Bab 2

Deret Taylor dan Analisis Galat

Matematik selalu memperlihatkan rasa ingin tahu untuk dapat diterapkan di alam, dan ini dapat mengungkapkan kaitan yang dalam antara pikiran kita dan alam. Kita membicarakan semesta, bagian dari alam. Jadi, tidak mengherankan bahwa sistem logik dan matematika kita bernyanyi seirama dengan alam.
(George Zebrowski)

Columbus menemukan Amerika melalui kesalahan.
(Majalah Intisari)

Prasyarat yang diperlukan untuk mempelajari metode numerik adalah matematika. Matematika adalah ilmu dasar, jadi anda diharapkan sudah memiliki pengetahuan mengenai konsep fungsi, geometri, konsep kalkulus seperti turunan dan integral, dan sebagainya. Tidak paham terlalu dalam tidak apa, yang penting anda mengerti.

Banyak teorema matematika yang dipakai di sini. Dari sekian banyak teorema tersebut, ada satu teorema yang menjadi **kakas** (*tools*) yang sangat penting dalam metode numerik, yaitu teorema **deret Taylor**. Deret Taylor adalah kakas yang utama untuk menurunkan suatu metode numerik.

Pada bagian yang lain, kita akan membahas konsep galat. Seperti sudah dijelaskan di dalam Bab 1, solusi yang diperoleh secara numerik adalah nilai hampiran dari solusi sejati. Ini berarti terdapat galat (*error*) pada solusi hampiran tersebut. Bab 2 ini akan menjelaskan konsep galat, cara mengukur galat, penyebab galat, perambatan galat, dan ketidakstabilan perhitungan akibat galat.

2.1 Deret Taylor

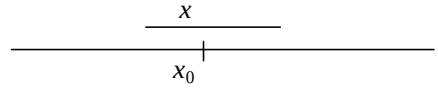
Kebanyakan dari metode-metode numerik yang diturunkan didasarkan pada penghampiran fungsi ke dalam bentuk polinom. Fungsi yang bentuknya kompleks menjadi lebih sederhana bila dihampiri dengan polinom, karena polinom merupakan bentuk fungsi yang paling mudah dipahami kelakuannya.

Kalau perhitungan dengan fungsi yang sesungguhnya menghasilkan solusi sejati, maka perhitungan dengan fungsi hampiran menghasilkan solusi hampiran. Pada bab 1 sudah dikatakan bahwa solusi numerik merupakan pendekatan (hampiran) terhadap solusi sejati, sehingga terdapat galat sebesar selisih antara solusi sejati dengan solusi hampiran. Galat pada solusi numerik harus dihubungkan dengan seberapa teliti polinom menghampiri fungsi sebenarnya. Kakas yang digunakan untuk membuat polinom hampiran adalah **deret Taylor**.

Definisi Deret Taylor

Andaikan f dan semua turunannya, f', f'', f''', ..., menerus di dalam selang $[a, b]$. Misalkan $x_0 \in [a, b]$, maka untuk nilai-nilai x di sekitar x_0 (Gambar 2.1) dan $x \in [a, b]$, $f(x)$ dapat diperluas (diekspansi) ke dalam deret Taylor:

$$f(x) = f(x_0) + \frac{(x-x_0)}{1!} f'(x_0) + \frac{(x-x_0)^2}{2!} f''(x_0) + \ldots + \frac{(x-x_0)^m}{m!} f^{(m)}(x_0) + \ldots \qquad \text{(P.2.1)}$$



Gambar 2.1. Nilai-nilai x di sekitar x_0

Persamaan (P.2.1) merupakan penjumlahan dari suku-suku (*term*), yang disebut **deret**. Perhatikanlah bahwa deret Taylor ini panjangnya tidak berhingga sehingga untuk memudahkan penulisan suku-suku selanjutnya kita menggunakan tanda elipsis (….). Jika dimisalkan $x - x_0 = h$, maka $f(x)$ dapat juga ditulis sebagai

$$f(x) = f(x_0) + \frac{h}{1!} f'(x_0) + \frac{h^2}{2!} f''(x_0) + \frac{h^3}{3!} f'''(x_0) + \ldots \frac{h}{m!} f^{(m)}(x_0) + \ldots \qquad \text{(P.2.2)}$$

Contoh 2.1

Hampiri fungsi $f(x) = \sin(x)$ ke dalam deret Taylor di sekitar $x_0 = 1$.

Penyelesaian:

Kita harus menentukan turunan sin(x) terlebih dahulu sebagai berikut

$f(x) = \sin(x)$,
$f'(x) = \cos(x)$,
$f''x) = -\sin(x)$,
$f'''(x) = -\cos(x)$,
$f^{(4)}(x) = \sin(x)$,
dan seterusnya.

Maka, berdasarkan (P.2.1), sin(x) dihampiri dengan deret Taylor sebagai berikut:

$$\sin(x) = \sin(1) + \frac{(x-1)}{1!}\cos(1) + \frac{(x-1)^2}{2!}(-\sin(1)) + \frac{(x-1)^3}{3!}(-\cos(1)) + \frac{(x-1)^4}{4!}\sin(1) + ...$$

Bila dimisalkan $x - 1 = h$, maka, berdasarkan (P.2.2),

$$\sin(x) = \sin(1) + h\cos(1) - \frac{h^2}{2}\sin(1) - \frac{h^3}{6}\cos(1) + \frac{h^4}{24}\sin(1) + ...$$

$$= 0.8415 + 0.5403h - 0.4208h^2 - 0.0901h^3 + 0.0351h^4 + ...$$

■

Kasus khusus adalah bila fungsi diperluas di sekitar $x_0 = 0$, maka deretnya dinamakan **deret Maclaurin**, yang merupakan deret Taylor baku. Kasus $x_0 = 0$ paling sering muncul dalam praktek.

Contoh 2.2

Uraikan sin(x), e^x, cos(x), dan ln(x +1) masing-masing ke dalam deret Maclaurin.

Penyelesaian:

Beberapa turunan sin(x) sudah dihitung pada Contoh 2.1. Deret Maclaurin dari sin(x) adalah

$$\sin(x) = \sin(0) + \frac{(x-0)}{1!}\cos(0) + \frac{(x-0)^2}{2!}(-\sin(0)) + \frac{(x-0)^3}{3!}(-\cos(0)) + \frac{(x-0)^4}{4!}\sin(0) + ...$$

$$= x - \frac{x^3}{3!} + \frac{x^5}{5!} - ...$$

Untuk menentukan deret Maclaurin dari e^x, kita harus menentukan turunan e^x terlebih dahulu sebagai berikut: $f(x) = e^x$, $f'(x) = e^x$, $f''(x) = e^x$, $f'''(x) = e^x$, $f^{(4)}(x) = e^x$, dan seterusnya. Deret Maclaurin dari e^x adalah

$$e^x = e^{(0)} + \frac{(x-0)}{1!}e^{(0)} + \frac{(x-0)^2}{2!}e^{(0)} + \frac{(x-0)^3}{3!}e^{(0)} + \frac{(x-0)^4}{4!}e^{(0)} + \ldots$$

$$= 1 + x + \frac{x^2}{2!} + \frac{x^3}{3!} + \frac{x^4}{4!} + \ldots$$

Untuk menentukan deret Maclaurin dari cos(x), kita harus menentukan turunan cos(x) terlebih dahulu sebagai berikut: $f(x) = \cos(x)$, $f'(x) = -\sin(x)$, $f''(x) = -\cos(x)$, $f'''(x) = \sin(x)$, $f^{(4)}(x) = \cos(x)$, dan seterusnya. Deret Maclaurin dari cos(x) adalah

$$= 1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!} + \ldots$$

Untuk menentukan deret Maclaurin dari ln(x + 1), kita harus menentukan turunan ln(x + 1) terlebih dahulu sebagai berikut: $f(x) = \ln(x+1)$, $f'(x) = (x+1)^{-1}$, $f''(x) = -(x+1)^{-2}$, $f'''(x) = 2(x+1)^{-3}$, $f^{(4)}(x) = -6(x+1)^{-4}$, dan seterusnya. Deret Maclaurin dari ln(x + 1) adalah

$$\ln(x+1) = \ln(0+1) + \frac{(x-0)}{1!}(0+1)^{-1} + \frac{(x-0)^2}{2!}(-(0+1)^{-2}) + \frac{(x-0)^3}{3!}2(0+1)^{-3} + \frac{(x-0)^4}{4!}(-6(0+1)^{-4}) + \ldots$$

$$= x - \frac{x^2}{2} + \frac{x^3}{3} - \frac{x^4}{4} + \ldots \quad \blacksquare$$

Karena suku-suku deret Taylor tidak berhingga banyaknya, maka -untuk alasan praktis- deret Taylor dipotong sampai suku orde tertentu. Deret Taylor yang dipotong sampai suku orde ke-n dinamakan **deret Taylor terpotong** dan dinyatakan oleh:

$$f(x) \approx f(x_0) + \frac{(x-x_0)}{1!}f'(x_0) + \frac{(x-x_0)^2}{2!}f''(x_0) + \ldots + \frac{(x-x_0)^n}{n!}f^{(n)}(x_0) + R_n(x) \qquad \text{(P.2.3)}$$

yang dalam hal ini,

$$R_n(x) = \frac{(x-x_0)^{(n+1)}}{(n+1)!}f^{(n+1)}(c) \qquad , x_0 < c < x \qquad \text{(P.2.4)}$$

disebut **galat** atau **sisa** (residu).

Dengan demikian deret Taylor yang dipotong sampai suku orde ke-*n* dapat ditulis sebagai

$$f(x) = P_n(x) + R_n(x) \qquad \text{(P.2.5)}$$

yang dalam hal ini,

$$P_n(x) = \sum_{k=1}^{n} \frac{(x - x_0)^k}{k!} f^{(k)}(x_0) \qquad \text{(P.2.6)}$$

$$R_n(x) = \frac{(x - x_0)^{(n+1)}}{(n+1)!} f^{(n+1)}(c) \qquad , x_0 < c < x \qquad \text{(P.2.7)}$$

Sebagai contoh, sin(*x*) pada Contoh 2.1 jika dihampiri dengan deret Taylor orde 4 di sekitar $x_0 = 1$ adalah:

$$\sin(x) = \sin(1) + \frac{(x-1)}{1!}\cos(1) - \frac{(x-1)^2}{2!}\sin(1) - \frac{(x-1)^3}{3!}\cos(1) + \frac{(x-1)^4}{4!}\sin(1) + R_4(x)$$

yang dalam hal ini,

$$R_4(x) = \frac{(x-1)^5}{5!}\cos(c) \qquad , 1 < c < x$$

Deret Taylor terpotong di sekitar $x_0 = 0$ disebut deret Maclaurin terpotong. Berdasarkan Contoh 2.2, deret MacLaurin terpotong untuk sin(*x*), e^x, cos(*x*), dan ln(*x* +1) adalah

$\sin(x) = x - x^3/3! + x^5/5! + R_5(x)$; $R_5(x) = \frac{x^6}{6!}\cos(c)$ (sampai suku orde 5)

$e^x = 1 + x + x^2/2! + x^3/3! + x^4/4! + R_4(x)$; $R_4(x) = \frac{x^5}{5!}e^c$ (sampai suku orde 4)

$\cos(x) = 1 - x^2/4! + x^4/6! - x^6/6! + R_6(x)$; $R_6(x) = \frac{x^7}{7!}\cos(c)$ (sampai suku orde 6)

$\ln(x+1) = x - x^2/2 + x^3/3 - x^4/4$; $R_4(x) = 24\frac{x^5}{5!}(c+1)^{-5}$ (sampai suku orde 4)

yang dalam hal ini, $0 < c < x$.

Fungsi pustaka matematika di dalam kalkulator dan *compiler* bahasa pemrograman dapat menggunakan deret Taylor untuk mengevaluasi nilai-nilai fungsi trigonometri dan fungsi transenden yang tidak dapat dihitung secara langsung.

Contoh 2.3

Hitunglah hampiran nilai cos(0.2), sudut dinyatakan dalam radian, dengan deret Maclaurin sampai suku orde $n = 6$.

Penyelesaian:

Dari hasil pada Contoh 2.2,

$$\cos(0.2) \approx 1 - 0.2^2/2 + 0.2^4/24 - 0.2^6/720 = 0.9800667$$

(sampai 7 angka di belakang koma) ■

Program 2.1 di bawah ini menghitung nilai hampiran cos(x). Setiap suku kita hitung nilainya, jika nilainya lebih kecil dari toleransi (epsilon) yang kita spesifikasikan, maka perhitungan suku berikutnya dapat dihentikan.

Program 2.1 Program menghitung hampiran nilai cos(x)

```
function cos(x:real):real;
{mengembalikan nilai cosinus sampai nilai suatu suku < e }
const
  epsilon = 0.00000001;   { toleransi nilai tiap suku }
  { jika nilai suku sudah lebih kecil dari epsilon, perhitungan cos dapat
     dihentikan }
var
  tanda, n     : integer;
  suku, jumlah : real;

  (* function pangkat(x:real; n:integer):real;
    { Menghitung nilai x^n }    *)

  (* function faktorial(n:integer):integer;
     { Menghitung n! }        *)

begin
   suku:=1;    {suku pertama deret cosinus}
   tanda:=1;   {tanda (+/-) suku pertama}
   n:=0;       {orde suku pertama}
   jumlah:=0;  {jumlah deret cosinus, inisialisasi dengan 0}
   while abs(suku) >= epsilon do
    begin
       jumlah:=jumlah + suku; {jumlah deret cosinus}
       n:=n+2;                {orde suku berikutnya}
       tanda:=-tanda;         {tanda suku berikutnya}
```

```
        suku:=tanda*pangkat(x,n)/faktorial(n);
     end;
    { abs(suku) < epsilon }
    cos:=jumlah;
end;
```

Deret Taylor banyak digunakan untuk menurunkan metode-metode numerik. Deret Taylor yang terpotong digunakan sebagai titik awal dalam menurunkan metode. Anda sebaiknya dapat menguasai deret Taylor terlebih dahulu sebagai alat bantu yang penting dalam metode numerik.

2.2 Analisis Galat

Menganalisis galat sangat penting di dalam perhitungan yang menggunakan metode numerik. Galat berasosiasi dengan seberapa dekat solusi hampiran terhadap solusi sejatinya. Semakin kecil galatnya, semakin teliti solusi numerik yang didapatkan. Kita harus memahami dua hal: (a) bagaimana menghitung galat, dan (b) bagaimana galat timbul.

Misalkan $\hat{a}$ adalah nilai hampiran terhadap nilai sejati a, maka selisih

$$e = a - \hat{a} \tag{P.2.8}$$

disebut **galat**. Sebagai contoh, jika $\hat{a}$ = 10.5 adalah nilai hampiran dari a = 10.45, maka galatnya adalah e = -0.01. Jika tanda galat (positif atai negatif) tidak dipertimbangkan, maka **galat mutlak** dapat didefenisikan sebagai

$$|e| = |a - \hat{a}| \tag{P.2.9}$$

Sayangnya, ukuran galat e kurang bermakna sebab ia tidak menceritakan seberapa besar galat itu dibandingkan dengan nilai sejatinya. Sebagai contoh, seorang anak melaporkan panjang sebatang kawat 99 cm, padahal panjang sebenarnya 100 cm. Galatnya adalah 100 - 99 = 1 cm. Anak yang lain melaporkan panjang sebatang pensil 9 cm, padahal panjang sebenarnya 10 cm, sehingga galatnya juga 1 cm. Kedua galat pengukuran sama-sama bernilai 1 cm, namun galat 1 cm pada pengukuran panjang pensil lebih berarti daripada galat 1 cm pada pengukuran panjang kawat. Jika tidak ada informasi mengenai panjang sesungguhnya, kita mungkin menganggap kedua galat tersebut sama saja. Untuk mengatasi interpretasi nilai galat ini, maka galat harus *dinormalkan* terhadap nilai sejatinya. Gagasan ini melahirkan apa yang dinamakan **galat relatif.**

Galat relatif didefinisikan sebagai

$$e_R = \frac{e}{a} \qquad \text{(P.2.10)}$$

atau dalam persentase

$$e_R = \frac{e}{a} \times 100\% \qquad \text{(P.2.11)}$$

Karena galat dinormalkan terhadap nilai sejati, maka galat relatif tersebut dinamakan juga **galat relatif sejati**. Dengan demikian, pengukuran panjang kawat mempunyai galat relatif sejati = 1/100 = 0.01, sedangkan pengukuran panjang pensil mempunyai galat relatif sejati = 1/10 = 0.1.

Dalam praktek kita tidak mengetahui nilai sejati a, karena itu galat e seringkali dinormalkan terhadap solusi hampirannya, sehingga galat relatifnya dinamakan **galat relatif hampiran**:

$$e_{RA} = \frac{e}{\hat{a}} \qquad \text{(P.2.12)}$$

Contoh 2.4

Misalkan nilai sejati = 10/3 dan nilai hampiran = 3.333. Hitunglah galat, galat mutlak, galat relatif, dan galat relatif hampiran.

Penyelesaian:

galat = 10/3 – 3.333 = 10/3 – 3333/1000 = 1/3000 = 0.000333…
galat mutlak = | 0.000333…| = 0.000333…
galat relatif = (1/3000)/(10/3) = 1/1000 = 0.0001
galat relatif hampiran = (1/3000)/3.333 = 1/9999 ■

Galat relatif hampiran yang dihitung dengan persamaan (P.2.12) masih mengandung kelemahan sebab nilai e tetap membutuhkan pengetahuan nilai a (dalam praktek kita jarang sekali mengetahui nilai sejati a). Oleh karena itu, perhitungan galat relatif hampiran menggunakan pendekatan lain. Pada perhitungan numerik yang menggunakan pendekatan lelaran (*iteration*), e_{RA} dihitung dengan cara

$$e_{RA} = \frac{a_{r+1} - a_r}{a_{r+1}} \qquad \text{(P.2.13)}$$

yang dalam hal ini a_{r+1} adalah nilai hampiran lelaran sekarang dan a_r adalah nilai hampiran lelaran sebelumnya. Proses lelaran dihentikan bila

$$|e_{RA}| < e_S$$

yang dalam hal ini e_S adalah toleransi galat yang dispesifikasikan. Nilai e_S menentukan ketelitian solusi numerik. Semakin kecil nilai e_S, semakin teliti solusinya, namun semakin banyak proses lelarannya. Contoh 2.5 mengilustrasikan hal ini.

Contoh 2.5

Misalkan ada prosedur lelaran sebagai berikut

$$x_{r+1} = (-x_r^3 + 3)/6 , \qquad r = 0, 1, 2, 3, \ldots .$$

Lelaran dihentikan bila kondisi $|e_{RA}| < e_S$, dalam hal ini e_S adalah toleransi galat yang diinginkan. Misalkan dengan memberikan $x_0 = 0.5$, dan $e_S = 0.00001$ kita memperoleh runtunan:

$x_0 = 0.5$
$x_1 = 0.4791667$; $|e_{RA} = (x_1 - x_0)/x_1| = 0.043478 > e_S$
$x_2 = 0.4816638$; $|e_{RA} = (x_2 - x_1)/x_2| = 0.0051843 > e_S$
$x_3 = 0.4813757$; $|e_{RA} = (x_3 - x_2)/x_3| = 0.0005984 > e_S$
$x_4 = 0.4814091$; $|e_{RA} = (x_4 - x_3)/x_4| = 0.0000693 > e_S$
$x_5 = 0.4814052.$; $|e_{RA} = (x_5 - x_4)/x_5| = 0.0000081 < e_S$, berhenti!

Pada lelaran ke-5, $|e_{RA}| < e_S$ sudah terpenuhi sehingga lelaran dapat dihentikan. ■

2.3 Sumber Utama Galat Numerik

Secara umum terdapat dua sumber utama penyebab galat dalam perhitungan numerik:

1. **Galat pemotongan** (*truncation error*)
2. **Galat pembulatan** (*round-off error*)

Selain kedua galat ini, masih ada sumber galat lain, antara lain [KRE88]:

a. **Galat eksperimental**, yaitu galat yang timbul dari data yang diberikan, misalnya karena kesalahan pengukuran, ketidaktelitian alat ukur, dan sebagainya
b. **Galat pemrograman**. Galat yang terdapat di dalam program sering dinamakan dengan kutu (*bug*), dan proses penghilangan galat ini dinamakan penirkutuan (*debugging*).

Kita tidak akan membicarakan kedua galat terakhir ini karena kontribusinya terhadap galat keseluruhan tidak selalu ada. Akan halnya galat utama, galat pemotongan dan galat pembulatan, keduanya selalu muncul pada solusi numerik. Terhadap kedua galat inilah perhatian kita fokuskan.

2.3.1 Galat Pemotongan

Galat pemotongan mengacu pada galat yang ditimbulkan akibat penggunaan hampiran sebagai pengganti formula eksak. Maksudnya, ekspresi matematik yang lebih kompleks "diganti" dengan formula yang lebih sederhana. Tipe galat pemotongan bergantung pada metode komputasi yang digunakan untuk penghampiran sehingga kadang-kadaang ia disebut juga **galat metode**. Misalnya, turunan pertama fungsi f di x_i dihampiri dengan formula

$$f'(x_i) \approx \frac{f(x_{i+1}) - f(x_i)}{h}$$

yang dalam hal ini h adalah lebar absis x_{i+1} dengan x_i. Galat yang ditimbulkan dari penghampiran turunan tersebut merupakan galat pemotongan.

Istilah "pemotongan" muncul karena banyak metode numerik yang diperoleh dengan penghampiran fungsi menggunakan deret Taylor. Karena deret Taylor merupakan deret yang tak-berhingga, maka untuk penghampiran tersebut deret Taylor kita hentikan/potong sampai suku orde tertentu saja. Penghentian suatu deret atau runtunan langkah-langkah komputasi yang tidak berhingga menjadi runtunan langkah yang berhingga itulah yang menimbulkan galat pemotongan.

Contohnya, hampiran fungsi cos(x) dengan bantuan deret Taylor di sekitar $x = 0$:

$$\cos(x) \approx \underbrace{1 - \frac{x^2}{2!} + \frac{x^4}{4!} - \frac{x^6}{6!}}_{\text{nilai hampiran}} \Bigg| \underbrace{+ \frac{x^8}{8!} - \frac{x^{10}}{10!} + \ldots}_{\text{galat pemotongan}}$$

pemotongan

Deret Taylor fungsi cos(x) sebenarnya tidak berhingga, namun untuk keperluan praktis, deret tersebut kita potong sampai suku orde tertentu, misalnya sampai suku orde $n = 6$ seperti pada contoh di atas. Kita melihat bahwa menghampiri cos(x) dengan deret Taylor sampai suku berderajat enam tidak memberikan hasil yang tepat. Galat pada nilai hampiran diakibatkan oleh pemotongan suku-suku deret. Jumlah suku-suku selanjutnya setelah pemotongan merupakan galat pemtongan untuk cos(x). Kita tidak dapat menghitung berapa persisnya galat pemtongan ini karena jumlah seluruh suku-suku setelah pemotongan tidak

mungkin dapat dihitung. Namun, kita dapat menghampiri galat pemotongan ini dengan rumus suku sisa:

$$R_n(x) = \frac{(x-x_0)^{(n+1)}}{(n+1)!} f^{(n+1)}(c) \qquad , x_0 < c < x$$

Pada contoh cos(x) di atas,

$$R_6(x) = \frac{x^7}{7!}\cos(c) \qquad , 0 < c < x$$

Nilai R_n yang tepat hampir tidak pernah dapat kita peroleh, karena kita tidak mengetahui nilai c sebenarnya terkecuali informasi bahwa c terletak pada suatu selang tertentu. Karenanya tugas kita adalah mencari nilai maksimum yang mungkin dari $|R_n|$ untuk c dalam selang yang diberikan itu [PUR84], yaitu:

$$|R_n(x)| < \max_{x_0<c<x} \left| f^{(n+1)}(c) \right| \times \frac{(x-x_0)^{n+1}}{n+1!}$$

Contoh komputasi lain yang menghasilkan galat pemotongan adalah perhitungan dengan menggunakan skema lelaran (lihat Contoh 2.5). Sayangnya, tidak seperti deret Taylor, galat pemotongan pada perhitungan dengan skema lelaran tidak ada rumusnya.

Galat pemotongan pada deret Taylor dapat dikurangi dengan meningkatkan orde suku-sukunya, namun jumlah komputasinya menjadi lebih banyak. Pada metode yang menerapkan skema lelaran, galat pemotongan dapat dikurangi dengan memperbanyak lelaran. Hal ini ditunjukkan pada Contoh 2.5 dengan memberikan nilai e_S yang sekecil mungkin

Contoh 2.6

[PUR89] Gunakan deret Taylor orde 4 di sekitar $x_0 = 1$ untuk menghampiri ln(0.9) dan berikan taksiran untuk galat pemotongan maksimum yang dibuat.

Penyelesaian:

Tentukan turunan fungsi $f(x) = \ln(x)$ terlebih dahulu

$f(x) = \ln(x)$ $\rightarrow f(1)=0$
$f'(x) = 1/x$ $\rightarrow f'(1)=1$
$f''(x) = -1/x^2$ $\rightarrow f''(1) = -1$
$f'''(x) = 2/x^3$ $\rightarrow f'''(1) = 2$

$f^{(4)}(x) = -6/x^4 \quad \rightarrow f^{(4)}(1) = -6$
$f^{(5)}(x) = 24/x^5 \quad \rightarrow f^{(5)}(c) = 24/c^5$

Deret Taylornya adalah

$$\ln(x) = (x - 1) - (x - 1)^2/2 + (x - 1)^3/3 - (x - 1)^4/4 + R_4(x)$$

dan

$$\ln(0.9) = -01. - (-0.1)^2/2 + (-0.1)^3/3 - (-0.1)^4/4 + R_4(x) = -0.1053583 + R_4(x)$$

juga

$$|R_5(0.9)| < \max_{0.9<c<1} \left|\frac{24}{c^5}\right| \times \frac{(-0.1)^5}{5!}$$

dan nilai Max $|24/c^5|$ di dalam selang $0.9 < c < 1$ adalah pada $c = 0.9$ (dengan mendasari pada fakta bahwa suatu pecahan nilainya semakin membesar bilamana penyebut dibuat lebih kecil), sehingga

$$|R_4(0.9)| < \max_{0.9<c<1} \left|\frac{24}{0.9^5}\right| \times \frac{(-0.1)^5}{5!} \approx 0.0000034$$

Jadi ln(0.9) = -0.1053583 dengan galat pemotongan lebih kecil dari 0.0000034. ■

Contoh 2.7

Deret Taylor dapat digunakan untuk menghitung integral fungsi yang sulit diintegralkan secara analitik (bahkan, adakalanya tidak mungkin dihitung secara analitik). Hitunglah hampiran nilai $\int_0^1 e^{x^2} dx$ secara numerik, yaitu fungsi $f(x) = e^{x^2}$ dihampiri dengan deret Maclaurin orde 8.

Penyelesaian:

Deret Maclaurin orde 8 dari fungsi $f(x) = e^{x^2}$ adalah

$e^{x^2} = 1 + x^2 + x^4/2! + x^6/3! + x^8/4!$ (silakan memeriksanya kembali sebagai latihan)

Dengan demikian, maka

$$\int_0^1 e^{x^2} dx \approx \int_0^1 (1 + x^2 + \frac{x^4}{2!} + \frac{x^6}{3!} + \frac{x^8}{4!}) dx = x + \frac{x^3}{3} + \frac{x^5}{10} + \frac{x^7}{42} + \frac{x^9}{216} \Big|_{x=0}^{x=1}$$

$$= 1 + \frac{1}{3} + \frac{1}{10} + \frac{1}{42} + \frac{1}{216} = \frac{397836}{272160} = 1.4617724$$ ■

2.3.2 Galat Pembulatan

Perhitungan dengan metode numerik hampir selalu menggunakan bilangan riil. Masalah timbul bila komputasi numerik dikerjakan oleh mesin (dalam hal ini komputer) karena semua bilangan riil tidak dapat disajikan secara tepat di dalam komputer. Keterbatasan komputer dalam menyajikan bilangan riil menghasilkan galat yang disebut **galat pembulatan**. Sebagai contoh 1/6 = 0.166666666… tidak dapat dinyatakan secara tepat oleh komputer karena digit 6 panjangnya tidak terbatas. Komputer hanya mampu merepresentasikan sejumlah digit (atau bit dalam sistem biner) saja. Bilangan riil yang panjangnya melebihi jumlah digit (bit) yang dapat direpresentasikan oleh komputer dibulatkan ke bilangan terdekat.

Misalnya sebuah komputer hanya dapat merepresentasikan bilangan riil dalam 6 digit angka berarti, maka representasi bilangan 1/6 = 0.1666666666… di dalam komputer 6-digit tersebut adalah 0.166667. Galat pembulatannya adalah 1/6 – 0.166667 = -0.000000333. Contoh dalam sistem biner misalnya 1/10 = $0.0001100110011001100110011\ 00110011\ldots_2$ direpresentasikan di dalam komputer dalam jumlah bit yang terbatas. Teknik yang digunakan untuk pembulatan bilangan riil dijelaskan di dalam upabab 2.5.3.

Kebanyakan komputer digital mempunyai dua buah cara penyajian bilangan riil, yaitu **bilangan titik-tetap** (*fixed point*) dan bilangan **titik-kambang** (*floating point*) [KRE88]. Dalam format bilangan titik-tetap setiap bilangan disajikan dengan jumlah tempat desimal yang tetap, misalnya 62.358, 0.013, 1.000. Sedangkan dalam format bilangan titik-kambang setiap bilangan disajikan dengan jumlah digit *berarti* yang sudah tetap, misalnya

0.6238×10^{3} 0.1714×10^{-13}

atau ditulis juga

0.6238E+03 0.1714E-13

Digit-digit berarti di dalam format bilangan titik-kambang disebut juga **angka bena** (*significant figure*). Konsep angka bena dijelaskan berikut ini.

Angka Bena

Konsep angka bena (*significant figure*) atau angka berarti telah dikembangkan secara formal untuk menandakan keandalan suatu nilai numerik. Angka bena adalah angka bermakna, angka penting, atau angka yang dapat digunakan dengan pasti [CHA91].

Contohnya,

43.123	memiliki 5 angka bena (yaitu 4, 3, 1, 2, 3)
0.1764	memiliki 4 angka bena (yaitu 1, 7, 6, 4)
0.0000012	memiliki 2 angka bena (yaitu 1, 2)
278.300	memiliki 6 angka bena (yaitu 2, 7, 8, 3, 0, 0)
270.0090	memiliki 7 angka bena (yaitu 2, 7, 0, 0, 0, 9, 0)
0.0090	memiliki 2 angka bena (yaitu 9, 0)
1360, 1.360, 0.001360 semuanya memiliki 4 angka bena	

Perhatikanlah bahwa angka 0 bisa menjadi angka bena atau bukan. Pada contoh 0.001360, tiga buah angka nol pertama tidak berarti, sedangkan 0 yang terakhir angka berarti karena pengukuran dilakukan sampai ketelitian 4 digit. Jumlah angka bena akan terlihat dengan pasti bila bilangan riil itu ditulis dalam penulisan ilmiah (*scientific notation*), misalnya tetapan dalam kimia dan fisika atau ukuran jarak dalam astronomi. Jumlah angka bena terletak pada jumlah digit *mantis*-nya (tentang mantis ini akan dibahas belakangan):

4.3123×10^{1}	memiliki 5 angka bena
1.764×10^{-1}	memiliki 4 angka bena
1.2×10^{-6}	memiliki 2 angka bena
2.78300×10^{2}	memiliki 6 angka bena
0.2700090×10^{3}	memiliki 7 angka bena
9.0×10^{-3}	memiliki 2 angka bena
13.60×10^{2} , 0.1360×10^{1} , 1.360×10^{-3}	memiliki 4 angka bena
6.02×10^{23}	memiliki 24 angka bena (bilangan *Avogadro*)
1.5×10^{7}	memiliki 8 angka bena (jarak bumi-matahari)

Komputer hanya menyimpan sejumlah tertentu angka bena. Bilangan riil yang jumlah angka benanya melebihi jumlah angka bena komputer akan disimpan dalam sejumlah angka bena komputer itu. Pengabaian angka bena sisanya itulah yang menimbulkan galat pembulatan.

2.3.3 Galat Total

Galat akhir atau galat total atau pada solusi numerik merupakan jumlah galat pemotongan dan galat pembulatan. Misalnya pada Contoh 2.3 kita menggunakan deret Maclaurin orde-4 untuk menghampiri cos(0.2) sebagai berikut:

$$\cos(0.2) \approx 1 - 0.2^2/2 + 0.2^4/24 \approx 0.9800667$$

galat pemotongan: $1 - 0.2^2/2 + 0.2^4/24$

galat pembulatan: $1 - 0.2^2/2 + 0.2^4/24 \approx 0.9800667$

Galat pemotongan timbul karena kita menghampiri cos(0.2) sampai suku orde empat, sedangkan galat pembulatan timbul karena kita membulatkan nilai hampiran ke dalam 7 digit bena.

2.4 Orde Penghampiran

Di dalam metode numerik, fungsi $f(x)$ sering diganti dengan fungsi hampiran yang lebih sederhana. Satu cara mengungkapkan tingkat ketelitian penghampiran itu adalah dengan menggunakan notasi ***O*-Besar** (*Big-Oh*).

Misalkan $f(h)$ dihampiri dengan fungsi $p(h)$. Jika $|f(h) - p(h)| \le M|h^n|$, yang dalam hal ini M adalah konstanta riil > 0, maka kita katakan bahwa $p(h)$ menghampiri $f(h)$ dengan orde penghampiran $O(h^n)$ dan kita tulis

$$f(h) = p(h) + O(h^n) \qquad \text{(P.2.14)}$$

$O(h^n)$ juga dapat diartikan sebagai orde galat dari penghampiran fungsi. Karena h umumnya cukup kecil yaitu lebih kurang dari 1, maka semakin tinggi nilai n, semakin kecil galat, yang berarti semakin teliti penghampiran fungsinya. Metode yang berorde $O(h^2)$, misalnya, lebih teliti hasilnya daripada metode yang berorde $O(h)$. Juga, pada metode yang berorde $O(h^2)$, jika ukuran h dijadikan setengah kali semula, maka galatnya menjadi seperempat kali galat semula.

Umumnya deret Taylor digunakan untuk penghampiran fungsi. Misalkan,

$$x_{i+1} = x_i + h, \qquad i = 0, 1, 2, \ldots$$

adalah titik-titik selebar h, maka hampiran $f(x_{i+1})$ dengan deret Taylor di sekitar x_i adalah

$$f(x_{i+1}) = f(x_i) + \frac{(x_{i+1}-x_i)}{1!} f'(x_i) + \frac{(x_{i+1}-x_i)^2}{2!} f''(x_i) + \ldots + \frac{(x_{i+1}-x_i)^n}{n!} f^{(n)}(x_i) + R_n(x_{i+1})$$

$$= f(x_i) + \frac{h}{1!} f'(x_i) + \frac{h^2}{2!} f''(x_i) + \ldots + \frac{h^n}{n!} f^{(n)}(x_i) + R_n(x_{i+1})$$

yang dalam hal ini

$$R_n(x_{i+1}) = \frac{h^{n+1}}{(n+1)!} f^{(n+1)}(t) = O(h^{n+1}) \quad , x_i < t < x_{i+1} \tag{P.2.15}$$

Jadi, kita dapat menuliskan

$$f(x_{i+1}) = \sum_{k=0}^{n} \frac{h^k}{k!} f^{(k)}(x_i) + O(h^{n+1}) \tag{P.2.16}$$

Persamaan (P.2.16) menyatakan bahwa jika fungsi $f(x)$ dihampiri dengan deret Taylor derajat n, maka suku sisanya cukup dinyatakan dengan lambang $O(h^{n+1})$. Sebagai catatan, suku sisa yang digunakan di dalam notasi O-Besar adalah suku yang dimulai dengan perpangkatan h^{n+1}.

Sebagai contoh,

$e^h = 1 + h + h^2/2! + h^3/3! + h^4/4! + O(h^5)$

$\ln(x+1) = x - x^2/2 + x^3/3 - x^4/4 + x^5/4 + O(h^5)$

$\sin(h) = h - h^3/3! + h^5/5! + O(h^7)$ (bukan $O(h^6)$, karena suku orde 6 = 0)

$\cos(h) = 1 - h^2/4! + h^4/6! - h^6/6! + O(h^8)$ (bukan $O(h^7)$, karena suku orde 7 = 0)

2.5 Bilangan Titik-Kambang

Untuk memahami galat pembulatan lebih rinci, kita perlu mengerti cara penyimpanan bilangan riil di dalam komputer. Format bilangan riil di dalam komputer berbeda-beda bergantung pada piranti keras dan compiler bahasa pemrogramannya. Bilangan riil di dalam komputer umumnya disajikan dalam format *bilangan titik-kambang*. Bilangan titik-kambang a ditulis sebagai

$$a = \pm m \times B^p = \pm 0.d_1d_2d_3d_4d_5d_6 \ldots d_n \times B^p \tag{P.2.17}$$

yang dalam hal ini,

m = mantisa (riil), $d_1d_2d_3d_4d_5d_6 \ldots d_n$ adalah digit atau bit mantisa yang nilainya dari 0 sampai $B - 1$, n adalah panjang digit (bit) mantisa.
B = basis sistem bilangan yang dipakai (2, 8, 10, 16, dan sebagainya)
p = pangkat (berupa bilangan bulat), nilainya dari $-P_{\text{min}}$ sampai $+P_{\text{maks}}$

Sebagai contoh, bilangan riil 245.7654 dinyatakan sebagai 0.2457654×10^3 dalam format bilangan titik kambang dengan basis 10. Cara penyajian seperti itu serupa dengan cara penulisan ilmiah. Penulisan ilmiah termasuk ke dalam sistem bilangan titik-kambang.

Sistem bilangan yang kita gunakan setiap hari menggunakan basis sepuluh (disebut juga sistem desimal), $B = 10$. Umumnya komputer menggunakan sistem biner ($B = 2$), tapi beberapa komputer menggunakan basis 8 dan 16. Untuk memudahkan pemahaman –juga karena kita lebih terbiasa sehari-hari dengan bilangan desimal– kebanyakan contoh-contoh bilangan titik-kambang di dalam bab ini disajikan dalam sistem desimal.

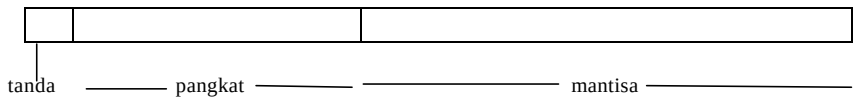
Bilangan titik-kambang di dalam sistem biner biner direpresentasikan oleh komputer dalam bentuk *word* seperti ditunjukkan pada Gambar 2.2. Bit pertama menyatakan tanda (+/-), deretan bit berikutnya menyatakan pangkat bertanda, dan deretan bit terakhir untuk mantisa.

Setiap komputer memiliki panjang *word* yang berbeda-beda. Pada komputer IBM PC, bilangan titik-kambang berketelitian tunggal (*single precission*) disajikan dalam 32 bit yang terdiri atas 1 bit sebagai tanda, 8 bit untuk pangkat dan 23 bit untuk mantisa. Jika dalam bentuk ternormalisasi (akan dijelaskan kemudian), maka bit pertama pada mantisa harus 1, sehingga jumlah bit mantisa efektif adalah 24:

$$a = \pm\, 0.1b_1b_2b_3b_4b_5b_6 \ldots b_{23} \times B^p$$

yang dalam hal ini b menyatakan bit biner (0 atau 1).

Sedangkan pada komputer IBM 370, bilangan titik-kambang berketelitian tunggal disajikan dalam 32 bit yang terdiri dari 1 bit tanda, 7 bit pangkat (basis 16), dan 24 bit mantis (setara dengan 6 sampai 7 digit desimal) [KRE88].



Gambar 2.2 Format bilangan titik-kambang biner (1 *word*)

2.5.1 Bilangan Titik-Kambang Ternormalisasi

Representasi bilangan titik-kambang (P.2.17) jauh dari unik, karena, sebagai contoh, kita juga dapat menulisnya sebagai

$$a = \pm (mb) \times B^{p-1} \qquad \text{(P.2.18)}$$

Misalnya, 245.7654 dapat ditulis sebagai

0.2457654×10^3 atau
2.457654×10^2 atau
0.02457654×10^4 , dan sebagainya

Agar bilangan titik-kambang dapat disajikan secara seragam, kebanyakan sistem komputer menormalisasikan formatnya sehingga semua digit mantisa selalu angka bena. Karena alasan itu, maka digit pertama mantisa tidak boleh nol. Bilangan titik-kambang yang dinormalisasi ditulis sebagai

$$a = \pm m \times B^p = \pm 0.d_1d_2d_3d_4d_5d_6 \ldots d_n \times B^p \qquad \text{(P.2.19)}$$

yang dalam hal ini, $d_1d_2d_3d_4d_5d_6 \ldots d_n$ adalah digit (atau bit) mantisa dengan syarat $1 \le d_1 \le b - 1$ dan $0 \le d_k \le B-1$ untuk $k > 1$. Pada sistem desimal,

$$1 \le d_1 \le 9 \text{ dan } 0 \le d_k \le 9,$$

sedangkan pada sistem biner,

$$d_1 = 1 \text{ dan } 0 \le d_k \le 1$$

Sebagai contoh, 0.0563×10^{-3} dinormalisasi menjadi 0.563×10^{-4}, 0.00023270×10^6 dinormalisasi menjadi 0.23270×10^3. Sebagai konsekuensi penormalan, nilai m adalah

$$1/B \le m < 1$$

Pada sistem desimal ($B = 10$), m akan berkisar dari 0.1 sampai 1, dan pada sistem biner ($B = 10$), antara 0.5 dan 1 [CHA91].

Sebagai catatan, nol adalah kasus khusus. Nol disajikan dengan bagian mantisa seluruhnya nol dan pangkatnya nol. Nol semacam ini tidak dinormalisasi.

Contoh 2.8

[BUC92] Tulislah bilangan *e* dalam format bilangan titik-kambang ternormalisasi dengan basis 10, basis 2, dan basis 16.

Penyelesaian:

Dalam basis 10 (menggunakan 8 angka bena),

$$e \approx 2.7182818 = 0.27182818 \times 10^1$$

(bilangan titik-kambang desimal ternormalisasi)

Dalam basis 2 (menggunakan 30 bit bena),

$$e \approx 0.10101101111110000101010001011_0 \times 2^2$$

(bilangan titik-kambang biner ternormalisasi)

Dalam basis 16 (gunakan fakta bahwa $16 = 2^4$, sehingga $2^2 = ¼ \times 16^1$

$$\begin{aligned} e &\approx 0.10101101111110000101010001011_0 \times 2^2 \\ &= ¼ \times 0.10101101111110000101010001011_0 \times 16^1 \\ &= 0.00101011011111100001010100010110_2 \times 16^1 \\ &= 0.2B7E1516_{16} \times 16^1 \end{aligned}$$

(bilangan titik-kambang heksadesimal ternormalisasi) ■

2.5.2 Epsilon Mesin

Karena jumlah bit yang digunakan untuk representasi bilangan titik-kambang terbatas, maka jumlah bilangan riil yang dapat direpresentasikan juga terbatas. Untuk ilustrasi, tinjau kasus bilangan titik-kambang biner 6-bit *word* (1 bit tanda, 3 bit untuk pangkat bertanda, dan 2 bit mantisa) dengan $B = 2$, dan nilai pangkat dari –2 sampai 3 [GER94]. Karena semua bilangan dinormalisasi, maka bit pertama harus 1, sehingga semua bilangan yang mungkin adalah berbentuk:

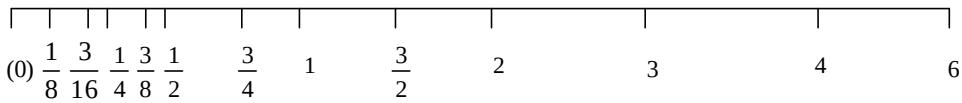
$$\pm 0.10_2 \times 2^p \quad \text{atau} \quad \pm 0.11_2 \times 2^p, \qquad -2 \le p \le 3$$

Daftar bilangan riil positif yang dapat direpresentasikan adalah

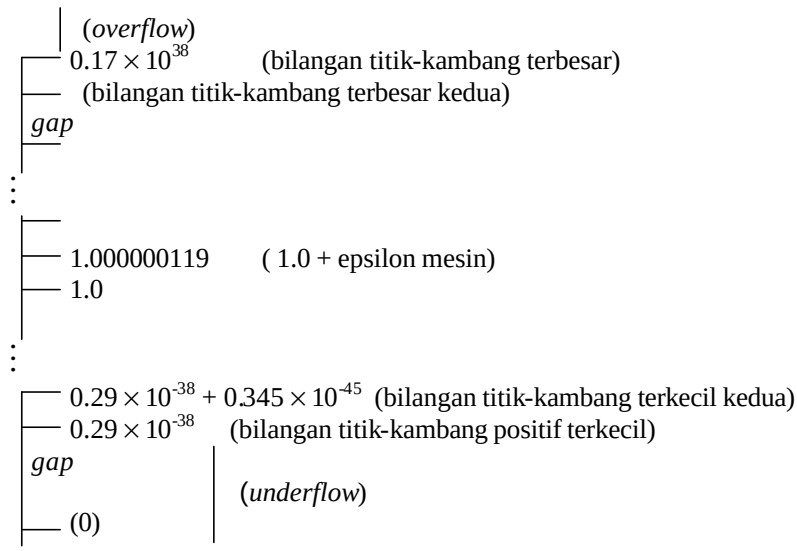
$$\begin{aligned} 0.10_2 \times 2^{-2} &= (1 \times 2^{-1} + 0 \times 2^{-2}) \times 2^{-2} = 1/8 = 0.125_{10} \\ 0.10_2 \times 2^{-1} &= (1 \times 2^{-1} + 0 \times 2^{-2}) \times 2^{-1} = 1/4 = 0.25_{10} \\ 0.10_2 \times 2^{0} &= (1 \times 2^{-1} + 0 \times 2^{-2}) \times 2^{0} = 1/2 = 0.5_{10} \\ 0.10_2 \times 2^{1} &= (1 \times 2^{-1} + 0 \times 2^{-2}) \times 2^{1} = 1 = 1.0_{10} \end{aligned}$$

$0.10_2 \times 2^2 = (1 \times 2^{-1} + 0 \times 2^{-2}) \times 2^2 = 2 = 2.0_{10}$
$0.10_2 \times 2^3 = (1 \times 2^{-1} + 0 \times 2^{-2}) \times 2^3 = 4 = 4.0_{10}$
$0.11_2 \times 2^{-2} = (1 \times 2^{-1} + 1 \times 2^{-2}) \times 2^{-2} = 3/16 = 0.1875_{10}$
$0.11_2 \times 2^{-1} = (1 \times 2^{-1} + 1 \times 2^{-2}) \times 2^{-1} = 3/8 = 0.375_{10}$
$0.11_2 \times 2^0 = (1 \times 2^{-1} + 1 \times 2^{-2}) \times 2^0 = 3/4 = 0.75_{10}$
$0.11_2 \times 2^1 = (1 \times 2^{-1} + 1 \times 2^{-2}) \times 2^1 = 3/2 = 1.5_{10}$
$0.11_2 \times 2^2 = (1 \times 2^{-1} + 1 \times 2^{-2}) \times 2^2 = 3 = 3.0_{10}$
$0.11_2 \times 2^3 = (1 \times 2^{-1} + 1 \times 2^{-2}) \times 2^3 = 6 = 6.0_{10}$

Bila kita susun dari nilai positif terkecil ke nilai terbesar, maka seluruh bilangannya digambarkan dalam diagram garis bilangan sebagai berikut:



Pada komputer IBM PC, bilangan titik-kambang berketelitian-tunggal dinyatakan dalam 32-bit *word* (1 bit tanda, 8 bit pangkat, dan 24 bit mantisa). Rentang nilai-nilai positifnya diperlihatkan pada Gambar 2.3.



Gambar 2.3 Rentang bilangan titik-kambang berketelitian tunggal pada *interpreter* Basic

Satu ukuran yang penting di dalam aritmetika komputer adalah seberapa kecil perbedaan antara dua buah nilai yang dapat dikenali oleh komputer. Ukuran yang digunakan untuk membedakan suatu bilangan riil dengan bilangan riil berikutnya adalah **epsilon mesin**. Epsilon mesin distandardisasi dengan menemukan bilangan titik-kambang terkecil yang bila ditambahkan dengan 1 memberikan hasil yang lebih besar dari 1. Dengan kata lain, jika epsilon mesin dilambangkan dengan e maka

$$1 + e > 1$$

(bilangan yang lebih kecil dari epsilon mesin didefinisikan sebagai nol di dalam komputer).

Epsilon mesin pada sistem bilangan riil yang ditunjukkan pada Gambar 2.3 adalah

$$e = 1.000000119 - 1.0 = 0.119 \times 10^{-6}$$

Gap (Δx) atau jarak antara sebuah bilangan titik-kambang dengan bilangan titik-kambang berikutnya, yang besarnya adalah

$$\Delta x = e \times R \qquad \text{(P.2.20)}$$

yang dalam hal ini R adalah bilangan titik-kambang sekarang. Contohnya, *gap* antara bilangan positif terkecil pertama 0.29×10^{-38} dengan bilangan titik-kambang terkecil kedua pada Gambar 2.3 adalah

$$\Delta x = (0.119 \times 10^{-6}) \times (0.29 \times 10^{-38}) = 0.345 \times 10^{-45}$$

dan dengan demikian bilangan titik-kambang terkecil kedua sesudah 0.29×10^{-38} adalah

$$0.29 \times 10^{-38} + 0.345 \times 10^{-45}$$

Dari persamaan P.2.16 dapat dilihat bahwa *gap* akan bertambah besar dengan semakin besarnya bilangan titik-kambang.

Keadaan *underflow* terjadi bila suatu bilangan titik-kambang tidak dapat dinyatakan di antara 0 dan bilangan positif terkecil (atau antara 0 dan bilangan negatif terbesar). Keadaan *overflow* terjadi bila suatu bilangan titik-kambang lebih besar dari bilangan positif terbesar (atau lebih kecil dari bilangan negatif terkecil)

Jika kita mengetahui jumlah bit mantisa dari suatu bilangan titik-kambang, kita dapat menghitung epsilon mesinnya dengan rumus

$$e = B^{1-n} \qquad \text{(P.2.21)}$$

yang dalam hal ini B adalah basis bilangan dan n adalah banyaknya digit (atau bit) bena di dalam mantisa. Pada contoh pertama di atas ($B = 2$ dan $n = 2$),

$$e = 2^{1-2} = 0.5$$

dan pada contoh bilangan titik-kambang berketelitian tunggal pada komputer IBM PC,

$$e = 2^{1-24} = 0.00000011920928955078125 = 0.119 \times 10^{-6}$$

Kita juga dapat menemukan perkiraan nilai epsilon mesin dengan prosedur yang sederhana. Gagasannya ialah dengan membagi dua secara terus menerus nilai 1 dan memeriksa apakah 1 ditambah hasil bagi itu lebih besar dari 1. Potongan programnya dituliskan di dalam Program 2.2.

Program 2.2 Program menghitung hampiran nilai epsilon mesin

```
procedure HitungEpsilonMesin1(var eps : real);
{ Prosedur untuk menemukan epsilon mesin
   Keadaan Awal : sembarang
  Keadaan Akhir: eps berisi harga epsilon mesin
}
begin
   eps:=1;
   while eps + 1 > 1 do
      eps:=eps/2;
   {eps + 1 < 1}

   eps:=2*eps;       {nilai epsilon mesin}
end;
```

Hasil pelaksanaan program dengan *compiler* Turbo Pascal dan komputer dengan *processor* 486DX adalah

$$e = 0.90949470177 \times 10^{-12}$$

Jika yang diinginkan adalah epsilon mesin dalam bentuk perpangkatan dari 2, prosedur untuk menghitungnya dituliskan di dalam Program 2.3.

Program 2.3 Program menghitung hampiran nilai epsilon mesin dalam bentuk 2^k

```
procedure HitungEpsilon_Mesin2(var n : integer);
{ Prosedur untuk menemukan epsilon mesin dalam bentuk perpangkatan 2
  Keadaan Awal  : sembarang
  Keadaan Akhir : 2^-(n-1) adalah epsilon mesin
}
var
  eps, ne : real;
begin
   eps:=1; ne:=2; n:=0;
   while ne > 1 do
    begin
      eps:=eps/2;
      ne:=1 + eps;
      n:=n + 1;
    end;
   {eps + 1 < 1}
   { Epsilon mesin ialah 2^-(n-1) }
end;
```

Hasil pelaksanaan program dengan *compiler* Turbo Pascal dan komputer dengan *processor* 486 adalah

$$e = 2^{-40} \qquad \text{(Keterangan: } 2^{-40} = 0.90949470177 \times 10^{-12}\text{)}$$

Nilai epsilon mesin yang diperoleh dapat berbeda-beda bergantung pada bahasa pemrograman dan komputer yang digunakan karena beberapa bahasa menggunakan bilangan berketelitian ganda (*double precision*) untuk representasi bilangan titik-kambangnya.

Epsilon dapat digunakan sebagai kriteria berhenti kekonvergenan pada pada prosedur lelaran yang konvergen. Nilai lelaran sekarang dibandingkan dengan nilai lelaran sebelumnya. Jika selisih keduanya sudah kecil dari epsilon mesin, lelaran dihentikan, tetapi jika tidak, lelaran diteruskan.

2.5.3 Pembulatan pada Bilangan Titik-Kambang

Dari ilustrasi pada upabab 2.5.2 jelaslah bahwa jumlah bilangan riil yang dapat dinyatakan sebagai bilangan titik-kambang terbatas banyaknya (bergantung pada banyaknya bit *word*). Bilangan titik-kambang yang tidak dapat mencocoki satu dari nilai-nilai di dalam rentang pasti terletak di dalam *gap*. Karena itu, bilangan tersebut dibulatkan (atau dikuantisasi) ke salah satu nilai di dalam rentang. Galat yang timbul akibat penghampiran tersebut diacu sebagai galat pembulatan (atau galat kuantisasi). Misalnya, 0.3748 dibulatkan ke 0.375, 3.2 dibulatkan ke 3.0, dan sebagainya.

Ada dua teknik pembulatan yang lazim dipakai oleh komputer, yaitu **pemenggalan** (*chopping*) dan **pembulatan ke digit terdekat** (*in-rounding*). Kedua teknik pembulatan tersebut diilustrasikan di bawah ini.

1. Pemenggalan (*chopping*)

Misalkan *a* adalah bilangan titik-kambang dalam basis 10:

$$a = \pm 0.d_1 d_2 d_3 \ldots d_n d_{n+1} \ldots \times 10^p$$

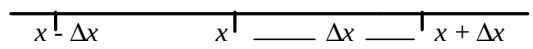
Misalkan *n* adalah banyak digit mantis komputer. Karena digit mantis *a* lebih banyak dari digit mantis komputer, maka bilangan *a* dipotong sampai *n* digit saja:

$$fl_{\text{chop}}(a) = \pm 0.d_1 d_2 d_3 \ldots d_{n-1} d_n \times 10^p \qquad \text{(P.2.21)}$$

Sebagai contoh, bilangan $p = 0.31459265358... \times 10^0$ di dalam komputer dengan 7 digit mantis disimpan sebagai

$fl_{\text{chop}}(p) = 0.3141592 \times 10^0$ dengan galat sebesar 0.00000065...

Perhatikan juga bahwa pemenggalan berarti sembarang besaran yang berada pada *gap* sebesar Δx akan disimpan sebagai besaran pada ujung selang yang lebih kecil, sehingga batas atas galat untuk pemenggalan adalah Δx (Gambar 2.4) [CHA91].



Gambar 2.4 Batas atas pemenggalan. Bilangan yang akan dipenggal pada mulanya terletak antara *x* dan $x + \Delta x$. Setelah pemenggalan, bilangan tersebut = *x*.

Contoh pemenggalan pada bilangan titik-kambang biner misalnya,

$$1/10 = 0.0001100110011001100110011001100110011 \ldots_2$$

Bila digunakan mantis 32 bit, komputer memenggal bilangan 1/10 di atas menjadi (setelah dinormalkan)

$$1/10 \approx 0.11001100110011001100110011001100_2 \times 2^{-3}$$

2. Pembulatan ke digit terdekat (*in-rounding*)

Misalkan a adalah bilangan titik-kambang dalam basis 10:

$$a = \pm 0.d_1 d_2 d_3 \ldots d_n d_{n+1} \ldots \times 10^p$$

Misalkan n adalah jumlah digit mantis komputer. Karena digit mantis a lebih banyak dari digit mantis komputer, maka bilangan a dibulatkan sampai n digit:

$$fl_{\text{round}}(a) = \pm 0.d_1 d_2 d_3 \ldots \hat{d}_n \times 10^p \qquad \text{(P.2.22)}$$

yang dalam hal ini

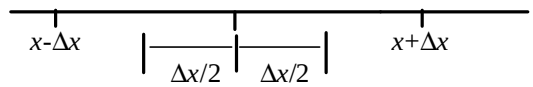
$$\hat{d}_n = \begin{cases} d_n & \text{, jika } d_{n+1} < 5 \\ d_n + 1 & \text{, jika } d_{n+1} > 5 \\ d_n & \text{, jika } d_{n+1} = 5 \text{ dan } n \text{ genap} \\ d_n + 1 & \text{, jika } d_{n+1} = 5 \text{ dan } n \text{ ganjil} \end{cases} \qquad \text{(P.2.23)}$$

Contohnya, bilangan p = 0.31459265358... $\times 10^0$ di dalam komputer hipotetis dengan 7 digit mantis dibulatkan menjadi fl(π) =0.3141593 $\times 10^0$ dengan galat sebesar 0.00000035.… Contoh ini memperlihatkan bahwa pembulatan ke digit terdekat menghasilkan galat yang lebih rendah daripada pemenggalan.

Contoh lainnya, nilai $a = 0.5682785715287 \times 10^{-4}$:

- di dalam komputer 7 digit dibulatkan menjadi $fl_{\text{round}}(a) = 0.5682786 \times 10^{-4}$
- di dalam komputer 8 digit dibulatkan menjadi $fl_{\text{round}}(a) = 0.56827857 \times 10^{-4}$
- di dalam komputer 6 digit dibulatkan menjadi $fl_{\text{round}}(a) = 0.568278 \times 10^{-4}$
- di dalam komputer 9 digit dibulatkan menjadi $fl_{\text{round}}(a) = 0.568278572 \times 10^{-4}$

Perhatikan juga bahwa pembulatan ke digit terdekat berarti sembarang besaran yang berada pada *gap* sebesar Δx akan disimpan sebagai bilangan terdekat yang diperbolehkan, sehingga batas atas galat untuk pembulatan adalah $\Delta x/2$ (Gambar 2.5):



Gambar 2.5 Batas atas pembulatan ke digit terdekat

Contoh pembulatan *p* yang diberikan di atas memperlihatkan bahwa galat pembulatan ke digit terdekat lebih rendah daripada galat pemenggalan, karena itu cara pemenggalan biasanya tidak direkomendasikan untuk dipakai. Namun yang mengherankan, kebanyakan komputer menggunakan cara pemenggalan! Alasannya adalah bahwa algoritma pembulatan ke digit terdekat lebih sukar sehingga membutuhkan waktu lebih lama dari pada waktu untuk pemenggalan. Sedangkan algoritma pemenggalan lebih sederhana sehingga mudah direalisasikan. Pendekatan ini dapat diterima dengan anggapan bahwa jumlah angka bena cukup besar sehingga galat pemenggalan yang dihasilkan biasanya dapat diabaikan [CHA91]. Algoritma pembulatan/pemenggalan dapat diimplementasilkan baik di dalam piranti keras atau di dalam rutin perangkat lunak.

Secara umum dapat dinyatakan bahwa satu-satunya cara meminimumkan galat pembulatan adalah menggunakan jumlah angka bena yang lebih banyak. Di dalam program komputer itu artinya kita menggunakan bilangan riil berketelitian ganda (*double precision*) ketimbang bilangan berketelitian tunggal (*single precision*).

2.5.4 Aritmetika Bilangan Titik-Kambang

Selain mengandung galat pembulatan pada representasinya, operasi aritmetika pada bilangan titik-kambang juga menghasilkan galat pembulatan yang lain. Operasi aritmetika pada bilangan titik-kambang meliputi operasi penambahan dan pengurangan, operasi perkalian, dan operasi pembagian.

2.5.4.1 Operasi Penambahan dan Pengurangan

Terdapat dua buah kasus serius yang menyebabkan timbulnya galat pembulatan pada operasi penjumlahan dua buah bilangan titik-kambang:

Kasus 1: Penjumlahan (termasuk pengurangan) bilangan yang sangat kecil ke (atau dari) bilangan yang lebih besar menyebabkan timbulnya galat pembulatan.

Galat pembulatan pada Kasus 1 ini terjadi karena untuk menjumlahkan dua buah bilangan yang berbeda relatif besar, pangkatnya harus disamakan terlebih dahulu (disamakan dengan pangkat bilangan yang lebih besar). Caranya adalah dengan menggeser digit-digit (atau bit) bilangan yang pangkatnya lebih kecil. Pergeseran digit (atau bit) ini mengakibatkan adanya digit (atau bit) yang hilang. Perhatikan contoh berikut.

Contoh 2.9

Misalkan digunakan komputer dengan mantis 4 digit (basis 10). Hitunglah

$$1.557 + 0.04381 = 0.1557 \times 10^{1} + 0.4381 \times 10^{-1}$$

Penyelesaian:

$0.1557 \times 10^{1} = 0.1557 \times 10^{1}$

$0.4381 \times 10^{-1} = 0.004381 \times 10^{1}$ + (pergeseran digit untuk menyamakan pangkat)

$= 0.160081 \times 10^{1}$

in-rounding $\rightarrow 0.1601 \times 10^{1}$

chopping $\rightarrow 0.1600 \times 10^{1}$

Perhatikanlah bahwa dua digit terakhir dari bilangan yang digeser ke kanan pada dasarnya telah hilang dari perhitungan.

Galat mutlak pembulatan $= | (0.160081 \times 10^{1}) - (0.1601 \times 10^{1}) | = 0.000019$

Galat mutlak pemenggalan $= | (0.160081 \times 10^{1}) - (0.1600 \times 10^{1}) | = 0.000081$ ■

Galat perhitungan semacam Kasus 1 ini dapat terjadi dalam perhitungan deret tak berhingga yang suku awalnya relatif lebih besar dibandingkan suku berikutnya. Jadi, setelah beberapa suku ditambahkan, kita berada dalam situasi penambahan besaran yang kecil terhadap besaran yang besar. Suatu cara mengurangi galat jenis ini adalah menjumlahkan deret dalam urutan terbalik -yakni dalam urutan yang menaik ketimbang menurun. Dengan cara ini, setiap suku baru akan sebanding besarnya dengan jumlah deret yang terakumulasi [CHA91].

Contoh 2.10

Misalkan digunakan komputer dengan mantis 4 digit (basis 10). Hitunglah

$$3677 - 0.3283 = 0.3677 \times 10^{4} - 0.3283 \times 10^{0}$$

Penyelesaian:

$0.3677 \times 10^{4} = 0.3677 \times 10^{4}$

$0.3283 \times 10^{0} = 0.00003283 \times 10^{4}$ - (pergeseran digit untuk menyamakan pangkat)

$= 0.36766717 \times 10^{4}$

in-rounding $\rightarrow 0.3677 \times 10^{4}$

chopping $\rightarrow 0.3676 \times 10^{4}$

Galat mutlak pembulatan $= |(0.36766717 \times 10^{4}) - (0.3677 \times 10^{4}) | = 0.00003283$

Galat mutlak pemenggalan $= |(0.36766717 \times 10^{4}) - (0.3676 \times 10^{4}) | = 0.00006717$ ■

Contoh 2.11

[NAK93] Misalkan digunakan komputer dengan mantis 24 bit (setara dengan 7 tempat desimal). Nilai

$$x = 1.0 + \sum_{i=1}^{10000} 0.00001 = 1.0 + \underbrace{0.00001 + 0.00001 + \ldots + 0.00001}_{10000 \text{ kali}}$$

akan dihitung dengan program Pascal yang menggunakan bilangan riil berketelitian-tunggal sebagai berikut:

```
program hitung;
var
   x, jumlah : real;
   i: integer;
begin
   jumlah:=1.0;
   for i:=1 to 10000 do
        jumlah:=jumlah + 0.00001;
   writeln('x=', jumlah);
end.
```

Bila digunakan komputer dengan 7 angka bena, hasil program di atas adalah $x = 1.100136$, padahal seharusnya $x = 1.0 + 10000(0.00001) = 1.0 + 0.1 = 1.1$.

Galat mutlak = $|1.1 - 1.100136| = 0.000136$

Galat relatif $= \dfrac{1.1 - 1.100136}{1.1} \times 100\% = -0.0124$

Bagaimana kita menjelaskan hasil perhitungan ini? Sumber penyimpangan ini adalah galat pembulatan. Karena bilangan 1 dapat dinyatakan secara persis dalam komputer, maka galatnya nol. Sebaliknya 0.00001 tidak dapat dinyatakan secara tepat dan dikuantisasi oleh nilai yang sedikit berbeda dibandingkan nilai sejatinya. Penyimpangan yang kecil itu dapat diabaikan untuk komputasi yang sedikit, namun terakumulasi setelah penjumlahan yang berulang kali [CHA91]. Untuk jelasnya perhatikan proses perhitungannya dalam sistem biner di bawah ini.

$$\begin{aligned}
1_{10} &= 0.10000000000000000000000_2 \times 2^1 \\
(0.00001)_{10} &= 0.10100111110001011010110 0_2 \times 2^{-16} \\
&= 0.00000000000000000101001111100010110101100_2 \times 2^1 + \\
\hline
(1 + 0.00001)_{10} &= 0.10000000000000000101001111100010110101100_2 \times 2^1
\end{aligned}$$

dibulatkan ke mantis 24 bit

$(1.00001)_{10}$ (nilai sejati)

$$0.10000000000000000101010 0_2 \times 2 = (1.0000100136)_{10}$$

(nilai hampiran)

galat mutlak = $\left| 1.00001 - 1.0000100136 \right| = 0.136 \times 10^{-7}$
galat total $\approx 10000 \times 0.136 \,.\, 10^{-7} = 0.136 \times 10^{-3} = 0.000136$

Satu cara untuk mengurangi galat total ini adalah menjumlahkan suku-suku dalam urutan terbalik, yaitu 0.00001 dijumlahkan terlebih dahulu sebanyak seribu kali, baru kemudian hasilnya dijumlahkan dengan 1.0. Jadi cara perhitungannya adalah

$$x = \left(\sum_{i=1}^{10000} 0.00001 \right) + 1.0 = \underbrace{0.00001 + 0.00001 + ... + 0.00001}_{10000 \text{ kali}} + 1.0$$

Selain itu, gunakan bilangan berketelitian ganda, sebab galat pembulatannya jauh lebih kecil. ■

Kasus 2: Pengurangan dua buah bilangan yang hampir sama besar (*nearly equal*). Bila dua bilangan titik-kambang dikurangkan, hasilnya mungkin mengandung nol pada posisi digit mantis yang paling berarti (posisi digit paling kiri). Keadaan ini dinamakan **kehilangan angka bena** *(loss of significance*). Baik pemenggalan maupun pembulatan ke digit terdekat menghasilkan jawaban yang sama.

Contoh 2.12

Kurangi 0.56780×10^5 dengan 0.56430×10^5 (5 angka bena)

Penyelesaian:

0.56780×10^5
0.56430×10^5 –
$0.00350 \times 10^5 \rightarrow$ dinormalisasi menjadi 0.350×10^3 (3 angka bena)

in-rounding $\rightarrow 0.350 \times 10^3$
chopping $\rightarrow 0.350 \times 10^3$

Hasil yang diperoleh hanya mempunyai 3 angka bena. Jadi kita kehilangan 2 buah angka bena. Meskipun kita dapat menuliskan hasilnya sebagai 0.35000×10^3, namun dua nol yang terakhir bukan angka bena tetapi sengaja ditambahkan untuk mengisi kekosongan digit yang hilang. ■

Hasil yang lebih dramatis diperlihatkan pada Contoh 2.13 dan Contoh 2.14 di bawah ini.

Contoh 2.13

Kurangi 3.1415926536 dengan 3.1415957341 (11 angka bena).

Penyelesaian:

$$3.1415926536 = 0.31415926536 \times 10^{1}$$
$$3.1415957341 = 0.31415957341 \times 10^{1} \text{ -}$$
$$\underline{\qquad\qquad\qquad\qquad\qquad\qquad}$$
$$-0.30805 \times 10^{-5} \quad \text{(5 angka bena)}$$

in-rounding $\rightarrow -0.30805 \times 10^{-5}$
chopping $\rightarrow -0.30805 \times 10^{-5}$

Jadi, kita kehilangan 6 angka bena!. ■

Contoh 2.14

Kurangi 0.7642×10^{3} dengan 0.7641×10^{3} (4 angka bena).

Penyelesaian:

$$0.7642 \times 10^{3}$$
$$0.7641 \times 10^{3} \text{ -}$$
$$\underline{\qquad\qquad\qquad\qquad}$$
$$0.0001 \times 10^{3} = 0.1 \times 10^{0} \quad \text{(1 angka bena)}$$

in-rounding $\rightarrow 0.1 \times 10^{0}$
chopping $\rightarrow 0.1 \times 10^{0}$

Jadi kita kehilangan 3 buah angka bena. ■

Kehilangan angka bena bila mengurangkan dua buah bilangan yang hampir sama besar merupakan sumber galat utama pada operasi bilangan titik-kambang. Kehilangan angka bena dapat dihindari dengan mengubah metode komputasi yang digunakan. Tujuan dari pengubahan metode komputasi adalah menghilangkan operasi pengurangan dua buah bilangan yang hampir sama besar, misalnya dengan pengelompokan suku-suku, perkalian dengan bentuk sekawan, menggunakan deret Taylor, atau manipulasi aljabar lainnya. Contoh 2.15 sampai 2.17 berikut mengilustrasikan cara pengubahan ini.

Contoh 2.15

[MAT92] Diberikan $f(x) = x(\sqrt{x+1} - \sqrt{x})$. Hitunglah $f(500)$ dengan menggunakan 6 angka bena dan pembulatan ke digit terdekat.

Penyelesaian:

$f(500) = 500(\sqrt{501} - \sqrt{500})$

= 500(22.3830 - 22.3607)

= 500(0.0223)

= 11.15 (empat angka bena)

(solusi sejatinya adalah 11.174755300747198..)

Hasil yang tidak akurat ini disebabkan adanya operasi pengurangan dua bilangan yang hampir sama besar, yaitu 22.3830 - 22.3607. Ketelitian hasil dapat kita tingkatkan bila kita dapat menghilangkan pengurangan tersebut. Caranya adalah mengubah metode komputasi sedemikian sehinggga pengurangan dua bilangan yang hampir sama besar menjadi hilang.

Susunlah kembali fungsi $f(x)$ menjadi bentuk yang lebih baik:

$$f(x) = x(\sqrt{x+1} - \sqrt{x})$$

$$= x(\sqrt{x+1} - \sqrt{x})\frac{(\sqrt{x+1} + \sqrt{x})}{(\sqrt{x+1} + \sqrt{x})}$$

$$= \frac{x[(\sqrt{x+1})^2 - (\sqrt{x})^2]}{(\sqrt{x+1} + \sqrt{x})}$$

$$= \frac{x}{\sqrt{x+1} + \sqrt{x}} = p(x)$$

sehingga

$$p(500) = \frac{500}{\sqrt{501} + \sqrt{500}} = \frac{500}{22.3830 + 22.3607} = 11.1748$$

Hasil ini jauh lebih baik dibandingkan yang pertama. Solusi sejatinya, 11.174755300747198..., jika dibulatkan sampai 6 angka bena adalah 11.1747, yang lebih dekat ke $p(500)$ daripada ke $f(500)$. ■

Contoh 2.16

Hitunglah akar-akar polinom $x^2 - 40x + 2 = 0$ sampai 4 angka bena.

Penyelesaian:

$$x_{1,2} = \frac{40 \pm \sqrt{(-40)^2 - 8}}{2} = 20 \pm \sqrt{398} = 20.00 \pm 19.95$$

$x_1 = 20 + 19.95 = 39.95$ (4 angka bena)

x_2 = 20 - 19.95 = 0.05 (1 angka bena)
(kehilangan tiga buah angka bena akibat pengurangan dua buah bilangan yang hampir sama, yaitu 20 – 19.95)

Nilai x_2 yang lebih akurat dapat diperoleh dengan mengingat lagi pelajaran matematika di sekolah lanjutan bahwa:

jika x_1 dan x_2 adalah akar-akar persamaan $ax^2 + bx + c = 0$ maka $x_1x_2 = c/a$

Dengan demikian, x_2 dihitung sebagai berikut:

$39.95x_2 = 2/1 \Rightarrow x_2 = 2.000/39.95 = 0.05006$ (4 angka bena, lebih akurat) ■

Contoh 2.17

[MAT92] Diberikan $f(x) = \dfrac{e^x - 1 - x}{x^2}$. Hitung $f(0.01)$ sampai 6 angka bena.

Penyelesaian:

$$f(0.01) = \frac{e^{0.01} - 1 - 0.01}{0.01^2} = \frac{1.01005 - 1 - 0.01}{0.0001} = 0.5 \quad (\text{1 angka bena})$$

Hasil yang tidak akurat ini karena adanya kehilangan angka bena akibat pengurangan dua buah nilai yang hampir sama besar, yaitu 1.01005 – 1. Hasil yang lebih akurat dapat diperoleh dengan menguraikan $f(x)$ ke dalam deret Maclaurin sampai suku orde 2 (Gunakan dalil *L'Hospital b*ila menjumpai pembagian 0/0):

$f(x) \approx p(x) = 1/2 + x/6 + x^2/24$

sehingga

$p(0.01) = 1/2 + 0.01/6 + 0.01^2/6 = 0.5 + 0.001667 + 0.00005 = 0.501671$

Solusi sejatinya adalah 0.50167084168057542..., yang jika dibulatkan sampai 6 angka bena adalah 0.501671 = $p(0.01)$. ■

2.5.4.2 Operasi Perkalian dan Pembagian

Operasi perkalian dan pembagian dua buah bilangan titik-kambang tidak memerlukan penyamaan pangkat seperti halnya pada penjumlahan Perkalian dapat dilakukan dengan mengalikan kedua mantis dan menambahkan kedua pangkatnya. Pembagian dikerjakan dengan membagi mantis dan mengurangkan pangkatnya. Selain itu, register dengan panjang ganda dibutuhkan untuk menyimpan hasil antara. Hasil akhir dipotong ke dalam register tunggal.

Contoh 2.18

Hitung perkalian 0.4652×10^{4} dengan 0.1456×10^{-1} (4 angka bena).

Penyelesaian:

Kalikan mantis: 0.4652 × 0.1456 = 0.06773312

Jumlahkan pangkat: 4 + (-1) = 3

Gabungkan mantis dengan pangkat: 0.06773312×10^{3}
Normalisasi: 0.6773312×10^{2}
in-rounding $\rightarrow 0.6773 \times 10^{2}$
chopping $\rightarrow 0.6773 \times 10^{2}$ ■

Contoh 2.19

Hitung $(0.8675 \times 10^{-5})/0.2543 \times 10^{-2}$ (4 angka bena).

Penyelesaian:

Bagi mantis: 0.8675 : 0.2543 = 3.4113252

Kurangi pangkat: -4 - (-2) = -2

Gabungkan mantis dengan pangkat: 3.4113252×10^{-2}
Normalisasi: $0.34113252 \times 10^{-1}$
in-rounding $\rightarrow 0.3411 \times 10^{-1}$
chopping $\rightarrow 0.3411 \times 10^{-1}$ ■

2.6 Perambatan Galat

Galat yang dikandung dalam bilangan titik-kambang merambat pada hasil komputasi. Misalkan terdapat dua bilangan a dan b (nilai sejati) dan nilai hampirannya masing-masing $\hat{a}$ dan $\hat{b}$, yang mengandung galat masing-masing e_a dan e_b. Jadi, kita dapat menulis

$$a = \hat{a} + e_a$$

dan

$$b = \hat{b} + e_b.$$

Di bawah ini akan diperlihatkan bagaimana galat merambat pada hasil penjumlahan dan perkalian *a* dan *b*.

Untuk penjumlahan,

$$a + b = (\hat{a} + e_a) + (\hat{b} + e_b) = (\hat{a} + \hat{b}) + (e_a + e_b) \qquad \text{(P.2.24)}$$

Jadi, galat hasil penjumlahan sama dengan jumlah galat masing-masing *operand*.

Untuk perkalian,

$$ab = (\hat{a} + e_a)(\hat{b} + e_b) = \hat{a}\,\hat{b} + \hat{a}\,e_b + \hat{b}\,e_a + e_a\,e_b$$

yang bila kita susun menjadi

$$ab - \hat{a}\,\hat{b} = \hat{a}\,e_b + \hat{b}\,e_a + e_a\,e_b$$

Dengan mengandaikan bahwa $a \neq 0$ dan $b \neq 0$, maka galat relatifnya adalah

$$\frac{ab - \hat{a}\hat{b}}{ab} = \frac{\hat{a}e_b + \hat{b}e_a + e_a e_b}{ab}$$
$$= \frac{\hat{a}e_b}{ab} + \frac{\hat{b}e_a}{ab} + \frac{e_a e_b}{ab}$$

Dengan mengandaikan bahwa *a* dan $\hat{a}$ hampir sama besar, yaitu $a \approx \hat{a}$, begitu juga *b* dan $\hat{b}$,dan e_a dan e_b sangat kecil maka $\hat{a}/a \approx 1$, $\hat{b}/b \approx 1$, dan $(e_a/a)(e_b/b) \approx 0$. Dengan demikian

$$\frac{ab - \hat{a}\hat{b}}{ab} = \frac{e_b}{b} + \frac{e_a}{a} = e_{Rb} + e_{Ra} \qquad \text{(P.2.25)}$$

Jadi, galat relatif hasil perkalian sama dengan jumlah galat relatif masing-masing *operand*.

Jika operasi aritmetika hanya dilakukan sekali saja, maka kita tidak perlu terlalu khawatir terhadap galat yang ditimbulkannya. Namun, bila operasi dilakukan terhadap seruntunan komputasi, maka galat operasi aritmetika awal akan merambat dalam seruntunan komputasi. Bila hasil perhitungan sebuah operasi aritmetika dipakai untuk operasi selanjutnya, maka akan terjadi penumpukan galat yang semakin besar, yang mungkin mengakibatkan hasil perhitungan akhir menyimpang dari hasil sebenarnya. Ketidakpastian hasil akibat galat pembulatan

yang bertambah besar itu dapat menyebabkan perhitungan menjadi **tidak stabil** (*unstable* atau *instability*), sedangkan lawannya adalah **stabil**, yang merupakan proses numerik yang diinginkan. Metode komputasi dikatakan stabil jika galat pada hasil antara (*intermediate*) hanya sedikit pengaruhnya pada hasil akhir. Jika galat pada hasil antara memberikan pengaruh yang besar pada hasil akhir maka metode komputasinya dikatakan tidak stabil [KRE88]. Ketidakstabilan ini dinamakan "ketidakstabilan numerik", yang dapat dihindari dengan memilih metode komputasi yang stabil (di dalam Bab Solusi Persamaan Diferensial Biasa masalah ini akan dikemukakan lagi). Ketidakstabilan numerik harus dibedakan dengan "ketidakstabilan matematik" dari persoalan yang diberikan. Ketidakstabilan matematik sering dinamakan **kondisi buruk** (*ill conditioned*), yaitu kondisi yang timbul karena hasil perhitungan sangat peka terhadap perubahan kecil data. Kondisi buruk didiskusikan lebih komprehensif di bawah ini.

2.7 Kondisi Buruk

Suatu persoalan dikatakan **berkondisi buruk** (*ill conditioned*) bila jawabannya sangat peka terhadap perubahan kecil data (misalnya perubahan kecil akibat pembulatan). Bila kita mengubah sedikit data, maka jawabannya berubah sangat besar (drastis). Lawan dari berkondisi buruk adalah **berkondisi baik** (*well conditioned*). Suatu persoalan dikatakan berkondisi baik bila perubahan kecil data hanya mengakibatkan perubahan kecil pada jawabannya.

Sebagai contoh, tinjau persoalan menghitung akar persamaan kuadrat $ax^2 + bx + c = 0$ di bawah ini. Di sini kita hanya mengubah nilai nilai tetapan *c*-nya saja:

(i) $x^2 - 4x + \mathbf{3.999} = 0 \Rightarrow$ akar-akarnya $x_1 = 2.032$ dan $x_2 = 1.968$

Sekarang, ubah 3.99 menjadi 4.00:

(ii) $x^2 - 4x + \mathbf{4.000} = 0 \Rightarrow$ akar-akarnya $x_1 = x_2 = 2.000$

Ubah 4.00 menjadi 4.001:

(iii) $x^2 - 4x + \mathbf{4.001} = 0 \Rightarrow$ akar-akarnya imajiner

Kita katakan bahwa persoalan akar-akar persamaan kuadrat di atas berkondisi buruk, karena dengan pengubahan sedikit saja data masukannya (dalam hal ini nilai koefisien *c*), ternyata nilai akar-akarnya berubah sangat besar.

Kapankah akar persamaan $f(x) = 0$ berkondisi buruk? Misalkan $f(x)$ diubah sebesar e sehingga akarnya berubah sebesar h:

$$f(x + h) + e = 0 \qquad \text{P.2.26)}$$

Bila karena pengubahan e yang sangat kecil mengakibatkan h menjadi besar, dikatakan persoalan mencari akar $f(x) = 0$ berkondisi buruk [NOB72].

Bukti:

Kita menggunakan teorema nilai rata-rata (TNR) di dalam kalkulus diferensial:

$$\frac{f(m) - f(n)}{m - n} = f'(t) \qquad , \quad m < t < n$$

Karena $m - n = h \Rightarrow m = n + h$, maka

$$\frac{f(m + h) - f(n)}{h} = f'(t)$$

atau

$$f(p + h) = f(p) + h\, f'(t) \qquad \text{(P.2.27)}$$

Terapkan (P.2.27) pada (P.2.26):

$$f(x) + hf'(t) + e = 0 \qquad , \; x < t < x + h \qquad \text{(P.2.28)}$$

Pada persoalan pencarian akar, $f(x) = 0$, sehingga

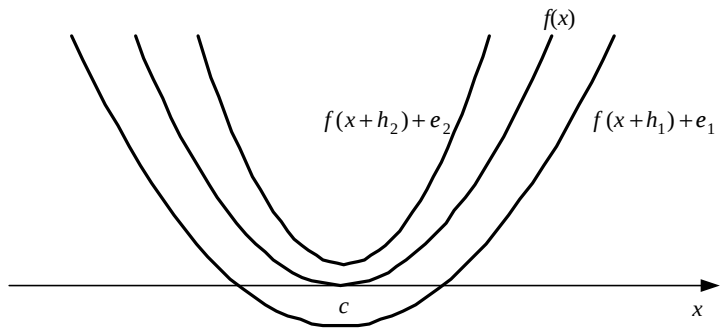
$$hf'(t) + e = 0 \qquad \text{(P.2.29)}$$

atau

$$h = \frac{-e}{f'(t)} \qquad \text{(P.2.30)}$$

Terlihat dari P.2.30 bahwa akar-akar $f(x) = 0$ akan berkondisi buruk bila $f'(t)$ bernilai sangat kecil. Bila $f'(t)$ sangat kecil nilainya maka h menjadi sangat besar yang berarti akar x bertambah sebesar h tersebut. ■

Masalah seperti ini sering muncul pada pencarian akar kembar (seperti pada contoh di atas). Nilai $f'(x)$ di sekitar akar kembar sangat kecil (mendekati 0), yang berakibat nilai h menjadi sangat besar (lihat Gambar 2.6).



Gambar 2.6 Kondisi buruk pada pencarian akar kembar

Contoh persoalan yang berkondisi buruk lainnya adalah persoalan mencari mencari solusi sistem persamaan lanjar berupa titik potong dari dua buah garis lurus yang berbentuk $ax + by = c$ yang diubah-ubah koefisiennya (b dan c):

(i) $x + y = 2$
$x + 0.9999y = 1.9999$
Solusi: $x = y = 1.0000$

(ii) $x + y = 2$
$x + 0.9999y = 2.0010$
Solusi: $x = 12,\ y = -10$

(iii) $x + y = 2$
$x + y = 1.9999$
Solusi: tidak ada

(iv) $x + y = 2$
$x + y = 2$
Solusi: tidak berhingga, yaitu disepanjang garis $x + y = 2$

Kondisi buruk yang terjadi pada perhitungan titik potong dua buah garis lurus dapat kita analisis sebagai berikut. Sistem persamaan lanjar di atas dapat ditulis sebagai

$$\begin{aligned} x + \quad & y = 2 \\ x + (1 + e)y \; & = 2 + d \end{aligned}$$

yang dalam hal ini e dan d dalam orde 10^{-4}.

Solusi sistem persamaan lanjar tersebut adalah

$$\begin{aligned} x + \quad & y = 2 \\ x + (1 + e)y \; & = 2 + d \quad - \end{aligned}$$

$$-ey = -d \Rightarrow y = d/e \qquad \text{dan} \quad x = 2 - d/e \quad , e \neq 0$$

Nilai e dan d sangat penting. Perubahan kecil e mempunyai pengaruh yang besar pada solusi, yang berarti persoalan mencari solusi sistem persamaan lanjar berkondisi buruk.

2.8 Bilangan Kondisi

Kondisi komputasi numerik dapat diukur dengan **bilangan kondisi**. Bilangan kondisi merupakan ukuran tingkat sejauh mana ketidakpastian dalam x diperbesar oleh $f(x)$ [CHA88]. Bilangan kondisi dapat dihitung dengan bantuan deret Taylor. Fungsi $f(x)$ diuraikan di sekitar $\hat{x}$ sampai suku orde pertama:

$$f(x) \approx f(\hat{x}) + f'(\hat{x})(x - \hat{x}) \qquad \text{(P.2.31)}$$

Galat relatif hampiran dari $f(x)$ adalah

$$e_{RA}[f(x)] = \frac{f(x) - f(\hat{x})}{f(\hat{x})} \approx \frac{f'(\hat{x})(x - \hat{x})}{f(\hat{x})} \qquad \text{(P.2.32)}$$

dan galat relatif hampiran dari x adalah

$$e_{RA}[x] = \frac{x - \hat{x}}{\hat{x}} \qquad \text{(P.2.33)}$$

Bilangan kondisi didefinisikan sebagai nisbah (*ratio*) antara P.2.32 dan P.2.33:

$$\text{Bilangan kondisi} = \left|\frac{e_{RA}[f(x)]}{e_{RA}[x]}\right| = \left|\frac{\hat{x}f'(\hat{x})}{f(\hat{x})}\right| \qquad \text{(P.2.34)}$$

Arti dari bilangan kondisi adalah:

- bilangan kondisi = 1 berarti galat relatif hampiran fungsi sama dengan galat relatif *x*
- bilangan kondisi lebih besar dari 1 berarti galat relatif hampiran fungsi besar
- bilangan kondisi lebih kecil dari 1 berarti galat relatif hampiran fungsi kecil (kondisi baik)

Suatu komputasi dikatakan berkondisi buruk jika bilangan kondisinya sangat besar, sebaliknya berkondisi baik bila bilangan kondisinya sangat kecil.

Contoh 2.20

Misalkan $f(x) = \sqrt{x}$. Tentukan bilangan kondisi perhitungan akar kuadrat *x*.

Penyelesaian:

Hitung $f'(x)$ terlebih dahulu

$$f'(x) = \frac{1}{2\sqrt{x}}$$

yang akan digunakan untuk menghitung

$$\text{bilangan kondisi} = \left|\frac{\hat{x}/(2\sqrt{\hat{x}})}{\sqrt{\hat{x}}}\right| = \frac{1}{2}$$

Bilangan kondisi ini sangat kecil, yang berarti penarikan akar kuadrat *x* merupakan proses yang berkondisi baik. Sebagai contoh, $\sqrt{20.999}$ = 4.5824665, dan jika 20.999 diubah sedikit (dibulatkan) menjadi 21.000 maka $\sqrt{21.000}$ = 4.5825756. Ternyata perubahan kecil pada nilai *x* hanya berakibat perubahan sedikit pada $f(x)$. ■

Contoh 2.21

Hitung bilangan kondisi $f(x) = \dfrac{10}{1-x^2}$.

Penyelesaian:

Hitung $f'(x)$ terlebih dahulu

$$f'(x) = \frac{20x}{(1-x^2)^2}$$

yang digunakan untuk menghitung

$$\text{bilangan kondisi} = \left| \frac{\hat{x}[20\hat{x}/(1-\hat{x}^2)^2]}{10/(1-\hat{x}^2)} \right| = \left| \frac{2\hat{x}^2}{1-\hat{x}^2} \right|$$

Bilangan kondisi ini sangat besar untuk $|x| \approx 1$. Jadi, menghitung $f(x)$ untuk x mendekati 1 atau -1 sangat buruk keadaannya, karena galat relatifnya besar. Sebagai contoh, $f(1.009)$ = -55.306675, tetapi $f(1.01)$ = -497.51243. Ternyata perubahan kecil pada nilai x di sekitar 1 (karena dibulatkan dari 4 angka bena menjadi 3 angka bena), mengakibatkan nilai $f(x)$ berubah sangat besar. Untuk x yang jauh dari 1 atau –1, $f(x)$ berkondisi baik. ■

Contoh 2.22

[CHA91] Hitung bilangan kondisi untuk $f(x) = \tan(x)$.

Penyelesaian:

Hitung $f'(x)$ terlebih dahulu

$$f'(x) = \frac{1}{\cos^2(x)}$$

yang digunakan untuk menghitung

$$\text{bilangan kondisi} = \left| \frac{\hat{x}[1/\cos^2(\hat{x})]}{\tan(\hat{x})} \right|$$

Bilangan kondisi ini sangat besar untuk $x \approx \pi/2$. Misalkan untuk $x = \pi/2 + 0.1(\pi/2)$,

bilangan kondisi = 1.7279(40.86)/-6.314 = -11.2

dan untuk $x = \pi/2 + 0.01(\pi/2)$,

bilangan kondisi = 1.5865(4053)/-63.66 = -101 ■

Orang yang bijaksana belajar dari kesalahan orang lain, hanya orang yang bodohlah yang belajar dari kesalahannya sendiri.
(Pepatah Rusia)

Soal Latihan

1. Tentukan hampiran fungsi di bawah ini ke dalam deret Taylor:

   (a) ln(x) sampai orde-4 di sekitar x_0=1, lalu hampiri nilai ln(0.9).
   (b) $f(x) = e^x - 1$ sampai orde-3 di sekitar x_0=0, Lalu hitung nilai f(0.0001) sampai empat angka bena.
   (c) sinh(x) = $^1/_2$ (e^x - e^{-x}) di sekitar x_0 = 0, lalu hitung nilai hampiran $\int_0^1 \sinh(x)dx$
   (d) sin(x) sampai orde-3, lalu tentukan batas atas galat sin(x) jika $0 \le x \le 0.5$.

2. (a) Tentukan polinom Maclarin orde 4 untuk $f(x)$, kemudian gunakan polinom tersebut untuk menghampiri nilai $f(0.23)$, serta tentukan batas atas galatnya.
      (i) $f(x) = \sin(2x)$
      (ii) $f(x) = \ln(1+x)$

   (b) Cari polinom Taylor orde 3 pada x_0=1 untuk $f(x) = x^3 - 2x^2 + 3x + 5$ dan perlihatkan bahwa ia mewakili $f(x)$ secara tepat

   (c) Hitunglah

   $$\int_0^1 \sin(2x)dx$$

      (i) secara analitis (solusi sejati)
      (ii) secara numerik, yang dalam hal ini sin(2x) dihampiri dengan deret Maclarin yang telah anda dapatkan pada jawaban 2(a)(i) di atas. Hitung galat mutlak dan galat relatif hasilnya. Pakailah enam angka bena untuk, baik untuk setiap hasil antara maupun hasil akhir.

3. Hitung $\sqrt{10.1} - \sqrt{10}$ secara langsung tetapi hasil setiap perhitungan antara dan hasil akhir dibulatkan sampai empat angka bena. Kemudian, hitunglah $\sqrt{10.1} - \sqrt{10}$ dengan cara yang lebih baik.

4. Carilah akar persamaan kuadrat $x^2 - 10.1x + 1 = 0$ dengan rumus *abc*, yang setiap hasil perhitungan antara maupun hasil perhitungan akhir dibulatkan dengan teknik:

   (a) pembulatan ke dalam (*in-rounding*)

(b) pemenggalan (*chopping*)

sampai empat angka bena. Bandingkan hasilnya jika akar terbesar (x_1) dihitung dengan rumus *abc* dan akar terkecil (x_2) dengan rumus $x_1x_2 = c/a$

5. Diberikan beberapa bilangan titik-kambang yang telah dinormalkan sebagai berikut:

   $a = 0.4523123 \times 10^{-4}$
   $b = 0.2365401 \times 10^{1}$
   $c = 0.4520156 \times 10^{-4}$
   $d = 0.1234567 \times 10^{-3}$

   Bila mesin yang digunakan untuk operasi aritmetika mempunyai tujuh angka bena, hitung hasil komputasi yang diberikan oleh mesin tersebut (dalam bentuk bilangan titik-kambang ternormalisasi):

   (i) $a + b + c + d$
   (ii) $a + c + d + b$
   (iii) $a - c$
   (iv) $ab - c$

6. Misalkan digunakan mesin hipotetik dengan mantis empat angka bena. Lakukan operasi aritmetika untuk bilangan titik-kambang ternormalisasi berikut. Normalkan hasilnya.

   (a) $0.3796 \times 10^{2} + 0.9643 \times 10^{-2}$
   (b) $0.4561 \times 10^{-2} - 0.6732 \times 10^{-2}$
   (c) 0.1234×10^{3} x 0.4321×10^{-1}

7. Carilah cara yang lebih baik untuk menghitung:

(i) $f(x) = (x - \sin(x))/\tan(x)$	untuk x mendekati nol
(ii) $f(x) = x - \sqrt{(x^2 - a)}$	untuk x yang jauh lebih besar dari a
(iii) $f(x) = \cos^2(x) - \sin^2(x)$	untuk x di sekitar $\pi/4$
(iv) $f(x) = \log(x + 1) - \log(x)$	untuk x yang besar
(v) $(1 + \alpha)^{1/2} - 1$, $\lvert\alpha\rvert \leq 0.01$	sampai enam angka bena
(vi) $\sin(\alpha + x) - \sin(\alpha)$	untuk x yang kecil
(vii) $(a + x)^n - a^n$	untuk x yang kecil
(viii) $((x^3 - 3x^2) + 3x) - 1$	untuk $x = 2.72$
(ix) $\sqrt{(1 + \cos x)/2}$	untuk $x \approx \pi/4$

8. Bagaimana cara menghitung

$$\frac{3a}{4} - \sin(a) + \frac{\sin(2a)}{8}$$

sampai 6 angka bena untuk $|a| \leq 0.2$? (Petunjuk : gunakan deret Maclaurin)

9. Diketahui $f(x) = \cos(x)$. Tentukan $f'(1)$ dengan teorema dasar turunan:

$$\lim_{h \to 0} \frac{f(x+h) - f(x)}{h} = f'(x)$$

Hitung $f'(1)$ dengan bermacam-macam h = 0.1, 0.01, 0.001, 0.0001, 0.00001, 0.000001. Untuk memperbaiki hasil perhitungan, hitunglah $f'(1)$ dengan cara yang lebih baik.

10. (a) Hitunglah dengan lima angka bena nilai $f(13.400)$ bila

$$f(x) = x - 1000(\sqrt{x+0.1} - \sqrt{x})$$

(b) Perbaiki hasil perhitungan anda dengan mengubah metode komputasi.

11. Tentukan bilangan kondisi fungsi-fungsi berikut ini dan tentukan apakah fungsi tersebut berkondisi baik atau berkondisi buruk. Jika berkondisi baik, tentukan di x berapakah fungsi ini berkondisi buruk (berikan contoh nilai x untuk memperjelas jawaban anda)

(a) $f(x) = 1/(1 - x)$
(b) $f(x) = x^2$
(c) $f(x) = x^n$
(d) $f(x) = \sin(x)$
(e) $f(x) = \log(x)$

12. Uraikan $f(x) = \cos(x)$ di sekitar $x = \pi/4$. Hitunglah $f(\pi/3)$ sampai galat relatif hampiran kurang dari 0.5%. (Petunjuk: hitung suku demi suku, setiap kali menambahkan dengan jumlah suku yang lama, hitung galat relatif hampiran)

Esensi dari matematika adalah kebebasannya
(George Cantor)