TF2202 Komputasi Rekayasa

Aproksimasi

Tim Praktikum Komputasi Rekayasa 2021
Teknik Fisika
Institut Teknologi Bandung

1 Chapra Contoh 3.2

Fungsi eksponensial dapat dihitung dengan menggunakan deret sebagai berikut:

$$e^x = 1 + x + \frac{x^2}{2} + \frac{x^3}{3!} + \cdots + \frac{x^n}{n!} \tag{1}$$

Kita ingin menggunakan Persamaan (1) untuk menghitung estimasi dari $e^{0.5}$. Dengan menggunakan kriteria dari Scarborough:

$$\epsilon_s = (0.5 \times 10^{2-3})\% = 0.05\%$$

Kita akan menambahkan suku-suku pada Persamaan (1) sampai ϵ_a lebih kecil dari ϵ_s.

Program Python berikut ini dapat digunakan untuk melakukan perhitungan yang ada pada buku teks.

```python
from math import factorial, exp

def approx_exp(x, N):
  assert(N >= 0)
  if N == 0:
    return 1
  s = 0.0
  for i in range(N+1):
    s = s + x**i/factorial(i)
  return s

x = 0.5
true_val = exp(x) # from math module

n_digit = 3
# Equation 3.7
ε_s_percent = 0.5*10**(2-n_digit)

prev_approx = 0.0
for N in range(50):
  approx_val = approx_exp(x, N)
  ε_t_percent = abs(approx_val - true_val)/true_val * 100
  if N > 0:
    ε_a_percent = abs(approx_val - prev_approx)/approx_val * 100
  else:
    ε_a_percent = float('nan')
  prev_approx = approx_val
  print("%3d %18.10f %10.5f%% %10.5f%%" % (N+1, approx_val, ε_t_percent, ε_a_percent))
  if ε_a_percent < ε_s_percent:
    print("Converged within %d significant digits" % n_digit)
    break
```

```
print("true_val   is %18.10f" % true_val)
print("approx_val is %18.10f" % approx_val)
```

Catatan: Pada program di atas, `for`-loop digunakan dengan jumlah iterasi yang relatif besar. Anda dapat menggunakan `while`-loop sebagai gantinya.

Contoh output:

```
  1       1.0000000000   39.34693%        nan%
  2       1.5000000000    9.02040%   33.33333%
  3       1.6250000000    1.43877%    7.69231%
  4       1.6458333333    0.17516%    1.26582%
  5       1.6484375000    0.01721%    0.15798%
  6       1.6486979167    0.00142%    0.01580%
Converged within 3 significant digits
true_val   is       1.6487212707
approx_val is       1.6486979167
```

Soal 1. Ulangi perhitungan ini untuk jumlah digit signifikan yang berbeda, misalnya 5, 8, dan 10 digit signifikan. Silakan melakukan modifikasi terhadap program yang diberikan.

2 Chapra Contoh 3.2, *single precision*

Secara default, perhitungan dengan *floating number* pada Python (dan NumPy) dilakukan dengan menggunakan *double precision*. Pada bagian ini, kita akan mengulangi Chapra Contoh 3.2 dengan menggunakan *single precision*. Pada C dan C++, tipe yang relevan adalah `float` untuk *single precision* dan `double` untuk *double precision*. Pada Fortran kita dapat menggunakan `REAL(4)` untuk *single precision* dan `REAL(8)` untuk *double precision*.

Karena Python merupakan bahasa pemrograman dinamik yang *type-loose* kita tidak dapat dengan mudah memberikan spesifikasi pada variabel yang kita gunakan. Meskipun demikian, kita dapat menggunakan *single precision* pada Python melalui `np.float32` [1], meskipun program yang dihasilkan kurang elegan. Selain itu, kita juga harus mengecek apakah hasil akhir yang diberikan tetap berupa *single precision* (tidak terjadi *type promotion* ke *double precision*).

```
from math import factorial
import numpy as np

def approx_exp(x, N):
  assert(N >= 0)
  if N == 0:
      return 1
  s = np.float32(0.0)
  for i in range(N+1):
      s = s + np.float32(x**i)/np.float32(factorial(i))
  return s

x = np.float32(0.5)
true_val = np.exp(x) # from np module

n_digit = 3
# Equation 3.7
ε_s_percent = np.float32(0.5)*np.float32(10**(2-n_digit))

prev_approx = np.float32(0.0)
for N in range(50):
  approx_val = approx_exp(x, N)
```

[1] np digunakan sebagai singkatan dari modul `numpy`

```
  ε_t_percent = abs(approx_val - true_val)/true_val * 100
  if N > 0:
    ε_a_percent = abs(approx_val - prev_approx)/approx_val * 100
  else:
    ε_a_percent = float('nan')
  prev_approx = approx_val
  print("%3d %18.10f %10.5f%% %10.5f%%" % (N+1, approx_val, ε_t_percent, ε_a_percent))
  if ε_a_percent < ε_s_percent:
    print("Converged within %d significant digits" % n_digit)
    break

print("true_val   is %18.10f" % true_val)
print("approx_val is %18.10f" % approx_val)

# Make sure that float32 is used
print()
print("type(true_val)   = ", type(true_val))
print("type(approx_val) = ", type(approx_val))
```

Berikut ini adalah keluaran dari program.

```
  1       1.0000000000   39.34693%        nan%
  2       1.5000000000    9.02040%   33.33333%
  3       1.6250000000    1.43876%    7.69231%
  4       1.6458333731    0.17516%    1.26583%
  5       1.6484375000    0.01721%    0.15798%
  6       1.6486979723    0.00141%    0.01580%
Converged within 3 significant digits
true_val   is       1.6487212181
approx_val is       1.6486979723

type(true_val)   =  <class 'numpy.float32'>
type(approx_val) =  <class 'numpy.float32'>
```

Soal 2. Ulangi perhitungan pada Chapra Contoh 3.2 dengan menggunakan *single precision* dengan jumlah digit signifikan yang berbeda, misalnya 5, 8, dan 10 digit signifikan (berdasarkan kriteria Scarborough). Bandingkan hasil yang Anda dapatkan jika *double precision*. Apa yang dapat Anda simpulkan?

3 Chapra Contoh 3.8

Akar-akar suatu polinomial kuadrat:

$$ax^2 + bx + c = 0$$

diberikan oleh formula berikut

$$x_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a} \tag{2}$$

Untuk kasus di mana $b^2 \gg 4ac$ perbedaan antara pembilang dapat menjadi sangat kecil. Pada kasus tersebut, kita dapat menggunakan *double precision* untuk mengurangi kesalahan pembulatan. Selain itu, kita juga dapat menggunakan formula:

$$x_{1,2} = \frac{-2c}{b \pm \sqrt{b^2 - 4ac}} \tag{3}$$

Mengikuti contoh yang diberikan pada buku, kita akan menggunakan $a = 1$, $b = 3000.001$, dan $c = 3$. Akar-akar eksaknya adalah $x_1 = -0.001$ dan $x_2 = -3000$.

Soal 3. Buat program Python dengan menggunakan *single precision* dan *double precision* untuk melihat perbedaan hasil yang diberikan dari Persamaan (2) dan Persamaan (3).

Program berikut ini adalah dalam *single precision* yang dapat Anda lengkapi. Anda juga dapat menggunakan program yang Anda tulis sendiri dari awal atau modifikasi dari program ini.

```python
import numpy as np

def calc_quad_root_v1(a, b, c):
  D = np.float32(b**2) - np.float32(4)*a*c
  x1 = (-b + np.sqrt(D))/(np.float32(2)*a)
  x2 = # ... lengkapi
  return x1, x2

def calc_quad_root_v2(a, b, c):
  D = # ... lengkapi
  x1 = # ... lengkapi
  x2 = # ... lengkapi
  return x1, x2

a = np.float32(1.0)
b = np.float32(3000.001)
c = np.float32(3.0)

x1_true = np.float32(-0.001)
x2_true = np.float32(-3000.0)

x1, x2 = calc_quad_root_v1(a, b, c)
print("Using 1st formula: approx roots: ", x1, " ", x2)
print(type(x1), type(x2)) # cek apakah x1 dan x2 merupakan np.float32

x1, x2 = calc_quad_root_v2(a, b, c)
print(type(x1), type(x2))
print("Using 2nd formula: approx roots: ", x1, " ", x2)
print("True roots: ", x1_true, " ", x2_true)
```

Bandingkan akar-akar yang Anda peroleh dengan akar-akar eksak. Untuk masing-masing akar, formula mana yang memberikan hasil yang paling dekat dengan hasil eksak?

Soal 4. Program berikut ini, kita akan menggunakan CAS atau *computer algebra system* untuk memastikan bahwa formula (2) dan (3) memberikan hasil yang identik. Lengkapi kode berikut ini dan cek apakah hasil yang diberikan dari kedua formula tersebut adalah sama.

```python
from sympy import *

def calc_quad_root_v1(a, b, c):
  D = b**2 - 4*a*c
  x1 = (-b + sqrt(D))/(2*a)
  x2 = (-b - sqrt(D))/(2*a)
  return x1, x2

def calc_quad_root_v2(a, b, c):
  D = # lengkapi ...
  x1 = # lengkapi ...
  x2 = # lengkapi ...
  return x1, x2

a = Rational(1)
b = Rational(3000001, 1000)
c = Rational(3)

x1_true = -Rational(1, 1000)
x2_true = -3000

x1, x2 = calc_quad_root_v1(a, b, c)
print("Using 1st formula: appprox roots: ", x1, " ", x2)

x1, x2 = calc_quad_root_v2(a, b, c)
print("Using 2nd formula: appprox roots: ", x1, " ", x2)

print("True roots: ", x1_true, " ", x2_true)
```

Perhatikan bahwa kode di atas juga mencetak tipe dari variabel `x1` dan `x2` adalah bilangan integer atau rasional. Pada SymPy, tipe untuk integer dan rasional adalah:

```
<class 'sympy.core.numbers.Integer'> <class 'sympy.core.numbers.Rational'>
```

Coba turunkan formula (3) dari (2).

4 Chapra Contoh 4.4

Diketahui sebuah fungsi:

$$f(x) = -0.1x^4 - 0.15x^3 - 0.5x^2 - 0.25x + 1.25 \tag{4}$$

Kita ingin menghitung pendekatan nilai turunan fungsi ini pada $x = 0.5$ dengan menggunakan tiga formula. Formula pertama adalah beda hingga maju (*forward finite difference*):

$$f'(x_i) \approx \frac{f(x_{i+1}) - f(x_i)}{x_{i+1} - x_i} = \frac{f(x_i + h) - f(x_i)}{h} \tag{5}$$

beda hingga mundur:

$$f'(x_i) \approx \frac{f(x_i) - f(x_{i-1})}{x_i - x_{i-1}} = \frac{f(x_i) - f(x_i - h)}{h} \tag{6}$$

dan beda hingga tengah:

$$f'(x_i) \approx \frac{f(x_{i+1}) - f(x_{i-1})}{x_{i+1} - x_{i-1}} = \frac{f(x_i + h) - f(x_i - h)}{2h} \tag{7}$$

Nilai pendekatan akan dibandingkan dengan hasil evaluasi langsung dari turunan $f(x)$:

$$f'(x) = -0.4x^3 - 0.45x^2 - x - 0.25 \tag{8}$$

Soal 5. Buat program Python untuk menghitung pendekatan nilai $f'(x)$ pada $x = 0.5$ dengan menggunakan $h = 0.5$ dan $h = 0.25$. Bandingkan hasilnya dengan nilai eksak. Formula mana yang memberikan kesalahan paling kecil?

5 Chapra Contoh 4.8

Diketahui sebuah fungsi:

$$f(x) = -0.1x^4 - 0.15x^3 - 0.5x^2 - 0.25x + 1.2 \tag{9}$$

Turunan pertama dari $f(x)$ adalah:

$$f'(x) = -0.4x^3 - 0.45x^2 - x - 0.25 \tag{10}$$

Kita akan menghitung pendekatan fungsi ini pada $x = 0.5$ dengan menggunakan formula beda hingga tengah. Kita akan mulai dari $h = 1$, kemudian secara bertahap memperkecil nilai h dengan faktor 10 untuk mendemonstrasikan bagaimana kesalahan pembulatan (*round-off*) error menjadi dominan.

Soal 6. Berikut ini adalah program yang dapat Anda lengkapi (versi *single precision*).

```
import numpy as np

def f(x):
  return -np.float32(0.1)*x**np.float32(4) - np.float32(0.15)*x**np.float32(3) - \
  np.float32(0.5)*x**np.float32(2) - np.float32(0.25)*x + np.float32(1.2)

def df(x):
  return -np.float32(0.4)*x**np.float32(3) - np.float32(0.45)*x**np.float32(2) - \
  x - np.float32(0.25)

def centered_diff(f, x, h):
  return ( .... )/(np.float32(2)*h) # isi titik-titik

  x = np.float32(0.5)
  h = np.float32(1.0)
  true_val = ! ... lengkapi

  print("---------------------------------------------------------")
  print("           h              approx_val              error")
  print("---------------------------------------------------------")

for i in range(11):
  approx_val = # ... lengkapi
  εt = abs(approx_val - true_val)
  print("%18.10f %18.14f %18.13f" % (h, approx_val, εt))
  h = h/np.float32(10)

print(type(h))
print(type(centered_diff(f,x,h)))
```

Coba juga untuk versi *double precision* (default pada NumPy, atau `np.float64`). Apakah error yang Anda peroleh semakin mendekati nol apabila nilai h semakin diperkecil? NumPy juga menyediakan tipe bilangan *quadruple precision*, yang lebih *precise* dari *double precision*, yaitu `np.float128`. Coba ulangi perhitungan dan analisis Anda dengan menggunakan `np.float128`.

6 Soal tambahan

Beberapa soal latihan dari Chapra.