,t) = u(L,t) = T_0$

2. **Metode Numerik: Metode Elemen Hingga atau Finite Difference**

Untuk menyelesaikan persamaan ini secara numerik, metode finite difference sering digunakan. Misalnya, metode eksplisit Euler maju:

$$u_i^{n+1} = u_i^n + \frac{\alpha \Delta t}{(\Delta x)^2}(u_{i+1}^n - 2u_i^n + u_{i-1}^n)$$

Dimana:

- u_i^n = suhu di titik grid ke-i pada waktu ke-n
- Δx = jarak antar titik grid
- Δt = interval waktu simulasi

Untuk stabilitas metode ini, berlaku kondisi:

$$\frac{\alpha \Delta t}{(\Delta x)^2} \leq \frac{1}{2}$$

Implementasi dalam Python

```python
import numpy as np
import matplotlib.pyplot as plt

# Parameter
L = 1.0       # panjang batang (m)
T = 0.5       # waktu simulasi (s)
alpha = 0.01  # diffusivity termal (m^2/s)

nx = 50       # jumlah grid spatial
dx = L / (nx - 1)
dt = 0.0005   # interval waktu
nt = int(T / dt)

# Inisialisasi suhu
u = np.ones(nx) * 20   # suhu awal 20 derajat Celsius
u[0] = 100             # suhu ujung kiri 100 C
u[-1] = 50             # suhu ujung kanan 50 C

u_new = u.copy()

for n in range(nt):
    for i in range(1, nx-1):
        u_new[i] = u[i] + alpha * dt / dx**2 * (u[i+1] - 2*u[i] + u[i-1])
    u[:] = u_new[:]

# Visualisasi
x = np.linspace(0, L, nx)
plt.plot(x, u, label='Suhu akhir')
plt.xlabel('Posisi (m)')
plt.ylabel('Suhu (°C)')
plt.title('Simulasi Perpindahan Panas pada Batang 1D')
plt.legend()
plt.show()
```

Implementasi dalam MATLAB

```
L = 1.0;       % panjang batang (m)
T = 0.5;       % waktu simulasi (s)
alpha = 0.01;  % diffusivity termal (m^2/s)

nx = 50;       % jumlah grid spatial
dx = L/(nx-1);
dt = 0.0005;   % interval waktu
nt = floor(T/dt);

u = 20*ones(nx,1);   % suhu awal 20 C
u(1) = 100;          % ujung kiri 100 C
u(end) = 50;         % ujung kanan 50 C

u_new = u;

for n=1:nt
    for i=2:nx-1
        u_new(i) = u(i) + alpha*dt/dx^2*(u(i+1) - 2*u(i) + u(i-1));
    end
    u = u_new;
end

x = linspace(0,L,nx);
plot(x,u,'-o')
xlabel('Posisi (m)')
ylabel('Suhu (°C)')
title('Simulasi Perpindahan Panas pada Batang 1D')
grid on

```

Simulasi ini menggambarkan bagaimana suhu dalam batang logam berubah dari kondisi awal dan batas yang ditetapkan sampai mencapai distribusi suhu akhir yang stabil. Hasil visualisasi dari Python dan MATLAB menunjukkan grafik suhu terhadap posisi sepanjang batang setelah waktu simulasi tertentu. Pada kedua bahasa, pendekatan numeriknya sama, yaitu metode beda hingga eksplisit dengan stabilitas dipastikan lewat pemilihan Δt dan Δx sesuai aturan numerik. Perbedaan utama terletak pada sintaks dan cara pengelolaan array atau vektor.

Studi kasus ini dapat diperluas ke dimensi lebih tinggi atau dengan kondisi batas dan sumber panas yang lebih kompleks. Selain itu, teknik numerik lain seperti metode implisit atau Crank-Nicolson dapat digunakan untuk mendapatkan kestabilan yang lebih baik dengan interval waktu yang lebih besar.

Simulasi perpindahan panas merupakan contoh klasik aplikasi komputasi numerik yang penting dalam berbagai disiplin. Pemahaman

teori dan keterampilan pemrograman dalam berbagai bahasa sangat membantu untuk mengembangkan solusi sesuai kebutuhan praktis. Melalui contoh implementasi di Python, MATLAB, dan C++, pembaca dapat memahami cara menyusun model numerik, mengimplementasikan algoritma, dan melakukan analisis hasil simulasi secara efektif.

Gunakan metode selisih hingga (Finite Difference) untuk mendekati solusi dari persamaan diferensial orde dua berikut:

$$\frac{d^2y}{dx^2} = -y, \quad \text{dengan} \quad y(0) = 0, \quad y(1) = 0$$

Langkah 1: Diskretisasi

Panjang interval: $a = 0, b = 1$, dibagi 4 → $h = 0{,}25$
Titik-titik: $x_0 = 0, x_1 = 0{,}25, x_2 = 0{,}5, x_3 = 0{,}75, x_4 = 1$

Gunakan pendekatan selisih hingga untuk turunan kedua:

$$\frac{d^2y}{dx^2} \approx \frac{y_{i+1} - 2y_i + y_{i-1}}{h^2}$$

Substitusi ke persamaan:

$$\frac{y_{i+1} - 2y_i + y_{i-1}}{h^2} = -y_i \Rightarrow y_{i+1} - (2 + h^2)y_i + y_{i-1} = 0$$

Karena $h = 0{,}25$, maka $h^2 = 0{,}0625$, sehingga:

$$y_{i+1} - 2{,}0625y_i + y_{i-1} = 0$$

Langkah 2: Sistem Persamaan Linear

Gunakan kondisi batas: $y_0 = 0, y_4 = 0$

- Persamaan 1 (untuk $i = 1$): $y_2 - 2{,}0625y_1 + y_0 = 0 \Rightarrow y_2 - 2{,}0625y_1 = 0$
- Persamaan 2 (untuk $i = 2$): $y_3 - 2{,}0625y_2 + y_1 = 0$
- Persamaan 3 (untuk $i = 3$): $y_4 - 2{,}0625y_3 + y_2 = 0 \Rightarrow -2{,}0625y_3 + y_2 = 0$

Langkah 3: Penyelesaian Sistem

- Dari (1): $y_2 = 2{,}0625 y_1$
- Dari (3): $y_2 = 2{,}0625 y_3$
- Maka $y_1 = y_3$

Substitusi: Misal $y_1 = A$, maka:

- $y_2 = 2{,}0625A$
- $y_3 = A$

B. Pemodelan Populasi dan Epidemi

Pemodelan populasi dan epidemi merupakan alat penting dalam ilmu kesehatan masyarakat untuk memahami dan memprediksi penyebaran penyakit menular. Salah satu model dasar yang populer adalah model SIR, yang membagi populasi ke dalam tiga kategori utama: *Susceptible* (rentan terinfeksi), *Infected* (terinfeksi), dan *Recovered* (sembuh dan kebal). Model ini membantu para ilmuwan dan pembuat kebijakan mengantisipasi laju penyebaran penyakit dan merancang strategi pengendalian.

Contoh nyata adalah pandemi COVID-19 yang melanda dunia sejak akhir 2019. Model SIR digunakan untuk memprediksi puncak kasus, durasi wabah, serta efek intervensi seperti pembatasan sosial dan vaksinasi. Dalam studi kasus ini, kita akan memodelkan dinamika penyebaran COVID-19 menggunakan model SIR dengan parameter yang disesuaikan dari data epidemiologi awal. Persamaan diferensial model SIR adalah:

$$\frac{dS}{dt} = -\beta \frac{SI}{N}$$

$$\frac{dI}{dt} = \beta \frac{SI}{N} - \gamma I$$

$$\frac{dR}{dt} = \gamma I$$

- $S(t)$: Jumlah individu rentan pada waktu *t*
- $I(t)$: Jumlah individu terinfeksi pada waktu *t*
- $R(t)$: Jumlah individu sembuh/kebal pada waktu *t*
- $N = S + I + R$: Total populasi (diasumsikan konstan)
- β : Laju penularan
- γ : Laju pemulihan

Parameter kunci yang digunakan untuk COVID-19 berdasarkan literatur awal adalah:

- β : 0.3 per hari (menunjukkan rata-rata tiap orang menularkan virus ke 0.3 orang per hari)
- γ : 0.1 per hari (rata-rata durasi infeksi 10 hari)

1. Kondisi Awal

a. Total populasi N = 1.000.000
b. Awal infeksi $I_0 = 1$ orang
c. Rentan $S_0 = N - I_0 = 999.999$
d. Sembuh $R_0 = 0$

2. Metode Numerik

Untuk menyelesaikan sistem persamaan diferensial ini, kita gunakan metode Euler maju dengan diskritisasi waktu Δt. Secara numerik:

$$S_{n+1} = S_n - \beta \frac{S_n I_n}{N} \Delta t$$

$$I_{n+1} = I_n + \left(\beta \frac{S_n I_n}{N} - \gamma I_n \right) \Delta t$$

$$R_{n+1} = R_n + \gamma I_n \Delta t$$

Implementasi Python

```python
import numpy as np
import matplotlib.pyplot as plt

# Parameter
N = 1_000_000
beta = 0.3
gamma = 0.1
dt = 0.1
t_max = 160

# Waktu simulasi
steps = int(t_max / dt)
t = np.linspace(0, t_max, steps)

# Inisialisasi
S = np.zeros(steps)
I = np.zeros(steps)
R = np.zeros(steps)

S[0] = N - 1
I[0] = 1
R[0] = 0

# Iterasi Euler
for n in range(steps - 1):
    dS = -beta * S[n] * I[n] / N
    dI = beta * S[n] * I[n] / N - gamma
    dR = gamma * I[n]

    S[n+1] = S[n] + dS * dt
    I[n+1] = I[n] + dI * dt
    R[n+1] = R[n] + dR * dt

# Visualisasi
plt.plot(t, S, label='Rentan (S)')
plt.plot(t, I, label='Terinfeksi (I)')
plt.plot(t, R, label='Sembuh (R)')
plt.xlabel('Hari')
plt.ylabel('Jumlah Orang')
plt.title('Model SIR Penyebaran COVID-19')
plt.legend()
plt.grid()
plt.show()
```

Implementasi MATLAB

```
N = 1e6;
beta = 0.3;
gamma = 0.1;
dt = 0.1;
t_max = 160;

steps = floor(t_max / dt);
t = linspace(0, t_max, steps);

S = zeros(1, steps);
I = zeros(1, steps);
R = zeros(1, steps);

S(1) = N - 1;
I(1) = 1;
R(1) = 0;

for n = 1:steps-1
    dS = -beta * S(n) * I(n) / N;
    dI = beta * S(n) * I(n) / N - gamma * I(n);
    dR = gamma * I(n);

    S(n+1) = S(n) + dS * dt;
    I(n+1) = I(n) + dI * dt;
    R(n+1) = R(n) + dR * dt;
end

plot(t, S, '-b', t, I, '-r', t, R, '-g');
xlabel('Hari');
ylabel('Jumlah Orang');
title('Model SIR Penyebaran COVID-19');
legend('Rentan (S)', 'Terinfeksi (I)', 'Sembuh (R)');
grid on;
```

Implementasi C++

```cpp
#include <iostream>
#include <vector>
#include <fstream>

using namespace std;

int main() {
    const double N = 1e6;
    const double beta = 0.3;
    const double gamma = 0.1;
    const double dt = 0.1;
    const double t_max = 160;
    int steps = static_cast<int>(t_max / dt);

    vector<double> S(steps, 0), I(steps, 0), R(steps, 0);
    S[0] = N - 1;
    I[0] = 1;
    R[0] = 0;

    for (int n = 0; n < steps - 1; ++n) {
        double dS = -beta * S[n] * I[n] / N;
        double dI = beta * S[n] * I[n] / N - gamma * I[n];
        double dR = gamma * I[n];

        S[n+1] = S[n] + dS * dt;
        I[n+1] = I[n] + dI * dt;
        R[n+1] = R[n] + dR * dt;
    }

    ofstream file("SIR_output.txt");
    for (int i = 0; i < steps; ++i) {
        file << i*dt << "\t" << S[i] << "\t" << I[i] << "\t" << R[i] << "\n";
    }
    file.close();

    }
    file.close();

    cout << "Simulasi selesai. Data disimpan di 'SIR_output.txt'." << endl;
    return 0;
}
```

Simulasi model SIR ini menggambarkan bagaimana jumlah populasi yang rentan, terinfeksi, dan sembuh berubah selama 160 hari. Grafik yang dihasilkan umumnya menunjukkan:

1. Awal wabah, jumlah terinfeksi meningkat tajam, sementara populasi rentan menurun.
2. Setelah mencapai puncak infeksi, jumlah pasien terinfeksi mulai menurun karena bertambahnya populasi yang sembuh.

3. Populasi sembuh meningkat secara konsisten, menandakan akumulasi kekebalan.

Model ini sangat berguna untuk memperkirakan beban sistem kesehatan, merencanakan intervensi, dan mengukur dampak kebijakan pembatasan sosial atau vaksinasi. Namun, model SIR sederhana memiliki keterbatasan karena mengasumsikan populasi homogen dan konstan serta tidak memasukkan faktor-faktor seperti mobilitas, mutasi virus, atau perilaku manusia. Model lebih kompleks seperti SEIR, agent-based, atau metapopulasi dapat dipakai untuk analisis lebih rinci.

C. Optimasi Portofolio dan Pemodelan Finans

Optimasi portofolio adalah proses memilih kombinasi aset investasi yang optimal untuk memaksimalkan return yang diharapkan dengan risiko yang dapat diterima. Salah satu pendekatan klasik adalah model Mean-Variance yang diperkenalkan oleh Harry Markowitz pada tahun 1952. Model ini menjadi dasar teori portofolio modern (*Modern Portfolio Theory*, MPT). Pada konteks nyata, investor di Bursa Efek Indonesia (BEI) sering dihadapkan pada pilihan beragam saham dengan return dan risiko yang berbeda. Studi ini mengaplikasikan model Markowitz untuk menentukan bobot optimal pada portofolio terdiri dari beberapa saham unggulan di BEI dengan tujuan meminimalkan risiko untuk tingkat return yang diharapkan.

Misalkan kita menggunakan data return historis bulanan selama 3 tahun terakhir dari 4 saham unggulan di BEI:

1. Saham A (contoh: PT Telekomunikasi Indonesia Tbk)
2. Saham B (contoh: PT Bank Central Asia Tbk)
3. Saham C (contoh: PT Unilever Indonesia Tbk)
4. Saham D (contoh: PT Astra International Tbk)

Data return bulanan (dalam persen) disederhanakan sebagai berikut (dalam bentuk matriks):

Bulan	A	B	C	D
1	1.2	1.0	0.8	1.5
2	0.5	1.1	0.7	1.3
...	...	...	...	...
36	1.3	0.9	1.0	1.4

1. Implementasi Python (menggunakan library cvxpy)

```python
import numpy as np
import cvxpy as cp

# Data return bulanan saham (simplifikasi)
returns = np.array([
    [0.012, 0.010, 0.008, 0.015],
    [0.005, 0.011, 0.007, 0.013],
    # ... data lain selama 36 bulan ...
    [0.013, 0.009, 0.010, 0.014]
])

# Hitung rata-rata return dan kovarians
mu = np.mean(returns, axis=0)
Sigma = np.cov(returns.T)

# Target return portofolio
R_target = 0.01

# Variabel bobot
w = cp.Variable(len(mu))

# Fungsi tujuan (minimize risiko)
risk = cp.quad_form(w, Sigma)
objective = cp.Minimize(risk)

# Constraints
constraints = [
    cp.sum(w) == 1,
    w @ mu >= R_target,
    w >= 0
]
```

```python

# Problem optimasi
prob = cp.Problem(objective, constraints)
prob.solve()

print("Status:", prob.status)
print("Bobot optimal portofolio:", w.value)
print("Risiko portofolio (varians):", risk.value)
print("Return portofolio:", w.value @ mu)

```

2. Implementasi MATLAB

```
% Data return bulanan saham
returns = [
    0.012 0.010 0.008 0.015;
    0.005 0.011 0.007 0.013;
    % ... data lain selama 36 bulan ...
    0.013 0.009 0.010 0.014
];

mu = mean(returns);
Sigma = cov(returns);

R_target = 0.01;
n = length(mu);

% Optimasi kuadratik dengan quadprog
H = 2 * Sigma; % quadprog mengharapkan 1/2 x' H x
f = zeros(n, 1);

Aeq = [ones(1,n); mu];
beq = [1; R_target];

lb = zeros(n,1);

options = optimoptions('quadprog','Display','off');
[w, risk] = quadprog(H, f, [], [], Aeq, beq, lb, [], [], options);

fprintf('Bobot optimal portofolio:\n');
disp(w');
fprintf('Risiko portofolio (varians): %.6f\n', risk);
fprintf('Return portofolio: %.6f\n', w' * mu');

```

3. Implementasi C++ (menggunakan pustaka Eigen untuk matriks dan optimasi sederhana)

```cpp
#include <iostream>
#include <Eigen/Dense>
#include <vector>

// Implementasi QP sederhana tidak mudah tanpa pustaka khusus.
// Berikut hanya contoh ilustrasi perhitungan rata-rata dan kovarians.

using namespace Eigen;
using namespace std;

int main() {
    // Data return saham (misal 3 bulan, 4 saham)
    MatrixXd returns(3,4);
    returns << 0.012, 0.010, 0.008, 0.015,
               0.005, 0.011, 0.007, 0.013,
               0.013, 0.009, 0.010, 0.014;

    // Hitung rata-rata return tiap saham
    VectorXd mu = returns.colwise().mean();

    // Hitung kovarians
    MatrixXd centered = returns.rowwise() - mu.transpose();
    MatrixXd Sigma = (centered.adjoint() * centered) / double(returns.rows() - 1);

    cout << "Rata-rata return tiap saham:" << endl << mu << endl;
    cout << "Matriks kovarians:" << endl << Sigma << endl;

    cout << "Untuk optimasi portofolio di C++, disarankan menggunakan pustaka optimasi QP seperti

    return 0;
}
```

Studi kasus ini menggambarkan aplikasi nyata optimasi portofolio di pasar saham Indonesia. Dengan model Markowitz, investor dapat menentukan distribusi investasi yang meminimalkan risiko untuk tingkat return yang diinginkan. Python dan MATLAB memudahkan implementasi melalui pustaka optimasi bawaan, sedangkan di C++ implementasi lengkap memerlukan pustaka tambahan untuk solusi QP.

Optimasi portofolio juga bisa dikembangkan dengan menambahkan batasan realistik, seperti batas bobot maksimum per saham, biaya transaksi, serta model risiko lain seperti CVaR (*Conditional Value at Risk*). Penggunaan model ini mendukung pengambilan keputusan investasi yang lebih rasional dan berbasis data historis, meningkatkan peluang return optimal dengan risiko terkendali.

DAFTAR PUSTAKA

Atkinson, K. (1989). An Introduction to Numerical Analysis (2nd ed.). John Wiley & Sons.

Atkinson, K. E. (1989). An Introduction to Numerical Analysis (2nd ed.). John Wiley & Sons.

Atkinson, K. E. (1989). An Introduction to Numerical Analysis. Wiley.

Burden, R. L., & Faires, J. D. (2010). Numerical Analysis (9th ed.). Brooks/Cole Cengage Learning.

Burden, R. L., & Faires, J. D. (2011). Numerical Analysis (9th ed.). Brooks/Cole, Cengage Learning.

Burden, R. L., & Faires, J. D. (2011). Numerical Analysis (9th ed.). Brooks/Cole.

Chapra, S. C., & Canale, R. P. (2010). Numerical Methods for Engineers (6th ed.). McGraw-Hill Education.

Chapra, S. C., & Canale, R. P. (2010). Numerical Methods for Engineers (6th ed.). McGraw-Hill.

Chapra, S. C., & Canale, R. P. (2015). Numerical Methods for Engineers (7th ed.). McGraw-Hill Education.

Goodfellow, I., Bengio, Y., & Courville, A. (2016). Deep Learning. MIT Press.

Heath, M. T. (2002). Scientific Computing: An Introductory Survey (2nd ed.). McGraw-Hill.

Higham, D., & Higham, N. (2016). MATLAB Guide (3rd Ed.). SIAM.

IEEE Spectrum. (2023). The Top Programming Languages 2023. Retrieved from https://spectrum.ieee.org/top-programming-languages-2023

Jolliffe, I. T. (2002). Principal Component Analysis. Springer.

Kurniawan, D., Subekti, R., & Wardani, S. (2021). Pemilihan Bahasa Pemrograman untuk Komputasi Numerik. Jurnal Teknologi dan Sistem Komputer, 9(2), 111-120.

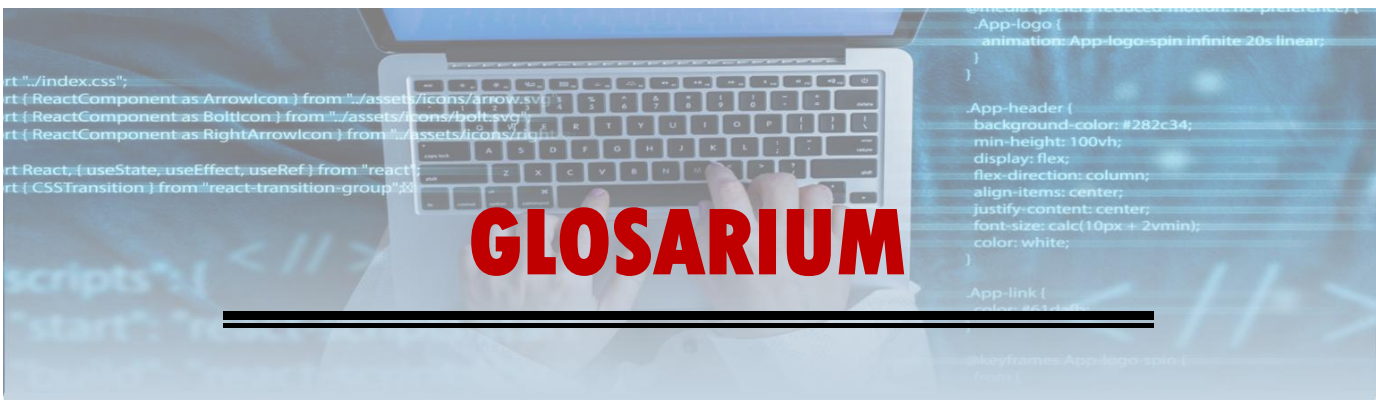
LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436–444.

Mallat, S. (2008). A Wavelet Tour of Signal Processing. Academic Press.

MathWorks. (2024). Why MATLAB?. Retrieved from https://www.mathworks.com/discovery/matlab.html

Nocedal, J., & Wright, S. J. (2006). Numerical Optimization (2nd ed.). Springer.

Oliphant, T. E. (2006). A Guide to NumPy. Trelgol Publishing.
Paszke, A., *et al*. (2019). PyTorch: An Imperative Style, High-Performance Deep Learning Library. Advances in Neural Information Processing Systems (NeurIPS).
Patricia, K., & Goldberg, D. (1991). What Every Computer Scientist Should Know About Floating-Point Arithmetic. ACM Computing Surveys.
Quarteroni, A., Sacco, R., & Saleri, F. (2007). Numerical Mathematics (2nd ed.). Springer.
Runge, C. (1901). Über empirische Funktionen und die Interpolation zwischen äquidistanten Ordinaten. Bishop, C. M. (2006). Pattern Recognition and *Machine Learning*. Springer.
Schwarz, S., Trefethen, L. N., & Higham, N. (2018). Numerical Computing with IEEE *Floating point* Arithmetic. SIAM.
Stroustrup, B. (2013). The C++ Programming Language (4th Ed.). Addison-Wesley.
Sundararajan, S. (2015). Fixed Point vs *Floating point* Representation. Embedded Systems Journal.
Xu, R., & Wunsch, D. (2005). Clustering. Wiley-IEEE Press.
Zaharia, M., *et al*. (2016). Apache Spark: A Unified Engine for Big Data Processing. Communications of the ACM, 59(11), 56–65.Zeitschrift für Mathematik und Physik.



GLOSARIUM

Bit Unit terkecil dalam representasi data digital, bernilai 0 atau 1, yang menjadi dasar dalam operasi logika komputer dan komunikasi biner.

Bug Kesalahan dalam penulisan kode program yang menyebabkan gangguan fungsi atau hasil yang tidak sesuai dengan yang diharapkan.

Loop Struktur kontrol dalam program yang memungkinkan pengulangan perintah selama kondisi tertentu masih terpenuhi.

Flag Variabel khusus yang digunakan untuk menandai status tertentu dalam eksekusi program, biasanya berupa nilai logika.

Code Kumpulan instruksi tertulis dalam bahasa pemrograman yang dapat diterjemahkan oleh mesin untuk menjalankan tugas tertentu.

Byte Unit data yang terdiri dari 8 bit, digunakan untuk menyimpan satu karakter atau nilai kecil dalam memori komputer.

Plot Representasi visual dari data numerik atau fungsi matematis dalam bentuk grafik untuk tujuan analisis dan interpretasi.

Hash Teknik konversi data menjadi nilai unik tetap menggunakan fungsi matematika, sering digunakan dalam pencarian cepat dan keamanan data.

Bool	Tipe data logika yang hanya memiliki dua nilai, yaitu benar (true) dan salah (false), esensial dalam pengambilan keputusan.
Char	Tipe data primitif yang merepresentasikan satu karakter, seperti huruf, angka, atau simbol dalam sistem pengkodean.
Read	Operasi untuk mengambil atau memperoleh data dari sumber luar seperti file, sensor, atau perangkat masukan.
Mean	Nilai rata-rata dari sekumpulan angka, diperoleh dengan menjumlahkan semua nilai dan membaginya dengan jumlah data.
Scan	Proses membaca setiap elemen data atau struktur dengan urutan tertentu untuk tujuan evaluasi atau pencarian.
Call	Instruksi untuk memanggil fungsi atau prosedur tertentu dalam program agar menjalankan serangkaian perintah tertentu.
Heap	Struktur data berbasis pohon biner yang digunakan dalam pengelolaan memori dan pengurutan prioritas.

INDEKS

BIOGRAFI PENULIS

Zunaida Sitorus, S.Si., M.Si.

Lahir di Kisaran, 9 Juni 1982, Lulus S2 di Program Studi Matematika Fakultas Matematika dan Ilmu Pengetahuan Alam Universitas Sumatera Utara Tahun 2010. Saat ini sebagai Dosen di Universitas Asahan Program Studi Teknik Informatika.

BUKU REFERENSI

PEMROGRAMAN DAN KOMPUTASI NUMERIK

DARI TEORI KE APLIKASI

Buku referensi "Pemrograman dan Komputasi Numerik: Dari Teori ke Aplikasi" ini membahas teknik-teknik numerik dalam pemrograman modern. Ditujukan bagi mahasiswa, dosen, peneliti, dan praktisi di bidang teknik, sains, maupun ilmu komputer, buku referensi ini menggabungkan landasan teoritis yang kuat dengan pendekatan aplikatif berbasis bahasa pemrograman. Buku referensi ini membahas berbagai konsep penting dalam komputasi numerik seperti metode interpolasi, integrasi numerik, penyelesaian sistem persamaan linier, dan solusi persamaan diferensial, dengan penekanan pada keakuratan, efisiensi, dan stabilitas numerik. Buku referensi ini membahas penjabaran algoritma, pseudocode, serta penerapan nyata menggunakan Python dan MATLAB sebagai alat bantu utama.

PT MEDIA PENERBIT INDONESIA

mediapenerbitindonesia.com
+6281362150605
Penerbit Idn
@pt.mediapenerbitidn

ISBN 978-634-7305-84-8
9 786347 305848