Some Linear Algebra Definitions

Strictly diagonally dominant matrices

A square matrix is said to be diagonally dominant if, for every row of the matrix, the magnitude of the diagonal entry in a row is greater than or equal to the sum of the magnitudes of all the other (off-diagonal) entries in that row. Consider a square matrix $A = \{a_{ij}\}$. A is called strictly diagonal dominant if

$$|a_{ii}| > \sum_{j=1, j\neq i}^{n} |a_{ij}|, \quad i = 1, 2, \cdots, n$$

Properties:

- A is regular, invertible, A^{-1} exists, and $Ax = b$ has a unique solution.
- $Ax = b$ can be solved by Gaussian Elimination without pivoting.

Norms

A norm: measures the "size" of the vector and matrix.

General norm properties: Denote $\|x\|$ the norm of x. Then

(1) $\|x\| \geq 0$, equal if and only if $x = 0$;

(2) $\|ax\| = |a| \cdot \|x\|, \quad a$: is a constant;

(3) $\|x + y\| \leq \|x\| + \|y\|$, triangle inequality.

Examples of vector norms: $x \in \boldsymbol{R}^n$

(1) $\|x\|_1 = \sum_{i=1}^{n} |x_i|$, l_1-norm

(2) $\|x\|_2 = \left(\sum_{i=1}^{n} x_i^2\right)^{1/2}$, l_2-norm

(3) $\|x\|_\infty = \max_{1\leq i\leq n} |x_i|$, l_∞-norm

Matrix norm is defined in term of the corresponding vector norm:

$$\|A\| = \max_{\bar{x}\neq 0} \frac{\|Ax\|}{\|x\|}$$

Properties:

$$\|A\| \geq \frac{\|Ax\|}{\|x\|} \quad \Rightarrow \quad \|Ax\| \leq \|A\| \cdot \|x\|$$
$$\|I\| = 1, \quad \|AB\| \leq \|A\| \cdot \|B\|.$$

Examples of matrix norms:

$$l_1 - \text{ norm } : \|A\|_1 = \max_{1\le j\le n} \sum_{i=1}^{n} |a_{ij}|$$
$$l_2 - \text{ norm } : \|A\|_2 = \max_i |\sigma_i| \,, \quad \sigma_i : \text{ singular values of } A$$
$$l_\infty - \text{ norm } : \|A\|_\infty = \max_{1\le i\le n} \sum_{j=1}^{n} |a_{ij}|$$

Eigenvalues and singular values

Assume A is a square matrix

$Av = \lambda v, \quad \lambda$: eigenvalue, $\quad v$: eigenvector $(A - \lambda I)v = 0, \quad \Rightarrow \quad \det(A - \lambda I) = 0$: polynomial of degree n Property:

$$\lambda_i\left(A^{-1}\right) = \frac{1}{\lambda_i(A)}$$

A singular value of a real matrix A is the positive square root of an eigenvalue of the symmetric matrix AA^T or A^TA. We denote the singular values of A by $\sigma_i(A)$.

Property:

$$\sigma_i\left(A^{-1}\right) = \frac{1}{\sigma_i(A)}$$

where $\sigma_i(A)$ denotes the i-th singular value of A.

This implies:

$$\left\|A^{-1}\right\|_2 = \max_i \left|\sigma_i\left(A^{-1}\right)\right| = \max_i \frac{1}{|\sigma_i(A)|} = \frac{1}{\min_i |\sigma_i(A)|}.$$

Condition number

Want to solve: $\quad Ax = b$

Put some perturbation: $\quad A\bar{x} = b + p$

Relative errors: $e_b = \frac{\|p\|}{\|b\|}, \quad e_x = \frac{\|\bar{x}-x\|}{\|x\|}$

We want to find relation between them.

We have

$$A(\bar{x} - x) = p, \quad \Rightarrow \quad \bar{x} - x = A^{-1}p$$

so

$$e_x = \frac{\|\bar{x} - x\|}{\|x\|} = \frac{\left\|A^{-1}p\right\|}{\|x\|} \le \frac{\left\|A^{-1}\right\| \cdot \|p\|}{\|x\|}.$$

$$Ax = b \quad \Rightarrow \quad \|Ax\| = \|b\| \quad \Rightarrow \quad \|A\|\|x\| \geq \|b\| \quad \Rightarrow \quad \frac{1}{\|x\|} \leq \frac{\|A\|}{\|b\|}$$

we get

$$e_x \leq \frac{\left\|A^{-1}\right\| \cdot \|p\|}{\|x\|} \leq \left\|A^{-1}\right\| \cdot \|p\| \cdot \frac{\|A\|}{\|b\|} = \|A\| \cdot \left\|A^{-1}\right\| e_b = \kappa(A) \cdot e_b,$$
$$\kappa(A) = \|A\| \cdot \left\|A^{-1}\right\| : \quad \text{the condition number of } A$$

Using l_2-norm: $\kappa(A) = \|A\|_2 \cdot \left\|A^{-1}\right\|_2 = \frac{\max_i|\sigma_i(A)|}{\min_i|\sigma_i(A)|}$

Error in b propagates with a factor of $\kappa(A)$ into the solution.

Fixed point iterative solvers for linear systems/Jacobi

Problem: Find approximate solution to $Ax = b$, where $A \in \boldsymbol{R}^{n \times n}$ has properties:

Idea: Avoiding directly computing A^{-1}.

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

Rewrite it: $$\begin{cases} x_1 = \dfrac{1}{a_{11}}(b_1 - a_{12}x_2 - \cdots - a_{1n}x_n) \\ x_2 = \dfrac{1}{a_{22}}(b_2 - a_{21}x_1 - \cdots - a_{2n}x_n) \\ \quad \vdots \\ x_n = \dfrac{1}{a_{nn}}(b_n - a_{n1}x_1 - a_{n2}x_2 - \cdots 0) \end{cases}$$

In a compact form: $x_i = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1, j\neq i}^{n} a_{ij}x_j\right), \quad i = 1, 2, \cdots, n$

This gives the Jacobi iterations:

- Choose a start point, $x^0 = \left(x_1^0, x_2^0, \cdots, x_n^0\right)^t$.
- for $k = 0, 1, 2, \cdots$ until stop criteria, update based on the following formulas:

Element-based formula

The element-based formula for each row i is thus:

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

The computation of $x_i^{(k+1)}$ requires each element in $\mathbf{x}^{(k)}$ except itself.

Matrix-based formula

Then A can be decomposed into a diagonal component D, a lower triangular part L and an upper triangular part U :

$$A = D + L + U \quad \text{where} \quad D = \begin{bmatrix} a_{11} & 0 & \cdots & 0 \\ 0 & a_{22} & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & a_{nn} \end{bmatrix} \text{ and } L + U$$
$$= \begin{bmatrix} 0 & a_{12} & \cdots & a_{1n} \\ a_{21} & 0 & \cdots & a_{2n} \\ \vdots & \vdots & \ddots & \vdots \\ a_{n1} & a_{n2} & \cdots & 0 \end{bmatrix}.$$

The solution is then obtained iteratively via

$$\mathbf{x}^{(k+1)} = D^{-1}\left(\mathbf{b} - (L+U)\mathbf{x}^{(k)}\right).$$

Jacobi Example

Problem Statement

Consider the system of linear equations:

$$\begin{cases} 4x_1 - x_2 + x_3 = 7 \\ -x_1 + 3x_2 + 2x_3 = 8 \\ x_1 + 2x_2 + 5x_3 = 12 \end{cases}$$

We want to find the values of x_1, x_2, and x_3.

Step-by-Step Jacobi Iteration

1. **Rewrite the system** in terms of x_1, x_2, and x_3:

$$\begin{cases} x_1 = \frac{7 + x_2 - x_3}{4} \\ x_2 = \frac{8 + x_1 - 2x_3}{3} \\ x_3 = \frac{12 - x_1 - 2x_2}{5} \end{cases}$$

1. **Initial guess**: Start with an initial guess for the variables, say $x_1^{(0)} = 0$, $x_2^{(0)} = 0$, and $x_3^{(0)} = 0$.

2. **Iterate** using the Jacobi formula until convergence. For iteration k, the new values $x_1^{(k+1)}$, $x_2^{(k+1)}$, and $x_3^{(k+1)}$ are computed as:

$$\begin{cases} x_1^{(k+1)} = \frac{7+x_2^{(k)}-x_3^{(k)}}{4} \\ x_2^{(k+1)} = \frac{8+x_1^{(k)}-2x_3^{(k)}}{3} \\ x_3^{(k+1)} = \frac{12-x_1^{(k)}-2x_2^{(k)}}{5} \end{cases}$$

1. **Perform the iterations**:

   - **Iteration 1**:

$$\begin{cases} x_1^{(1)} = \frac{7+0-0}{4} = \frac{7}{4} = 1.75 \\ x_2^{(1)} = \frac{8+0-2\cdot 0}{3} = \frac{8}{3} \approx 2.67 \\ x_3^{(1)} = \frac{12-0-2\cdot 0}{5} = \frac{12}{5} = 2.40 \end{cases}$$

   - **Iteration 2**:

$$\begin{cases} x_1^{(2)} = \frac{7+2.67-2.40}{4} = \frac{7+0.27}{4} = \frac{7.27}{4} \approx 1.82 \\ x_2^{(2)} = \frac{8+1.75-2\cdot 2.40}{3} = \frac{8+1.75-4.80}{3} = \frac{4.95}{3} \approx 1.65 \\ x_3^{(2)} = \frac{12-1.75-2\cdot 1.65}{5} = \frac{12-1.75-3.30}{5} = \frac{6.95}{5} \approx 1.39 \end{cases}$$

   - **Iteration 3**:

     Continue the process until the values converge to a stable solution.

Gauss-Seidal iterations

Recall Jacobi iteration

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

If the computation is done in a sequential way, for $i = 1, 2, \cdots$, then in the first summation term, all x_j^k are already computed for step $k+1$. We will replace all these x_j^k with x_j^{k+1}.

Use the latest computed values of x_i.

$$\begin{array}{l} \text{for } k = 0, 1, 2, \cdots, \text{ until stop criteria} \\ \quad \text{for } i = 1, 2, \cdots, n \\ \quad\quad x_i^{k+1} = \frac{1}{a_{ii}}\left(b_i - \sum_{j=1}^{i-1} a_{ij}x_j^{k+1} - \sum_{j=i+1}^{n} a_{ij}x_j^k\right) \\ \quad \text{end} \\ \text{end} \end{array}$$

SOR (Successive Over Relaxation)

SOR is a more general iterative method. A version based on Gauss-Seidal.

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

Note the second term is the Gauss-Seidal iteration multiplied with w.

w : relaxation parameter. Usual value: $0 < w < 2$ (for convergence reason)

- $w = 1$: Gauss-Seidal
- $0 < w < 1$: under relaxation
- $1 < w < 2$: over relaxation

Update Rules

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

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

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

Iterative Solvers

Want to solve $Ax = b$. We change it into a fixed-point problem $x = Mx + y$ for some matrix M and vector y, with fixed point iteration $x^{k+1} = Mx^k + y$.

Splitting of the matrix A :

$$A = L + D + U$$

Now we have

$$Ax = (L + D + U)x = Lx + Dx + Ux = b$$

Jacobi iterations:

$$Dx^{k+1} = b - Lx^k - Ux^k$$

so

$$x^{k+1} = D^{-1}b - D^{-1}(L+U)x^k = y_J + M_J x^k$$

where

$$y_J = D^{-1}b, \quad M_J = -D^{-1}(L+U).$$

Gauss-Seidal:

$$Dx^{k+1} + Lx^{k+1} = b - Ux^k$$

so

$$x^{k+1} = (D+L)^{-1}b - (D+L)^{-1}Ux^k = y_{GS} + M_{GS}x^k$$

where

$$y_{GS} = (D+L)^{-1}b, \quad M_{GS} = -(D+L)^{-1}U.$$

SOR:

$$\begin{aligned} x^{k+1} &= (1-w)x^k + wD^{-1}\left(b - Lx^{k+1} - Ux^k\right) \\ \Rightarrow \quad Dx^{k+1} &= (1-w)Dx^k + wb - wLx^{k+1} - wUx^k \\ \Rightarrow \quad (D+wL)x^{k+1} &= wb + [(1-w)D - wU]x^k \end{aligned}$$

SO

$$x^{k+1} = (D+wL)^{-1}b + (D+wL)^{-1}[(1-w)D - wU]x^k = y_{\mathrm{SOR}} + M_{\mathrm{SOR}}\, x^k$$

where

$$y_{SOR} = (D+wL)^{-1}b, \quad M_{SOR} = (D+wL)^{-1}[(1-w)D - wU].$$

Iterative Solvers. Convergence Analysis

Iteration $\quad x^{k+1} = y + Mx^k$ for solving $Ax = b$

Assume s is the solution: $\quad As = b, \quad s = y + Ms.$

Define the error vector: $\quad e^k = x^k - s$

$$e^{k+1} = x^{k+1} - s = y + Mx^k - (y + Ms) = M\left(x^k - s\right) = Me^k.$$

$$e^{k+1} = x^{k+1} - s = y + Mx^k - (y + Ms) = M\left(x^k - s\right) = Me^k.$$

This gives the propagation of error:

$$e^{k+1} = Me^k.$$

Take the norm on both sides:

$$\left\|e^{k+1}\right\| = \left\|Me^k\right\| \leq \|M\| \cdot \left\|e^k\right\|$$

This implies:

$$\left\|e^k\right\| \leq \|M\|^k \left\|e^0\right\|, \quad e^0 = x^0 - s$$

Theorem If $||M|| < 1$ for some norm $|| \cdot ||$, then the iterations converge in that norm.

Convergence Theorem. If A is diagonal dominant, i.e.,

$$|a_{ii}| > \sum_{j=1, j\neq i}^{n} |a_{ij}|, \quad \text{for every } i = 1, 2, \cdots, n.$$

Then, all three iteration methods converge for all initial choice of x^0.

Numerical solutions for ODEs. Introduction.

Definition of ODE: an equation which contains one or more ordinary derivatives of an unknown function.

Example 1. Let $x = x(t)$ be the unknown function of t, ODE examples can be

$$x' = x^2, \quad x'' + x \cdot x' + 4 = 0, \quad \text{etc.}$$

We consider the initial-value problem for first-order ODE

$$\begin{cases} x' = f(t, x), & \text{differential equation} \\ x\left(t_0\right) = x_0 & \text{given initial condition (IC)} \end{cases}$$

Some examples: Some examples:

$$\begin{aligned} x'(t) = 2, \quad x(0) = 0. \quad & \text{solution:} \quad x(t) = 2t. \\ x'(t) = 2t, \quad x(0) = 0. \quad & \text{solution:} \quad x(t) = t^2. \\ x'(t) = x + 1, \quad x(0) = 0. \quad & \text{solution:} \quad x(t) = e^t - 1. \end{aligned}$$

In many situations, exact solutions can be very difficult/impossible to obtain.

We seek approximate values of the solution at discrete sampling points. Uniform grid for time variable. Let h be the time step length

$$t_{k+1} - t_k = h, \quad t_k = t_0 + kh, \quad t_0 < t_1 < \cdots < t_N.$$

Given an ODE, and a final computing time t_N. We seek values $x_n \approx x\left(t_n\right), n = 1, 2, \cdots, N$, and $t_0 < t_1 < \cdots < t_N$.

Taylor series methods for ODEs

Given

$$x'(t) = f(t, x(t)), \quad x\left(t_0\right) = x_0.$$

Let $t_1 = t_0 + h$. Let's find the value $x_1 \approx x\left(t_1\right) = x\left(t_0 + h\right)$. Taylor expansion of $x\left(t_0 + h\right)$ expand at t_0 gives

$$x\left(t_0 + h\right) = x\left(t_0\right) + hx'\left(t_0\right) + \frac{1}{2}h^2 x''\left(t_0\right) + \cdots = \sum_{m=0}^{\infty} \frac{1}{m!} h^m x^{(m)}\left(t_0\right)$$

We take the first $(m+1)$ terms in Taylor expansion.

$$x\left(t_0 + h\right) \approx x\left(t_0\right) + hx'\left(t_0\right) + \frac{1}{2}h^2 x''\left(t_0\right) + \cdots + \frac{1}{m!} h^m x^{(m)}\left(t_0\right).$$

Error in each step:

$$x\left(t_0 + h\right) - x_1 = \sum_{k=m+1}^{\infty} \frac{1}{k!} h^k x^{(k)}\left(t_0\right) = \frac{1}{(m+1)!} h^{m+1} x^{(m+1)}(\xi)$$

for some $\xi \in \left(t_0, t_1\right)$.

For $m=1$:

$$x_1=x_0+hx'(t_0)=x_0+h\cdot f(t_0,x_0)$$

This is called forward Euler step.

General formula for step number k :

$$x_{k+1}=x_k+h\cdot f(t_k,x_k), \quad k=0,1,2,\cdots N-1$$

For $m=2$:

$$x_1=x_0+hx'(t_0)+\frac{1}{2}h^2x''(t_0)$$

Computing x'' :

$$x''=\frac{d}{dt}x'(t)=\frac{d}{dt}f(t,x(t))=f_t+f_x\cdot x'$$

we get

$$x_1=x_0+hf(t_0,x_0)+\frac{1}{2}h^2\left[f_t(t_0,x_0)+f_x(t_0,x_0)\cdot f(t_0,x_0)\right]$$

For general step $k, k=0,1,2,\cdots N-1$, we have

$$x_{k+1}=x_k+hf(t_k,x_k)+\frac{1}{2}h^2\left[f_t(t_k,x_k)+f_x(t_k,x_k)\cdot f(t_k,x_k)\right]$$

Examples

Example 1. Set up Taylor series methods with $m=1$ and $m=2$ for

$$x'=-x+e^{-t}, \quad x(0)=0.$$

The exact solution is $x(t)=te^{-t}$. Answer. The initial data gives $t_0=0, x_0=0$. For $m=1$, we have

$$x_{k+1}=x_k+h\left(-x_k+e^{-t_k}\right)=(1-h)x_k+he^{-t_k}$$

For $m=2$, we have

$$x''=\left(-x+e^{-t}\right)'=-x'-e^{-t}=x-e^{-t}-e^{-t}=x-2e^{-t}$$

so

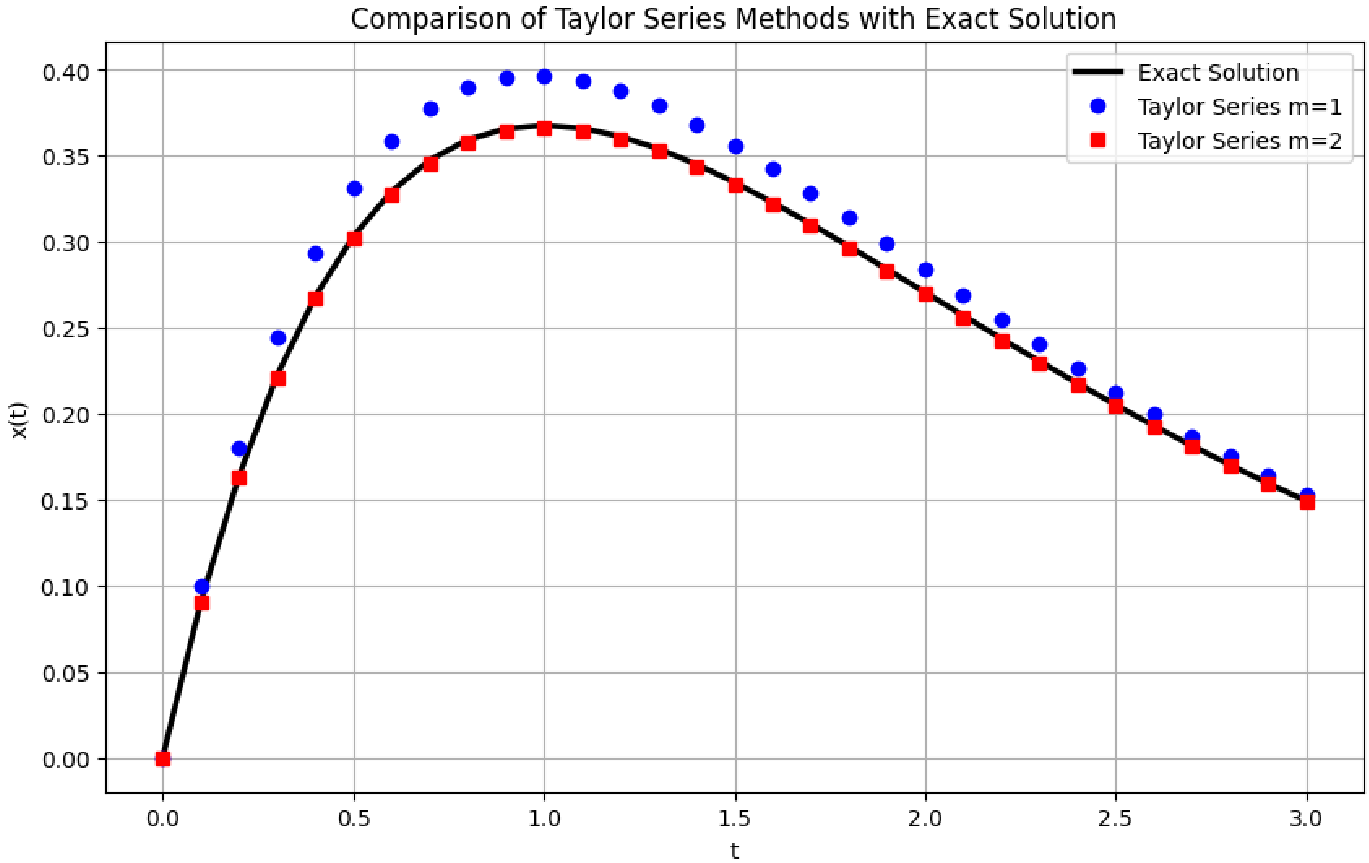
$$\begin{aligned} x_{k+1} &= x_k+hx'_k+0.5h^2x''_k \\ &= x_k+h\left(-x_k+e^{-t_k}\right)+0.5h^2\left(x_k-2e^{-t_k}\right) \\ &= \left(1-h+0.5h^2\right)x_k+\left(h-h^2\right)e^{-t_k} \end{aligned}$$

We take the first $(m+1)$ terms in Taylor expansion.

$$x(t_0+h)\approx x(t_0)+hx'(t_0)+\frac{1}{2}h^2x''(t_0)+\cdots+\frac{1}{m!}h^mx^{(m)}(t_0).$$

In [ ]:

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

# Define the exact solution
def exact_solution(t):
    return t * np.exp(-t)

# Define the differential equation and its derivatives
def f(t, x):
    return -x + np.exp(-t)

def f_prime(t, x):
    return x - 2 * np.exp(-t)

# Implement the Taylor series method with m=1
def taylor_m1(t0, x0, h, n):
    t_values = [t0]
    x_values = [x0]

    for i in range(n):
        t_next = t_values[-1] + h
        x_next = x_values[-1] + h * f(t_values[-1], x_values[-1])
        t_values.append(t_next)
        x_values.append(x_next)

    return t_values, x_values

# Implement the Taylor series method with m=2
def taylor_m2(t0, x0, h, n):
    t_values = [t0]
    x_values = [x0]

    for i in range(n):
        t_next = t_values[-1] + h
        x_next = x_values[-1] + h * f(t_values[-1], x_values[-1]) + 0.5 * h**2 * f_prime
        t_values.append(t_next)
        x_values.append(x_next)

    return t_values, x_values

# Initial conditions and parameters
t0 = 0
x0 = 0
h = 0.1
n = 30

# Compute the numerical solutions
t_values_m1, x_values_m1 = taylor_m1(t0, x0, h, n)
t_values_m2, x_values_m2 = taylor_m2(t0, x0, h, n)
t_exact = np.linspace(t0, t0 + n*h, n+1)
x_exact = exact_solution(t_exact)

# Plot the results
plt.figure(figsize=(10, 6))
plt.plot(t_exact, x_exact, linewidth=2.5, label='Exact Solution', color='black')
plt.plot(t_values_m1, x_values_m1, 'o',linewidth=0.5, label='Taylor Series m=1', color='
plt.plot(t_values_m2, x_values_m2, 's',linewidth=0.5, label='Taylor Series m=2', color='
plt.xlabel('t')
plt.ylabel('x(t)')
plt.title('Comparison of Taylor Series Methods with Exact Solution')
plt.legend()
plt.grid(True)
plt.show()
```



Example 2. Set up Taylor series methods with $m=1,2,3,4$ for

$$x^{\prime}=x, \quad x(0)=1.$$

The exact solution is $x(t)=e^{t}$. Answer. We set $t_{0}=0, x_{0}=1$. Note that

$$x^{\prime \prime}=x^{\prime}=x, \quad x^{\prime \prime \prime}=x^{\prime \prime}=x, \quad \cdots \quad x^{(m)}=x$$

Taylor series method of order m :

$$x_{k+1}=x_{k}+h x_{k}+h^{2} x_{k} / 2+\cdots+h^{m} x_{k} /(m!)$$

So

$$\begin{aligned}
m=1: & \quad x_{k+1}=x_{k}+h x_{k}=(1+h) x_{k} \\
m=2: & \quad x_{k+1}=x_{k}+h x_{k}+h^{2} x_{k} / 2=\left(1+h+h^{2} / 2\right) x_{k} \\
m=3: & \quad x_{k+1}=x_{k}+h x_{k}+h^{2} x_{k} / 2+h^{3} x_{k} / 6=\left(1+h+h^{2} / 2+h^{3} / 6\right) x_{k} \\
m=4: & \quad x_{k+1}=\cdots=\left(1+h+h^{2} / 2+h^{3} / 6+h^{4} / 24\right) x_{k}
\end{aligned}$$

Error analysis for Taylor Series Methods

Given ODE

$$x^{\prime}=f(t, x), \quad x\left(t_{0}\right)=x_{0}.$$

Local error (error in each time step) for Taylor series method of order m.

Let t_{k}, x_{k} be given,

let x_{k+1} be the numerical solution after one iteration,

and let $x\left(t_k+h\right)$ be the exact solution for the IVP

$$x'=f(t,x), \quad x\left(t_k\right)=x_k.$$

Then, the local truncation error is define as

$$e_k \doteq \left|x_{k+1}-x\left(t_k+h\right)\right|.$$

Theorem: For Taylor series method of order m at step k, the local error is of order $m+1$, i.e., $e_k \leq M h^{m+1}$ for some bounded constant M.

Proof. We have

$$e_k=\left|x_{k+1}-x\left(t_k+h\right)\right|=\frac{h^{m+1}}{(m+1)!}\left|x^{(m+1)}(\xi)\right|=\frac{h^{m+1}}{(m+1)!}\left|\frac{d^m f}{d t^m}(\xi, x(\xi))\right|,$$

for some $\xi \in\left(t_k, t_{k+1}\right)$.

We assume

$$\left|\frac{d^m f}{d t^m}\right| \leq M$$

Now we have

$$e_k \leq \frac{M}{(m+1)!} h^{m+1}=\mathcal{O}\left(h^{m+1}\right)$$

Definition. The ODE $x'=f(t,x)$ is called well-posed if it is stable w.r.t. perturbations in initial data. This means, let $x(t)$ and $\tilde{x}(t)$ be the solutions with two different initial conditions $x\left(t_0\right)=x_0$ and $\tilde{x}\left(t_0\right)=\tilde{x}_0$. Fix a final time T. Then, there exists a constant C, independent of t, such that

$$|x(t)-\tilde{x}(t)| \leq C\left|x_0-\tilde{x}_0\right|.$$

Total error is the error at the final computing time T.

Let

$$N=\frac{T}{h}, \quad \text{i.e.,} \quad T=N h.$$

The total error is defined as

$$E \doteq\left|x(T)-x_N\right|$$

Theorem: Assume that the ODE is well-posed. If the local error of a numerical iteration satisfies

$$e_k \leq M h^{m+1}$$

then the total error satisfies

$$E \leq C h^m$$

for some bounded constant C, where C is uniform in t.

Proof. We observe two facts about the errors. First, at every step k, the local error is being carried on through the rest of the simulation. Second, the local errors accumulate through time iteration steps. By the well-posedness assumption, at each time step k, the local error e_k is amplified at most by a factor of C in the answer at the final time T.

We can add up all the accumulated errors at T caused by all the local errors

$$\begin{aligned} E &= C\sum_{k=1}^{N}\left|e_L^{(k)}\right| \le C\sum_{k=1}^{N}\frac{M}{(m+1)!}h^{m+1} \\ &= CN\frac{M}{(m+1)!}h^{m+1} = C(Nh)\frac{M}{(m+1)!}h^{m} = \frac{CMT}{(m+1)!}h^{m} = \mathcal{O}\left(h^{m}\right) \end{aligned}$$

Therefore, the method is of order m.

Lecture note based on Intro Numeric Comput (2nd Ed): Wen Shen: 9789811204418