# 1.

## (a)

**Algorithm 1:** Solving $\mathbf{Ax} = \mathbf{b}$ by using the Cholesky Decomposition Algorithm

```
Require: A is symmetric positive definite
Input: Matrix A [a_ij] of n × n and matrix b [b_ij] of n × m
Output: Solution of equations Ax = b , matrix x [x_ij] of n × m
Function Solve(A, b):
    /*Applying Cholesky Decomposition Algorithm to calculate matrix C first*/
    for i ← 1 to n do
        for j ← i+1 to n do
            c[ij] ← 0;
        end
    end
    for j ← 1 to n do
        c_jj ← √(a_jj);
        for i ← j+1 to n do
            c_ij ← a_ij/c_jj;
            for k ← j to i do
                a_ik ← a_ik − c_ij · c_kj
            end
        end
    end
    /*Solving equation Cy = b*/
    for i ← 1 to n do
        for j ← 1 to i - 1 do
            b_i ← b_i - c_ij· y_j;            /*y_i, b_i denote the i^th row of y, b*/
        end
        y_i ← b_i/c_ii;
    end
    /*Solving equation C^T x = y*/
    for i ← n to 1 step -1 do
        for j ← i + 1 to n do
            y_i ← y_i - c_ji· x_j;            /*x_i denote the i^th row of x*/
        end
        x_i ← y_i/c_ii;
    end
    return x;
end
```

Jiang Yicheng
515370910224

### (b)

```
function [x] = Gauss_Solver(A, b)
    [n, m] = size(b);
    e(1) = 0; f(1) = A(1, 1); g(1) = A(1, 2);
    e(n) = A(n, n-1); f(n) = A(n, n); g(n) = 0;
    for i = 2 : n - 1
        e(i) = A(i, i-1); f(i) = A(i, i); g(i) = A(i, i+1);
    end
    for i = 2 : n
        q = e(i)/f(i-1);
        f(i) = f(i) - q * g(i-1);
        for k = 1 : m
           b(i, k) = b(i, k) - q * b(i - 1, k);
        end
    end
    for k = 1 : m
       x(n, k) = b(n, k)/f(n);
       for i = n - 1 : -1 : 1
          x(i, k) = (b(i, k) - g(i) * x(i+1, k))/f(i);
       end
    end
end
```

```
function [x] = Choleksy(a, b)
    [n, m] = size(b); c(n, n) = 0;
    for j = 1 : n
        c(j, j) = sqrt(a(j, j));
        for i = j + 1 : n
            c(i, j) = a(i, j)/c(j, j);
            for k = j : i
                a(i, k) = a(i, k) - c(i, j) * c(k, j);
            end
        end
    end
    x(n, m) = 0; y(n, m) = 0;
    for i = 1 : n
       for j = 1 : i - 1
           for k = 1 : m
              b(i, k) = b(i, k) - c(i, j) * y(j, k);
           end
       end
       for k = 1 : m
          y(i, k) = b(i, k)/c(i, i);
       end
    end
    for i = n : -1 : 1
       for j = i + 1 : n
           for k = 1 : m
              y(i, k) = y(i, k) - c(j, i) * x(j, k);
           end
       end
       for k = 1 : m
          x(i, k) = y(i, k)/c(i, i);
       end
    end
end
```

Jiang Yicheng
515370910224

```matlab
for i = 1 : 3
    N = 10 * 10^i;
    z = [2 1 zeros(1, N - 2 )] ;
    A = toeplitz(z, z);
    b = ones(N, 1);
    disp("For N = " + N + ", Tridiagonal method: ");
    tic; Gauss_Solver(A, b); toc;
    disp("For N = " + N + ", Choleksy method: ")
    tic; Choleksy(A, b); toc;
    disp("For N = " + N + ", Backslash method: ")
    tic; A\b; toc;
end
```

```
For N = 100, Tridiagonal method:
时间已过 0.001796 秒。
For N 100, Choleksy method:
时间已过 0.005401 秒。
For N 100, Backslash method:
时间已过 0.002063 秒。
For N = 1000, Tridiagonal method:
时间已过 0.001050 秒。
For N 1000, Choleksy method:
时间已过 0.622900 秒。
For N 1000, Backslash method:
时间已过 0.013007 秒。
For N = 10000, Tridiagonal method:
时间已过 0.003580 秒。
For N 10000, Choleksy method:
时间已过 1357.763879 秒。
For N 10000, Backslash method:
时间已过 2.747848 秒。
```

| | n | 100 | 1000 | 10000 |
|---|---|---|---|---|
| Elapsed time (in seconds) | Tridiagonal | 0.001796 | 0.005401 | 0.002063 |
| | Choleksy | 0.001050 | 0.622900 | 0.013007 |
| | Backslash | 0.003580 | 1357.763879 | 2.747848 |

# 2.

**Algorithm 2:** Finding the eignvalue of a given matrix $A$ of $n \times n$

**Require:** $\mathbf{A}$ is symmetric positive definite
**Input:** Matrix $\mathbf{A}$ of $n \times n$
**Output:** eignvalues of a given matrix $\mathbf{A}$

```
Function eignvalue(A):
    while true do
        Q, R ← HHQR(A);
        A1 ← RQ;
        sum ← 0;
        for i ← 2 to n do
            for j ← 1 to i - 1 do
                sum ← sum + |A1(i, j)|; /*sum of elements in lower triangle part of A_1*/
            end
        end
        if sum < 1 × 10^{-5} then
            break;
        end
        A ← A1;
    end
    return diag(A);
end
Function HHQR(A_{n×n}):
    Q ← I_{n×n};
    R ← A;
    for j ← 1 to n - 1 do
        a_1 ← R[1 : (j − 1), j];
        a_2 ← R[j : n, j];
        c ← sign(R(j, j)) · ||a_2||;
        v[1 : (j − 1)] ← 0_{(j−1)×1};
        v[j : n] ← a_2;
        v ← v − c e_j;
        H ← I_{n×n} − 2 v v^T / (v^T v);
        R ← HR;
        Q ← QH;
    end
    return (Q, R);
end
```

```
function [lambda] = eignvalue(A)
    [n, ~] = size(A);
    while 1
        [Q, R] = HHQR(A);
        A1 = R * Q;
        sum = 0;
        for i = 2 : n
            for j = 1 : i - 1
                sum = sum + abs(A1(i,j));
            end
        end
        if sum < 1e-5
            break;
        end
        A = A1;
    end
    lambda = diag(A);
end

function [Q, R] = HHQR(A)
    n = size(A, 1);
    I = eye(n);
    Q = I;
    R = A;
    for j = 1 : 1 : (n - 1)
        a2 = R(j : n, j);
        c = sign(R(j, j)) * norm(a2);
        v(1 : (j - 1),1) = zeros(j - 1   , 1);
        v(j : n, 1) = a2;
        v = v - c * I(:, j);
        H = I - 2 * (v * v')/(v' * v);
        R = H * R;
        Q = Q * H;
    end
end
```

eignvalue.m × HW4_2.m × +

```
N = 100;
rng(471) ;
X = randn(N) ;
A = transpose(X) * X;
lambda = sort(eignvalue(A));
eignvalue = sort(eig(A));
error = max(lambda - eignvalue);
```

```
>> HW4_2
>> error

error =

   1.5888e-10
```

**Algorithm 3:** Computing eignvector of matrix $\mathbf{A}$

**Require:** $\mathbf{A}$ has $n$ different eignvalues
**Input:** Matrix $\mathbf{A}$ of $n \times n$, and its eignvalue $\lambda$
**Output:** eignvector of $\mathbf{A}$ corresponding to each eignvalue

**Function** `eignvector` *($\mathbf{A}, \lambda$)*:
  **for** $m \leftarrow 1$ **to** $n$ **do**
    $\mathbf{a} \leftarrow \mathbf{A} - \lambda_m \cdot \mathbf{I}_{n\times n}$;
    **for** $k \leftarrow 1$ **to** $n$ - $1$ **do**
      /*Try to find a row in which $a_{lk} \neq 0$, where $a_{lk}$ denotes the element in matrix $\mathbf{a}$*/
      $l \leftarrow k$;
      **while** $l <= n$ **do**
        **if** $a_{lk} = 0$ **then**
          $l \leftarrow l + 1$;
        **else**
          break;
        **end**
      **end**
      **if** $l > n$ **then**
        **continue**;/*If fail, we have done Gauss elimination for $k^{th}$ column*/
      **else**
        swap$(a_l, a_k)$/*Exhange row $k$ with row $l$ to make sure $a_{kk} \neq 0$*/
      **end**
      **for** $i \leftarrow k + 1$ **to** $n$ - $1$ **do**
        $m \leftarrow a_{ik}/a_{kk}$;
        $a_{ik} \leftarrow 0$;
        **for** $j \leftarrow k + 1$ **to** $n$ **do**
          $a_{ij} \leftarrow a_{ij} - m \cdot a_{kj}$;
        **end**
      **end**
    **end**
    $\mathbf{v_m} \leftarrow 0$; /*$\mathbf{v_m}$ denotes the m$^{\text{th}}$ column of matrix $\mathbf{V}$*/
    $v(n, m) \leftarrow 1$; /*Only the last row of $\mathbf{A}$ would contain only zeros*/
    **for** $i \leftarrow n$ - $1$ **to** $1$ ***step*** -$1$ **do**
      $y_i \leftarrow 0$;
      **for** $j \leftarrow i + 1$ **to** $n$ **do**
        $y_i \leftarrow y_i$ - $a_{ji} \cdot v_{jm}$; /*$\mathbf{x_i}$ denote the i$^{th}$ row of $\mathbf{x}$*/
      **end**
      $v_{im} \leftarrow y_i/a_{ii}$;
    **end**
  **end**
  **return** $\mathbf{V}$; /*each column of matrix $\mathbf{V}$ is an eignvector of $A$*/
**end**

# 3.

## (a)

For any $(t, y_1), (t, y_2) \in \mathcal{A}$, $y_1 < y_2$, since $\mathcal{A}$ is convex, $\forall y \in (y_1, y_2)$, $(t, y) \in \mathcal{A}$. Since $\Phi(t, y)$ is differentiable on $y$A, according to mean value theorem, $\forall (t, y_1), (t, y_2) \in \mathcal{A}$, $y_1 < y_2$, $\exists \xi \in (y_1, y_2)$, such that

$$\left.\frac{\partial \Phi(t, y)}{\partial y}\right|_{y=\xi} = \frac{\Phi(t, y_1) - \Phi(t, y_2)}{y_1 - y_2}$$

since $(t, \xi) \in \mathcal{A}$,

$$\left| \left.\frac{\partial \Phi(t, y)}{\partial y}\right|_{y=\xi} \right| \leqslant c$$

and therefore, $\forall (t, y_1), (t, y_2) \in \mathcal{A}$,

$$\left| \frac{\Phi(t, y_1) - \Phi(t, y_2)}{y_1 - y_2} \right| \leqslant c$$

so $\Phi$ satisfies a Lipschitz condition in $y$ on $\mathcal{A}$ with Lipschitz constant $c$.

## (b)

For all $(t_1, y_1), (t_2, y_2) \in \mathcal{D}$, $(\lambda t_1 + (1-\lambda)t_2, \lambda y_1 + (1-\lambda)y_2)$, $\lambda \in [0, 1]$ denotes a point on line segment joining these two points. Since $y_1, y_2 \in \mathbb{R}$, $\lambda y_1 + (1-\lambda)y_2) \in \mathbb{R}$. Since $t_1, t_2 \in [t_0, T]$,

$$\lambda t_1 + (1-\lambda)t_2 \in [\min\{t_1, t_2\}, \max\{t_1, t_2\}] \subset [t_0, T]$$

so $(\lambda t_1 + (1-\lambda)t_2, \lambda y_1 + (1-\lambda)y_2) \in \mathcal{D}$ holds for any $\lambda \in [0, 1]$. And therefore, $\mathcal{D}$ is convex.

## (c)

Define $\mathcal{B} = \{(t, y) | 0 \leqslant t \leqslant 1, y \in \mathbb{R}\}$, it is convex. Define $\Phi(t, y) = \dfrac{4t^3 y}{1 + t^4}$, for all $(t, y) \in \mathcal{B}$,

$$\left| \frac{\partial \Phi(t, y)}{\partial y} \right| = \frac{4t^3}{1 + t^4} \leqslant 4$$

and therefore $\Phi$ satisfies a Lipschitz condition in $y$ on $\mathcal{B}$ with Lipschitz constant $c = 4$. So $\dot{y} = \Phi(t, y)$ has a unique solution.

## (d)

We can solve the differential equation analytically

$$\dot{y} = 1 + y^2,\ y(0) = 0 \Rightarrow \int_0^y \frac{1}{1 + u^2} du = \int_0^t 1 dx$$
$$\Rightarrow \arctan(y) = t$$

so we can see that for $t \geqslant \frac{\pi}{2}$, the ODE has no solution. So we cannot find the solution for $0 \leqslant t \leqslant 3$ numerically.

If we use Euler's forward method,

$$\hat{y}(t_{i+1}) = \hat{y}(t_i) + h \cdot (1 + \hat{y}(t_i)^2)$$

it will not return our such kind of information. It seems to work, while actually, it has failed.

# 4.

## (a)

Define $\mathcal{A} = \{(t, y) | t_0 \leqslant t \leqslant T, y \in \mathbb{R}\}$, it is convex. Define $\Phi(t, y) = \arctan(y)$, for all $(t, y) \in \mathcal{A}$,

$$\left| \frac{\partial \Phi(t, y)}{\partial y} \right| = \frac{1}{1 + y^2} \leqslant 1$$

and therefore $\Phi$ satisfies a Lipschitz condition in $y$ on $\mathcal{A}$ with Lipschitz constant $c = 1$, i.e. a Lipschitz constant for $\arctan(y)$ is $c = 1$.

## (b)

Take derivative on the ODE,

$$|\ddot{y}| = \left| \frac{d \arctan(y)}{dx} \right| = \frac{1}{1 + y^2} \cdot |\dot{y}| = \frac{|\arctan(y)|}{1 + y^2} \leqslant \frac{\pi}{2}$$

So an upper bound on $|\ddot{y}|$ is $\frac{\pi}{2}$.

## (c)

For the local discretisation error,

$$\begin{aligned}
|\tau_k| =& |y(t_k) - (y(t_{k-1}) + h \cdot \arctan(y(t_{k-1})))| \\
\leqslant& |y(t_k) - y(t_{k-1})| + h \cdot |\arctan(y(t_{k-1}))| \\
\leqslant& 1 \cdot |t_k - t_{k-1}| + h \cdot \frac{\pi}{2} \\
=& (1 + \frac{\pi}{2}) h
\end{aligned}$$

then

$$|e_k| \leqslant \frac{|\tau^*|}{h \cdot 1} (e^{1 \cdot (t_k - t_0)} - 1) \leqslant (1 + \frac{\pi}{2})(e^{kh} - 1)$$

where $h$ is the step size and $k$ is the step number.

## 5.

### (a)

```
function [t, y] = Euler_f(Phi, t0, T, y0, n)
    f = inline(Phi, 't', 'y');
    h = (T - t0)/n;
    t(1) = t0;
    y(1) = y0;
    for i = 2 : n + 1
        t(i) = t(i - 1) + h;
        y(i) = y(i - 1) + h * f(t(i - 1), y(i - 1));
    end
end
```

Here is the running result of Euler's method, the data has been rounded to 2 digital numbers in scientific notation

| $t$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| $\hat{y}(t)$ | 4.00E+00 | 1.20E+01 | 3.61E+01 | 1.10E+02 | 3.39E+02 | 1.07E+03 | 3.48E+03 | 1.17E+04 |
| $t$ | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| $\hat{y}(t)$ | 4.09E+04 | 1.49E+05 | 5.67E+05 | 2.27E+06 | 9.54E+06 | 4.24E+07 | 1.99E+08 | 9.86E+08 |

Table 1: Running result of Euler's method

### (b)

```
function [t, y] = Euler_b(Phi, t0, T, y0, n)
    f = inline(Phi, 't', 'y');
    h = (T - t0)/n;
    t(1) = t0;
    y(1) = y0;
    for i = 2 : n + 1
        t(i) = t(i - 1) + h;
        y(i) = y(i - 1)/(1 - f(t(i), h));
    end
end
```

Here is the running result of backward Euler's method, the data has been rounded to 2 digital numbers in scientific notation

| $t$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| $\hat{y}(t)$ | 4.00E+00 | -3.96E+00 | 3.81E+00 | -3.49E+00 | 3.01E+00 | -2.41E+00 | 1.77E+00 | -1.19E+00 |
| $t$ | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| $\hat{y}(t)$ | 7.25E-01 | -4.01E-01 | 2.00E-01 | -9.06E-02 | 3.71E-02 | -1.38E-02 | 4.66E-03 | -1.44E-03 |

Table 2: Running result of backward Euler's method

### (c)

```
function [t, y] = Taylor_sec(phi, phi_t, phi_y, t0, T, y0, n)
    Phi = inline(phi, 't', 'y');
    Phi_t = inline(phi_t, 't', 'y');
```

```
    Phi_y = inline(phi_y, 't', 'y');
    t(1) = t0;
    y(1) = y0;
    h = (T - t0)/n;
    for i = 2 : n + 1
        t(i) = t(i - 1) + h;
        p = Phi(t(i - 1), y(i - 1));
        pt = Phi_t(t(i - 1), y(i - 1));
        py = Phi_y(t(i - 1), y(i - 1));
        y(i) = y(i - 1) + h * p + 1/2 * h * h * (pt + py * p);
    end
end
```

Here is the running result of second-order Taylor's method, the data has been rounded to 2 digital numbers in scientific notation

| $t$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| $\hat{y}(t)$ | 4.00E+00 | 2.00E+01 | 1.01E+02 | 5.18E+02 | 2.75E+03 | 1.52E+04 | 8.87E+04 | 5.50E+05 |
| $t$ | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| $\hat{y}(t)$ | 3.66E+06 | 2.64E+07 | 2.07E+08 | 1.78E+09 | 1.69E+10 | 1.77E+11 | 2.06E+12 | 2.66E+13 |

Table 3: Running result of second-order Taylor's method

## (d)

```
function [t, y] = Henu(Phi, t0, T, y0, n)
    h = (T - t0)/n;
    t(1) = t0;
    y(1) = y0;
    f = inline(Phi, 't', 'y');
    for i = 2 : n + 1
        t(i) = t(i - 1) + h;
        k1 = f(t(i - 1), y(i - 1));
        ystar = y(i - 1) + h * k1;
        k2 = f(t(i), ystar);
        y(i) = y(i - 1) + 1/2 * h * (k1 + k2);
    end
end
```

Here is the running result of Henu's method, the data has been rounded to 2 digital numbers in scientific notation

| $t$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| $\hat{y}(t)$ | 4.00E+00 | 2.01E+01 | 1.02E+02 | 5.29E+02 | 2.85E+03 | 1.60E+04 | 9.56E+04 | 6.09E+05 |
| $t$ | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| $\hat{y}(t)$ | 4.17E+06 | 3.10E+07 | 2.52E+08 | 2.25E+09 | 2.21E+10 | 2.41E+11 | 2.93E+12 | 3.96E+13 |

Table 4: Running result of Henu's method

## (e)

```
function [t, y] = Adams(Phi, t0, T, y0, n)
    h = (T - t0)/n;
    t(1) = t0;
```

```
    y(1) = y0;
    f = inline(Phi, 't', 'y');
    t(2) = t0 + h;
    k1 = f(t(1), y(1));
    ystar = y(1) + h * k1;
    k2 = f(t(2), ystar);
    y(2) = y(1) + 1/2 * h * (k1 + k2);
    for i = 3 : n + 1
        t(i) = t(i - 1) + h;
        y(i) = y(i - 1) + 1/2 * h * (3 * f(t(i - 1), y(i - 1))-f(t(i - 2), y
    (i - 2)));
    end
end
```

Here is the running result of two-step Adams-Bashforth method, the data has been rounded to 2 digital numbers in scientific notation

| $t$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| $\hat{y}(t)$ | 4.00E+00 | 2.01E+01 | 7.65E+01 | 2.91E+02 | 1.12E+03 | 4.46E+03 | 1.83E+04 | 7.81E+04 |
| $t$ | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| $\hat{y}(t)$ | 3.48E+05 | 1.63E+06 | 8.04E+06 | 4.19E+07 | 2.32E+08 | 1.36E+09 | 8.49E+09 | 5.64E+10 |

Table 5: Running result of two-step Adams-Bashforth method

## (f)

**Algorithm 4:** Second-order Runge-Kutta method using the mid-point rule

**Require:** **A** has $n$ different eignvalues
**Input:** function $\Phi$, endpoints $t_0, T$, initial condition $y_0$, number of steps $n$
**Output:** t contains n + 1 equally spaced mesh points, starting from t0 to T y contains y0 and approximations to the solution at the corresponding t

```
Function Runge_Kutta_mid (Φ, t_0, T, y_0, n):
    h ← (T − t_0)/n;
    t[0] ← t_0;
    y[0] ← y_0;
    for i ← 1 to n do
        t[i] ← t[i − 1] + h;
        k_1 ← Φ(t[i − 1], y[i − 1]);
        ystar ← y[i − 1] + h · k_1;
        ymid ← 1/2 · (y[i − 1] + ystar);
        tmid ← t[i − 1] + h/2;
        y[i] ← y[i − 1] + h · Φ(tmid, ymid);
    end
    return (t, y);
end
```

```
function [t, y] = Runge_Kutta_mid(Phi, t0, T, y0, n)
    h = (T - t0)/n;
    t(1) = t0;
    y(1) = y0;
    f = inline(Phi, 't', 'y');
    for i = 2 : n + 1
```

```
        t(i) = t(i - 1) + h;
        k1 = f(t(i - 1), y(i - 1));
        ystar = y(i - 1) + h * k1;
        ymid = 1/2 * (y(i - 1) + ystar);
        tmid = t(i - 1) + h/2;
        y(i) = y(i - 1) + h * f(tmid, ymid);
    end
end
```

Here is the running result of Runge-Kutta method using the mid-point rule, the data has been rounded to 2 digital numbers in scientific notation

| $t$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| $\hat{y}$ | 4.00E+00 | 2.00E+01 | 1.01E+02 | 5.23E+02 | 2.79E+03 | 1.56E+04 | 9.18E+04 | 5.77E+05 |
| $t$ | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| $\hat{y}$ | 3.90E+06 | 2.85E+07 | 2.27E+08 | 1.99E+09 | 1.92E+10 | 2.06E+11 | 2.44E+12 | 3.23E+13 |

Table 6: Running result of Runge-Kutta method using the mid-point rule

## (g)

```
function [t, y] = Runge_Kutta_quad(Phi, t0, T, y0, n)
    h = (T - t0)/n;
    t(1) = t0;
    y(1) = y0;
    f = inline(Phi, 't', 'y');
    for i = 2 : n + 1
        t(i) = t(i - 1) + h;
        k1 = f(t(i - 1), y(i - 1));
        k2 = f(t(i - 1) + h/2, y(i - 1) + h/2 * k1);
        k3 = f(t(i - 1) + h/2, y(i - 1) + h/2 * k2);
        k4 = f(t(i - 1) + h, y(i - 1) + h * k3);
        y(i) = y(i - 1) + h/6 * (k1 + k2 + k3 + k4);
    end
end
```

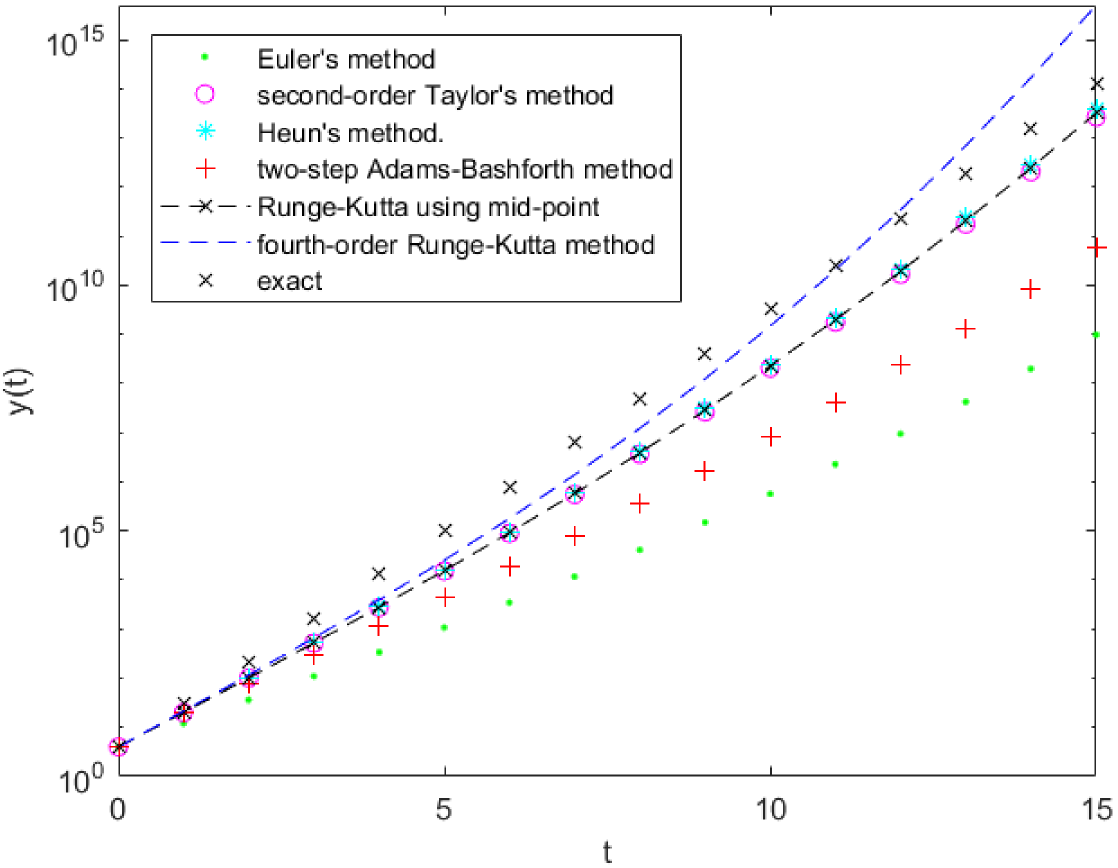
Here is the running result of fourth-order Runge-Kutta method, the data has been rounded to 2 digital numbers in scientific notation

| $t$ | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 |
|---|---|---|---|---|---|---|---|---|
| $\hat{y}$ | 4.00E+00 | 2.14E+01 | 1.17E+02 | 6.61E+02 | 3.95E+03 | 2.54E+04 | 1.78E+05 | 1.39E+06 |
| $t$ | 8 | 9 | 10 | 11 | 12 | 13 | 14 | 15 |
| $\hat{y}$ | 1.23E+07 | 1.25E+08 | 1.48E+09 | 2.06E+10 | 3.43E+11 | 6.89E+12 | 1.68E+14 | 5.04E+15 |

Table 7: Running result of fourth-order Runge-Kutta method

## (h)

All of the approximations and the exact solution is plotted on the figure below. The axis of $y$ is in logarithm to show all lines clearly. And the approximation done by backward Euler's method is not shown on this figure, since there is obvious error.




## (i)

```
function [P] = Newton_interpolation(t, y, n, x)
    P = y(1);
    for i = 2 : n
        prod = divided_difference(1, i, y, t);
        for k = 1 : i - 1
            prod = prod * (x - t(k));
        end
        P = P + prod;
    end
end

function [x] = divided_difference(i, k, y, t)
    if k - i == 1
        x = (y(k) - y(i))/(t(k) - t(i));
    else
        x = (divided_difference(i + 1, k, y, t) - divided_difference(i, k -
    1, y, t))/(t(k) - t(i));
    end
end
```

```
>> [t,y1]=Euler_f('(2+0.01*t^2)*y',0,15,4,15);
>> Newton_interpolation(t, y1, 16, 9.625)

ans =

   3.4085e+05
```

So the value of $y$ at $t = 9.625$ by using the approximation from Euler's method and interpolation in Newton's form is $\hat{y}(9.625) = 3.4085 \times 10^5$.

# 6.

Set $y_1 = y, y_2 = \dot{y}, y_3 = \ddot{y}$, then

$$\begin{pmatrix} \dot{y_1} \\ \dot{y_2} \\ \dot{y_3} \end{pmatrix} = \begin{pmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -\frac{2}{t^3} & \frac{2}{t^2} & -\frac{1}{t} \end{pmatrix} \begin{pmatrix} y_1 \\ y_2 \\ y_3 \end{pmatrix} + \begin{pmatrix} 0 \\ 0 \\ 8 - \frac{2}{t^3} \end{pmatrix}$$

with the initial condition

$$\begin{pmatrix} y_1(1) \\ y_2(1) \\ y_3(1) \end{pmatrix} = \begin{pmatrix} 2 \\ 8 \\ 6 \end{pmatrix}$$

Then we use Matlab to apply fourth-order Runge-Kutta method

```
clear all clc;
h = 0.1;
t(1) = 1;
n = (2 - 1)/h;
y1(1) = 2;
y2(1) = 8;
y3(1) = 6;
f = @(t,y1,y2,y3)[[0, 1, 0];[0, 0, 1];[-2/t^3, 2/t^2, -1/t]] * [y1; y2; y3]
    + [0; 0; 8 - 2/t^3];
for i = 2 : n + 1
    t(i) = t(i - 1) + h;
    k1 = f(t(i - 1), y1(i - 1), y2(i - 1), y3(i - 1));
    k2 = f(t(i - 1) + h/2, y1(i - 1) + h/2 * k1(1), y2(i - 1) + h/2 * k1(2),
      y3(i - 1) + h/2 * k1(3));
    k3 = f(t(i - 1) + h/2, y1(i - 1) + h/2 * k2(1), y2(i - 1) + h/2 * k2(2),
      y3(i - 1) + h/2 * k2(3));
    k4 = f(t(i - 1) + h, y1(i - 1) + h * k3(1), y2(i - 1) + h * k3(2), y3(i
    - 1) + h * k3(3));
    y1(i) = y1(i - 1) + h/6 * (k1(1) + k2(1) + k3(1) + k4(1));
    y2(i) = y2(i - 1) + h/6 * (k1(2) + k2(2) + k3(2) + k4(2));
    y3(i) = y3(i - 1) + h/6 * (k1(3) + k2(3) + k3(3) + k4(3));
end
```

and we find the approximation to solution is (results have been rounded to 2 digital numbers)

| $t$ | 1 | 1.1 | 1.2 | 1.3 | 1.4 | 1.5 | 1.6 | 1.7 | 1.8 | 1.9 | 2 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $\hat{y}(t)$ | 2.00 | 2.55 | 3.14 | 3.76 | 4.42 | 5.11 | 5.85 | 6.63 | 7.46 | 8.34 | 9.26 |

Table 8: Approximation to solution by using fourth-order Runge-Kutta method

compared with the exact solution, we can see the error is listed in below

| $t$ | 1 | 1.1 | 1.2 | 1.3 | 1.4 | 1.5 | 1.6 | 1.7 | 1.8 | 1.9 | 2 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| $\hat{y}(t)$ | 2.00 | 2.55 | 3.14 | 3.76 | 4.42 | 5.11 | 5.85 | 6.63 | 7.46 | 8.34 | 9.26 |
| $y(t)$ | 2.00 | 2.83 | 3.73 | 4.72 | 5.79 | 6.96 | 8.23 | 9.61 | 11.12 | 12.74 | 14.50 |
| relative error(%) | 0.00 | -9.79 | -15.90 | -20.27 | -23.68 | -26.50 | -28.90 | -31.01 | -32.89 | -34.59 | -36.14 |

# 7.

## (a)

Use test function $u(x)$ which satisfies that $u(1) = u(3) = u'(1) = u'(3) = 0$,

$$\int_1^3 u(x) \cdot x^3 y^{(4)} dx + \int_1^3 u(x) \cdot 6x^2 y^{(3)} dx + \int_1^3 u(x) \cdot 6xy'' dx - \int_1^3 u(x) \cdot 10x dx = 0$$

$$\Rightarrow u(x) \cdot (x^3 y'')'|_1^3 - \int_1^3 u'(x) \cdot (x^3 y'')' dx = \int_1^3 u(x) \cdot 10x dx$$

$$\Rightarrow - u'(x) \cdot (x^3 y'')|_1^3 + \int_1^3 u''(x) \cdot (x^3 y'') dx = \int_1^3 u(x) \cdot 10x dx$$

$$\Rightarrow \int_1^3 x^3 y''(x) u''(x) dx = \int_1^3 u(x) \cdot 10x dx$$

## (b)

Assume $u(x) = y(x)$, and set the energy function

$$I[u] = \int_1^3 x^3 (u''(x))^2 dx - 2\int_1^3 u(x) \cdot 10x dx$$

Let $\hat{y}(x) = \sum_{j=1}^{n} c_j \phi_j(x)$, substituting $\hat{y}(x)$ in to $I[u]$, we have

$$I = \int_1^3 x^3 (\sum_{j=1}^{n} c_j \phi_j''(x))^2 dx - 2\int_1^3 \sum_{j=1}^{n} c_j \phi_j(x) \cdot 10x dx$$

then

$$\frac{\partial I}{\partial c_i} = \int_1^3 x^3 (2\phi_i''(x) \sum_{j=1}^{n} c_j \phi_j''(x)) dx - 2\int_1^3 \phi_i(x) \cdot 10x dx$$

set it to zero and we obtain a set of equations $\mathbf{Ax} = \mathbf{b}$, where

$$a_{ij} = \int_1^3 x^3 \phi_i''(x) \phi_j''(x) dx, \quad b_i = \int_1^3 10x \phi_i(x) dx$$

Choose $\phi(x)$ as

$$\phi_i(x) = \begin{cases} 0 & ,x \in [1, x_{i-1}] \\ \dfrac{(x - x_{i-1})(x_{i+1} - x)}{(x_i - x_{i-1})(x_{i+1} - x_i)} & ,x \in (x_{i-1}, x_{i+1}] \\ 0 & ,x \in (x_{i+1}, 3] \end{cases} \Rightarrow \phi_i''(x) = \begin{cases} 0 & ,x \in [1, x_{i-1}] \\ -\dfrac{2}{(x_i - x_{i-1})(x_{i+1} - x_i)} & ,x \in (x_{i-1}, x_{i+1}] \\ 0 & ,x \in (x_{i+1}, 3] \end{cases}$$

then

$$\begin{aligned} a_{ii} &= \int_1^3 x^3 \Big(\frac{2}{(x_i - x_{i-1})(x_{i+1} - x_i)}\Big)^2 dx \\ &= \int_{x_{i-1}}^{x_{i+1}} x^3 \Big(\frac{2}{(x_i - x_{i-1})(x_{i+1} - x_i)}\Big)^2 dx \\ &= \frac{x_{i+1}^4 - x_{i-1}^4}{(x_i - x_{i-1})^2 (x_{i+1} - x_i)^2} \end{aligned}$$

$$\begin{aligned} a_{i,i+1} &= \int_1^3 x^3 \frac{2}{(x_i - x_{i-1})(x_{i+1} - x_i)} \frac{2}{(x_{i+1} - x_i)(x_{i+2} - x_{i+1})} dx \\ &= \int_{x_i}^{x_{i+1}} x^3 \frac{4}{(x_i - x_{i-1})(x_{i+1} - x_i)(x_{i+1} - x_i)(x_{i+2} - x_{i+1})} dx \\ &= \frac{x_{i+1}^4 - x_i^4}{(x_i - x_{i-1})(x_{i+1} - x_i)(x_{i+1} - x_i)(x_{i+2} - x_{i+1})} \end{aligned}$$

else $a_{ij} = 0$. And

$$\begin{aligned} b_i &= \int_1^3 10x\phi_i(x) dx = \int_{x_{i-1}}^{x_{i+1}} 10x \frac{(x - x_{i-1})(x_{i+1} - x)}{(x_i - x_{i-1})(x_{i+1} - x_i)} dx \\ &= \frac{\frac{5}{6}(x_{i+1}^4 - x_{i-1}^4) - \frac{5}{3} x_{i-1} x_{i+1} (x_{i+1}^2 - x_{i-1}^2)}{(x_i - x_{i-1})(x_{i+1} - x_i)} \end{aligned}$$

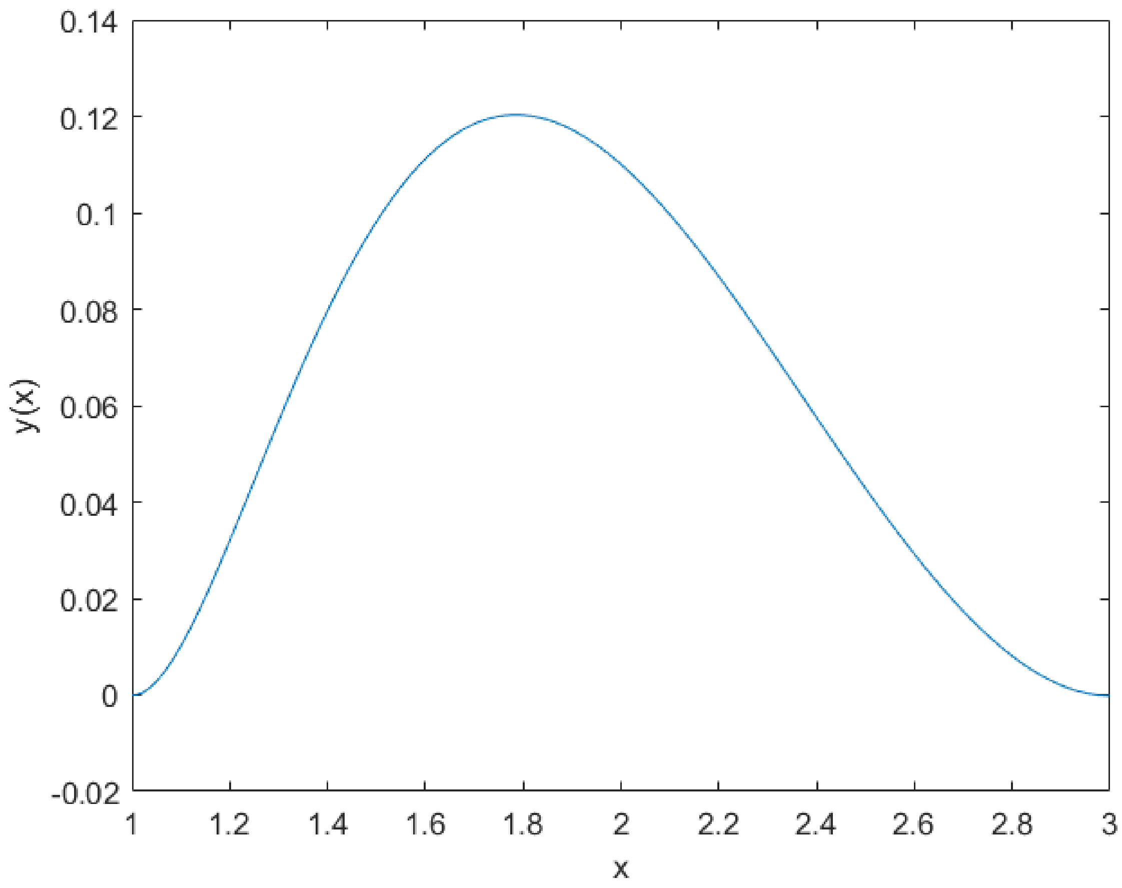
## (c)

To find the exact solution,

$$\begin{aligned} & x^3 y^{(4)} + 6x^2 y^{(3)} + 6xy'' - 10x = 0 \\ \Leftrightarrow & (x^3 y'')'' = 10x \\ \Rightarrow & x^3 y'' = \frac{5}{3} x^3 + c_1 x + c_2 \\ \Rightarrow & y'' = \frac{5}{3} + \frac{c_1}{x^2} + \frac{c_2}{x^3} \\ \Rightarrow & y' = \frac{5}{3} x - \frac{c_1}{x} - \frac{c_2}{2x^2} + c_3 \\ \Rightarrow & y = \frac{5}{6} x^2 - c_1 \ln x + \frac{c_2}{2x} + c_3 x + c_4 \end{aligned}$$

Since $y(1) = y(3) = y'(1) = y'(3) = 0$,

$$c_1 = -16.9012, \quad c_2 = 17.8518, \quad c_3 = -9.6420, \quad c_4 = -0.1173$$


0.14
0.12
0.1
0.08
0.06
0.04
0.02
0
-0.02
y(x)
1
1.2
1.4
1.6
1.8
2
2.2
2.4
2.6
2.8
3
x