Solutions to Problem Sets

Chuizheng Kong

October 9, 2024

1

1.1

(1) We begin with $\varepsilon_{\mathrm{m}} = 1$ and repeatedly halve it until the sum $1 + \varepsilon_{\mathrm{m}}$ is numerically equal to 1 at the given precision. The last value before $1 + \varepsilon_{\mathrm{m}}$ equals 1 is considered the machine precision ε_{m}.

(2) Starting with $f_{\min} = 1$, we repeatedly halve it until the value becomes zero at the given precision. The last non-zero value before it underflows to zero is taken as $f_{\min}$.

When we execute the corresponding code, the output is:

```
Machine precision epsilon_m (single precision/np.float32):    1.1920928955078125e-07
Machine precision epsilon_m (double precision/np.float64):    2.220446049250313e-16
Smallest positive number f_min (single precision/np.float32): 1.401298464324817e-45
Smallest positive number f_min (double precision/np.float64): 5e-324
```

The machine precision ε_{m} for single precision (32-bit) is approximately 1.19×10^{-7}. For double precision (64-bit), it is approximately 2.22×10^{-16}. Machine precision indicates the relative accuracy of floating-point arithmetic for each data type. Double precision offers higher accuracy due to the larger number of bits allocated for the mantissa (significant digits).

The smallest positive subnormal number representable in single precision $f_{\min}$ is approximately 1.40×10^{-45}. In double precision, it is approximately 5×10^{-324}. This smallest positive number indicates the underflow limit of the floating-point representation. Numbers smaller than $f_{\min}$ will underflow to zero in computations.

1.2

Let:

$$h_1 = x_2 - x_1 \tag{1.2.1}$$

$$h_2 = x_3 - x_2 \tag{1.2.2}$$

Use Taylor series to expand $f(x)$ around x_2 at x_1 and x_3:

$$f(x_1) = f(x_2 - h_1) = f(x_2) - h_1 f'(x_2) + \frac{h_1^2}{2} f''(x_2) - \frac{h_1^3}{6} f'''(x_2) + \frac{h_1^3}{24} f^{(4)}(x_2) \cdots \tag{1.2.3}$$

$$f(x_3) = f(x_2 + h_2) = f(x_2) + h_2 f'(x_2) + \frac{h_2^2}{2} f''(x_2) + \frac{h_2^3}{6} f'''(x_2) + \frac{h_2^3}{24} f^{(4)}(x_2) \cdots \tag{1.2.4}$$

Let:

$$s_1 = f(x_1) - f(x_2) = -h_1 f'(x_2) + \frac{h_1^2}{2} f''(x_2) - \frac{h_1^3}{6} f'''(x_2) + \frac{h_1^3}{24} f^{(4)}(x_2) \cdots \tag{1.2.5}$$

$$s_2 = f(x_3) - f(x_2) = h_2 f'(x_2) + \frac{h_2^2}{2} f''(x_2) + \frac{h_2^3}{6} f'''(x_2) + \frac{h_2^3}{24} f^{(4)}(x_2) \cdots \tag{1.2.6}$$

$$r_1 = -\frac{h_1^3}{6} f'''(x_2) + \frac{h_1^3}{24} f^{(4)}(x_2) \cdots \tag{1.2.7}$$

$$r_2 = \frac{h_2^3}{6} f'''(x_2) + \frac{h_2^3}{24} f^{(4)}(x_2) \cdots \tag{1.2.8}$$

Now we have 2 equations with 2 unknowns, $f'(x_2)$ and $f''(x_2)$:

$$\begin{cases} -h_1 f'(x_2) + \dfrac{h_1^2}{2} f''(x_2) = s_1 - r_1 \\ \\ h_2 f'(x_2) + \dfrac{h_2^2}{2} f''(x_2) = s_2 - r_2 \end{cases} \tag{1.2.9}$$

Solve for $f'(x_2)$ and $f''(x_2)$:

$$\begin{cases} f'(x_2) = \dfrac{h_1^2(s_2 - r_2) - h_2^2(s_1 - r_1)}{h_1 h_2 (h_1 + h_2)} \\ \\ f''(x_2) = 2\dfrac{h_1(s_2 - r_2) + h_2(s_1 - r_1)}{h_1 h_2 (h_1 + h_2)} \end{cases} \tag{1.2.10}$$

Neglecting the high-order terms r_1 and r_2, we obtain the following estimate:

$$\begin{cases} f'(x_2) \approx \dfrac{h_1^2 s_2 - h_2^2 s_1}{h_1 h_2 (h_1 + h_2)} = \dfrac{h_1^2[f(x_3) - f(x_2)] - h_2^2[f(x_1) - f(x_2)]}{h_1 h_2 (h_1 + h_2)} \\ \\ f''(x_2) \approx 2\dfrac{h_1 s_2 + h_2 s_1}{h_1 h_2 (h_1 + h_2)} = 2\dfrac{h_1[f(x_3) - f(x_2)] + h_2[f(x_1) - f(x_2)]}{h_1 h_2 (h_1 + h_2)} \end{cases} \tag{1.2.11}$$

When the points are equally spaced, with $h_1 = h_2 = h$, the formula simplifies to:

$$\begin{cases} f'(x_2) \approx \dfrac{h_1^2[f(x_3) - f(x_2)] - h_2^2[f(x_1) - f(x_2)]}{h_1 h_2 (h_1 + h_2)} = \dfrac{f(x_3) - f(x_1)}{2h} \\ \\ f''(x_2) \approx 2\dfrac{h_1[f(x_3) - f(x_2)] + h_2[f(x_1) - f(x_2)]}{h_1 h_2 (h_1 + h_2)} = \dfrac{f(x_3) - 2f(x_2) + f(x_1)}{h^2} \end{cases} \tag{1.2.12}$$

These are the standard central difference formulas for the first and second derivatives.

1.3

First, we define the integrand function $f(x)$:

$$f(x) = \frac{1}{\sqrt{0.3(1+x)^3 + 0.7}} \tag{1.3.1}$$

This function encapsulates the dependence of the Hubble parameter $H(z)$ on redshift z. The composite Simpson formula approximates the integral I over the interval $[a, b]$ as:

$$I = \int_a^b f(x)\mathrm{d}x \approx \frac{b-a}{3N}\left[f(x_0) + 4\sum_{i=1,\text{odd}}^{N-1} f(x_i) + 2\sum_{i=2,\text{even}}^{N-2} f(x_i) + f(x_N)\right] \tag{1.3.2}$$

To achieve the desired relative precision $\varepsilon = 10^{-4}$, we adopt an adaptive integration strategy:

1. Start with a small number of intervals, $N = 2$;
2. Compute the integral I_N using composite Simpson formula;
3. Double the number of intervals, $N \to 2N$;
4. Recompute the integral I_{2N} using composite Simpson formula;
5. Estimate the relative error using I_N and I_{2N}:
$$\varepsilon_{2N} \approx \frac{|I_{2N} - I_N|}{15 I_{2N}} \tag{1.3.3}$$
The factor of 15 will be proven later.
6. Check if the relative error is within the desired precision, $\varepsilon_{2N} < \varepsilon$;
7. Repeat steps 3–6 until the desired precision is achieved.

Assume that:
$$I = I_N + E_N = I_{2N} + E_{2N} \tag{1.3.4}$$
Here E_N is the absolute error term in the composite Simpson formula, which is approximately proportional to $\frac{1}{N^4}$. So:
$$E_{2N} \approx \frac{E_N}{16} \tag{1.3.5}$$
Thus, the relative error ε_{2N} can be expressed as:
$$\varepsilon_{2N} = \frac{|I_{2N} - I|}{I} = \frac{|E_{2N}|}{I_{2N} + E_{2N}} \approx \frac{\frac{1}{16}|E_N|}{I_{2N}} = \frac{\frac{15}{16}|E_N|}{15 I_{2N}} = \frac{|E_N| - |E_{2N}|}{15 I_{2N}} = \frac{|E_N - E_{2N}|}{15 I_{2N}} = \frac{|I_{2N} - I_N|}{15 I_{2N}} \tag{1.3.6}$$
We then apply the adaptive integral strategy to each redshift value, the output is:

```
Redshift            z      = 1.0
Co-moving distance  d_c(z) = 3451.60 Mpc
Number of intervals N      = 4
----------------------------------------
Redshift            z      = 3.0
Co-moving distance  d_c(z) = 6640.22 Mpc
Number of intervals N      = 16
----------------------------------------
Redshift            z      = 8.2
Co-moving distance  d_c(z) = 9402.92 Mpc
Number of intervals N      = 32
----------------------------------------
```

1.4

(1) The Cholesky decomposition of a real symmetric positive definite matrix A, is a decomposition of the form:

$$A = LL^{\mathrm{T}} \tag{1.4.1}$$

where L is a real lower triangular matrix with positive diagonal entries:

$$L_{ij} = \begin{cases} \sqrt{A_{ii} - \sum_{k=1}^{i-1} L_{ik}^2}, & \text{if } i = j \\ \frac{1}{L_{jj}} \left(A_{ij} - \sum_{k=1}^{j-1} L_{ik} L_{jk} \right), & \text{if } i > j \end{cases} \tag{1.4.2}$$

To numerically solve $A\boldsymbol{x} = \boldsymbol{b}$ using Cholesky Decomposition, first compute the Cholesky decomposition Equation 1.4.1. Then solve $L\boldsymbol{y} = \boldsymbol{b}$ for $\boldsymbol{y}$ by forward substitution and finally solve $L^{\mathrm{T}}\boldsymbol{x} = \boldsymbol{y}$ for $\boldsymbol{x}$ by backward substitution.

(2) Given that

$$b_i = \sum_{j=1}^{n} A_{ij} = \sum_{j=1}^{n} \frac{1}{i+j-1} \tag{1.4.3}$$

so the exact solution should be

$$\boldsymbol{x}^{\text{exact}} = [1, 1, \ldots, 1]^{\mathrm{T}} \tag{1.4.4}$$

Take $N = 5$ and solve $A\boldsymbol{x} = \boldsymbol{b}$ for both single and double precisions, the output is

```
--------------------------------------------------------------------------------
Data type:          float32
Hilbert matrix size N         = 5
Exact solution      x_exact   = [1. 1. 1. 1. 1.]
Numeric solution    x_numeric = [1.    0.998 1.008 0.989 1.006]
Relative error      err_r     = 0.670%
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Data type:          float64
Hilbert matrix size N         = 5
Exact solution      x_exact   = [1. 1. 1. 1. 1.]
Numeric solution    x_numeric = [1. 1. 1. 1. 1.]
Relative error      err_r     = 0.000%
--------------------------------------------------------------------------------
```

(3) We increment N step by step and monitor the relative error using the 2-norm of the vectors

$$\varepsilon = \frac{||\boldsymbol{x}^{\text{numeric}} - \boldsymbol{x}^{\text{exact}}||_2}{||\boldsymbol{x}^{\text{exact}}||_2} = \frac{\sqrt{(\boldsymbol{x}_1^{\text{numeric}} - \boldsymbol{x}_1^{\text{exact}})^2 + (\boldsymbol{x}_2^{\text{numeric}} - \boldsymbol{x}_1^{\text{exact}})^2 + \cdots + (\boldsymbol{x}_N^{\text{numeric}} - \boldsymbol{x}_N^{\text{exact}})^2}}{\sqrt{(\boldsymbol{x}_1^{\text{exact}})^2 + (\boldsymbol{x}_1^{\text{exact}})^2 + \cdots + (\boldsymbol{x}_N^{\text{exact}})^2}} \tag{1.4.5}$$

For single precision, the output is

```
--------------------------------------------------------------------------------
Data type:          float32
```

```
Hilbert matrix size N         = 5
Exact solution      x_exact   = [1. 1. 1. 1. 1.]
Numeric solution    x_numeric = [1.    0.998 1.008 0.989 1.006]
Relative error      err_r     = 0.670%
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Data type:          float32
Hilbert matrix size N         = 6
Exact solution      x_exact   = [1. 1. 1. 1. 1. 1.]
Numeric solution    x_numeric = [1.001 0.98  1.135 0.654 1.379 0.852]
Relative error      err_r     = 22.505%
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Data type:          float32
Hilbert matrix size N         = 7
Exact solution      x_exact   = [1. 1. 1. 1. 1. 1. 1.]
Numeric solution    x_numeric = [1.001 0.974 1.206 0.32  2.071 0.197 1.231]
Relative error      err_r     = 57.956%
--------------------------------------------------------------------------------
Method becomes unstable at N = 7 for data type float32.
```

For double precision, the output is

```
--------------------------------------------------------------------------------
Data type:          float64
Hilbert matrix size N         = 7
Exact solution      x_exact   = [1. 1. 1. 1. 1. 1. 1.]
Numeric solution    x_numeric = [1. 1. 1. 1. 1. 1. 1.]
Relative error      err_r     = 0.000%
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Data type:          float64
Hilbert matrix size N         = 8
Exact solution      x_exact   = [1. 1. 1. 1. 1. 1. 1. 1.]
Numeric solution    x_numeric = [1. 1. 1. 1. 1. 1. 1. 1.]
Relative error      err_r     = 0.000%
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Data type:          float64
Hilbert matrix size N         = 9
Exact solution      x_exact   = [1. 1. 1. 1. 1. 1. 1. 1. 1.]
Numeric solution    x_numeric = [1. 1. 1. 1. 1. 1. 1. 1. 1.]
Relative error      err_r     = 0.002%
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Data type:          float64
Hilbert matrix size N         = 10
Exact solution      x_exact   = [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
Numeric solution    x_numeric = [1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
Relative error      err_r     = 0.007%
```

```
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Data type:          float64
Hilbert matrix size N         = 11
Exact solution      x_exact   = [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
Numeric solution    x_numeric = [1.    1.    1.    1.    0.997 1.009 0.98  1.027
0.977 1.011 0.998]
Relative error      err_r     = 1.299%
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Data type:          float64
Hilbert matrix size N         = 12
Exact solution      x_exact   = [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
Numeric solution    x_numeric = [1.    1.    1.    1.005 0.966 1.152 0.577 1.767
0.1   1.66  0.726 1.049]
Relative error      err_r     = 41.980%
--------------------------------------------------------------------------------
--------------------------------------------------------------------------------
Data type:          float64
Hilbert matrix size N         = 13
Exact solution      x_exact   = [1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1. 1.]
Numeric solution    x_numeric = [  1.      1.      1.002   0.968   1.289  -0.571
6.499 -11.784  20.954 -19.668  14.623  -4.172   1.861]
Relative error      err_r     = 974.537%
--------------------------------------------------------------------------------
Method becomes unstable at N = 13 for data type float64.
```

(4) The condition number of A using the ∞-norm is

$$\kappa_\infty(A) = ||A^{-1}||_\infty ||A||_\infty \tag{1.4.6}$$

the output is

```
   N |  Data Type | Condition Number (infinity-norm)
-----------------------------------------------------
   3 |    float32 |           7.4800e+02
   6 |    float32 |           2.8125e+07
   9 |    float32 |           5.1737e+09
  12 |    float32 |           3.3679e+09
   3 |    float64 |           7.4800e+02
   6 |    float64 |           2.9070e+07
   9 |    float64 |           1.0997e+12
  12 |    float64 |           3.8536e+16
```

The results from Problem 1.4.4 directly relate to the observations in Problem 1.4.3 by highlighting how the increasing condition number of the Hilbert matrix with larger N leads to greater numerical instability in solving linear systems. The high condition numbers explain why the method becomes unstable at smaller N for single precision compared to double precision. The condition number serves as a key indicator of the potential for significant errors in numerical computations due to the amplification of rounding errors inherent in finite-precision arithmetic.

(5) Here are several strategies to improve the situation, focusing on conceptual approaches:

- Increase numerical precision: utilize extended precision or arbitrary precision arithmetic to reduce rounding errors; leverage software libraries and tools that support higher precision computations beyond standard double precision.

- Use numerically stable algorithms: singular value decomposition (SVD); QR decomposition with pivoting; algorithms based on orthogonal transformations (like Householder reflections)...

- Iterative refinement: After obtaining an initial solution, iteratively correct it by solving for the residual error. Each refinement step involves solving $A\delta\boldsymbol{x} = \boldsymbol{b} - A\boldsymbol{x}$, improving the solution's accuracy.

- Diagonal ccaling: Scale rows and columns to normalize the magnitude of elements, reducing the condition number.