P1

a) Runge phenomenon is to describe extreme "polynomial wiggle" associated with high degree polynomial interpolation at evenly spaced points.

~~How to reduce~~ How to reduce: move some of the interpolation points towards the outside of the interval, where the function produce data can be better fit

Interpolation error: $\frac{\prod_{i=1}^{n}(x-x_i)}{n!} f^{n}(c) \quad x \in [-1,1]$

So we choose x_i $i \in \{1, \dots n\}$ to make ~~min of~~ $\max_{-1 \le x \le 1} |\prod (x - x_i)|$ as small as possible

$\Rightarrow$ choose $x_i = \cos \frac{(2i-1)\pi}{2n} \Rightarrow$ min value is $\frac{1}{2^{n-1}}$

Generalize to any $[a, b]$ interval

$$\Rightarrow x_i = \frac{a+b}{2} + \frac{b-a}{2} \cos\left(\frac{(2i-1)\pi}{2n}\right)$$

To prove this, we only need to prove $[-1, 1]$ interval

Let P_n monic polynomial.

T_n is chebyshev polynomial $(T_n(x) = \cos(n \arccos(x)))$

Assume $|P_n(x)| \le 1/2^{n-1}$

Since T_n alternate between -1 and 1 $n+1$ times

(It has ~~proven~~ roots at $\cos \frac{i\pi}{n}$ $i = 0, \dots n$ ~~1~~)

$\Rightarrow P_n - T_n/2^{n-1}$ alternate positive and negative $n+1$ times at these points

$\Rightarrow P_n - T_n/2^{n-1}$ crosses zeros at least n times

$\Rightarrow P_n - T_n/2^{n-1}$ has at least n ~~proven~~ roots

$\Rightarrow$ contradict ~~the~~ the degree difference is $\le n-1$

$\Rightarrow$ ~~8~~ $\max_{-1 \le x \le 1} |P_n| \ge 1/2^{n-1}$

$\Rightarrow$ for $(a, b]$ interval $x_i = \frac{a+b}{2} + \frac{b-a}{2} \cos\left(\frac{(2i-1)\pi}{2n}\right)$

$\Rightarrow$ ~~min of max~~ $\left|\prod_{i=1}^{n}(x - x_i)\right| \le \left(\frac{b-a}{2}\right)^n / 2^{n-1}$

(2)

b) Runge - Kutta pairs is pairs of Runge - Kutta methods one of order p and another of order p+1.

Adapt ~~time~~ step's size

Define $e_{i+1} \approx |z_{i+1} - w_{i+1}|$ (z_{i+1} and w_{i+1} is 2 ~~new~~ estimation of Runge - Kutta methods)

estimated error

Let ε = error tolerance

If $|e_{i+1}| > |\varepsilon|$

$\Rightarrow h_{i+1} = h_i/2$ (We cut the step size in ~~two~~ half)

~~If~~ else if $|e_{i+1}| \leq |\varepsilon|$

$h_{i+1} = h^*$ (Choose ~~on~~ a new appropriate step size base on h_i: $h^* = g(h_i)$).

(3)

c) Stability of a finite difference method solution for the heat equation is when the error is amplified when the time step k is too large relative to spatial ~~and~~ step h

The necessary condition: CFL condition

for heat equation, we need

$$\sigma = \frac{Dk}{h^2} \le \cancel{\tfrac{1}{2}}1$$

With
D: diffusion coefficient
k: time step (temporal step size)
h: spatial step

$\boxed{P2}$

a) Matlab code below.

b). $\begin{bmatrix} 3 & 7 \\ 6 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ -11 \end{bmatrix}$

$A = A_1 = \begin{bmatrix} 3 & 7 \\ 6 & 1 \end{bmatrix}$ $\Rightarrow M_1 = \begin{bmatrix} 1 & 0 \\ -2 & 1 \end{bmatrix}$

$\Rightarrow M_1^{-1} = \begin{bmatrix} 1 & 0 \\ 2 & 1 \end{bmatrix}$

$\Big\downarrow$ ~~2xrow 1 - row 2~~ row 2 − 2 row 1

$A_2 = \begin{bmatrix} 3 & 7 \\ 0 & -13 \end{bmatrix}$ $b_2 = \begin{bmatrix} 1 \\ -13 \end{bmatrix}$

$\Rightarrow U = A_2 = \begin{bmatrix} 3 & 7 \\ 0 & -13 \end{bmatrix}$; $L = \begin{bmatrix} 1 & 0 \\ 2 & 1 \end{bmatrix}$; $b_2 = \begin{bmatrix} 1 \\ -13 \end{bmatrix}$

④

Back substitution

- $Lc = b$

$$\Rightarrow \begin{bmatrix} 1 & 0 \\ 2 & 1 \end{bmatrix} \begin{bmatrix} c_1 \\ c_2 \end{bmatrix} = \begin{bmatrix} 1 \\ -11 \end{bmatrix}$$

$$\Rightarrow \begin{cases} c_1 = 1 \\ 2c_1 + c_2 = -11 \end{cases} \Rightarrow \begin{cases} c_1 = 1 \\ c_2 = -13 \end{cases}$$

- $Ux = c$ for x

$$\Rightarrow \begin{bmatrix} 3 & 7 \\ 0 & -13 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ -13 \end{bmatrix}$$

$$\Rightarrow \begin{cases} 3x_1 + 7x_2 = 1 \\ -13x_2 = -13 \end{cases}$$

$$\Rightarrow \begin{cases} x_1 = \cancel{1} -2 \\ x_2 = 1 \end{cases} \Rightarrow x = \begin{bmatrix} \cancel{x} -2 \\ 1 \end{bmatrix}$$

$\left[v_i \right] \quad \left[w_{i+1} = f(u_{i+1}) \right.$

8

P3

a) The Jacobi methods apply ~~both~~ value calculated by the previous ~~sthep~~ step onto both L and U

Jacobi

x_0 = initial vector

$x_{k+1} = D^{-1}\{b - (L+U)x_k\}$

for $k = 0, 1, 2 \ldots$

The Gaussian-Seidel use the latest updated value ~~of~~ onto L and the value of previous ~~step~~ step

Gaussian - Seidel

latest updated value of $x_{i\,k+1}$

x_0 = initial vector

$x_{k+1} = D^{-1}(b - L x_{k+1} - U x_k)$ $\quad i = 1, 2 \ldots n$

for $k = 0, 1, 2 \ldots$

b) I run the ~~answer~~ matrices A, b with matlab and ~~they~~ it converge for both Jacobi and Gaussian Seidel.

7

c) Check Matlab code

P4 ⑥

$$f(x) = f(w) + f'(w)(x-w) + \frac{f''(c_x)}{2}(x-w)^2$$

$$\int_a^b f(x)\,dx$$

Prove composite Midpoint Rule ~~apple~~:

$$\int_a^b f(x)\,dx = h\sum_{i=1}^{m} f(w_i) + \frac{(b-a)h^2}{24} f''(c)$$

$\forall [a,b]$ ~~f~~ for given h

~~Let m~~ $x_0 = a$, $x_m = b$, $h = x_m - x_{m-1} = \dots = x_1 - x_0$

~~On $[x_{i-1}, x_{i+1}]$ choose $w_i =$~~ $\quad h = \frac{b-a}{m}$

On $[x_{i-1}, x_i]$ choose $w_i = \frac{x_i - x_{i-1}}{2}$ mid point

$$\int_{x_{i-1}}^{x_i} f(x)\,dx = \int_{x_{i-1}}^{x_i} f(w_i) + f'(w_i)(x-w_i) + f''(c_{x_i})(x-w_i)^2$$

$$= \Big[x f(w_i) + f'(w_i)\frac{(x-w_i)^2}{2} + f''(c_{x_i})\frac{(x-w_i)^3}{2\times 3} \Big]_{x_{i-1}}^{x_i}$$

$$= \underbrace{(x_i - x_{i-1})}_{h} f(w_i) + f'(w_i)\frac{\overbrace{(x_i-w_i)^2}^{h^2/2^2} - \overbrace{(x_{i-1}-w_i)^2}^{+h^2/2^2}}{2}$$

$$+ f''(c_{x_i})\frac{\overbrace{(x_i-w_i)^3}^{+h^3/8} - \overbrace{(x_{i-1}-w_i)^3}^{-h^3/8}}{6}$$

$$= h f(W_i) + 0 + f''(c_{x_i})\frac{h^3}{24} \quad (1)$$

$$\int_a^b f(x)\,dx = \sum_{i=1}^{m} \int_{x_{i-1}}^{x_i} f(x)\,dx$$

$$\overset{(1)}{=} \sum_{i=1}^{m} h f(W_i) + f''(c_{x_i})\frac{h^3}{24}$$

$$= h\sum_{i=1}^{m} f(w_i) + \frac{h^3}{24}\sum_{i=1}^{m} f''(c_{x_i})$$

7

$$\int_a^b f(x)\,dx = h\sum_{i=1}^{m} f(w_i) + \frac{h^3}{24}\, m\, f''(c)$$

$$= h\sum_{i=1}^{m} f(w_i) + \frac{(b-a)}{24} h^2 f(c)$$

$$\Rightarrow \int_a^b f(x)\,dx = h\sum_{i=1}^{m} f(w_i) + \frac{b-a}{24} h^2 f(c)$$

$(a < c < b)$

$\boxed{P5}$ found in Matlab code below.

$\boxed{P6}$ $f(t,y)$ Lipschitz constant $L = 1$ for y

$$\begin{cases} y' = f(t,y) \\ y(0) = y_0 \\ t \text{ in } [0,T] \end{cases}$$

y_i at t_i ~~of~~ approximated by one-step ODE.

local truncation error $e_i \leq Ch^{k+1}$

$C = 1,\ k \geq 0$

$\forall\, 0 \leq t_i \leq T$ global truncation error

$$g_i = |w_i - y_i| \leq \frac{Ch^k}{L}\left(e^{Lt_i} - 1\right)$$

Estimate global truncation errors at $T = 10$ for

a) Euler's method with $e_i \leq Ch^2$

b) midpoint method with $e_i \leq Ch^3$

if at $T = 1$, 2 method have same local truncation error $g_1 = 0.1$, $h = 0.1$

8

a) We choose $h = 0.1$

~~$e_i \leq Ch^2 = 1 \times 1^2 = 1$~~

~~$\Rightarrow e_i \leq 1$~~

for Euler's method

$$e_i \leq c h^2 \Rightarrow k = 1.$$

$$\Rightarrow g_i = |w_i - y_i| \leq \frac{c h^1}{L} (e^{L t_i} - 1).$$

with $c = 1, h = 0.1, L = 1, t_i = T$

$$g_i \leq \frac{1 \times 0.1^1}{1} (e^{1 \times 10} - 1)$$

$$\Rightarrow g_i \leq 0.1 (e^{10} - 1)$$

b) We choose $h = 0.1$

for Mid-point method

$$e_i \leq c h^3 \Rightarrow k = 2$$

$$g_i = |w_i - y_i| \leq c \frac{h^2}{L} (e^{L t_i} - 1)$$

with $c = 1, h = 0.1, L = 1, t_i = T$

$$\Rightarrow g_i \leq \frac{1 \times 0.1^2}{1} (e^{1 \times 10} - 1)$$

$$\Rightarrow g_i \leq 0.01 (e^{10} - 1)$$

Problem 2

```
% a
% Test my function
% You can find the function at the end
A = [1,2,3;40,35,6;7,8,9];
b = [1;2;2];
x = gaussian_partial_pivoting(A,b)
```

```
x = 3×1
   -0.6458
    0.7917
    0.0208
```

```
linsolve(A,b)
```

```
ans = 3×1
   -0.6458
    0.7917
    0.0208
```

```
% b
% Check my answer in the paper
A = [3,7;6,1]
```

```
A = 2×2
     3     7
     6     1
```

```
b = [1;-11]
```

```
b = 2×1
     1
   -11
```

```
linsolve(A,b)
```

```
ans = 2×1
    -2
     1
```

```
gaussian_partial_pivoting(A,b)
```

```
ans = 2×1
    -2
     1
```

Problem 3

```
% b
A = [5,1,-1;
    3,6,2;
    2,-6,9]
```

```
A = 3×3
     5     1    -1
     3     6     2
     2    -6     9
```

```
b = [1;1;1]
```

```
b = 3×1
     1
     1
     1
```

```
n = length(b);
x0 = zeros(n,1);

% Check the convergence of jacobi and gaussian seidel
% we see that both method converge to the solution solved by matlab
k =1000000; % number of iteration

%a
jacobi(A,b, k, x0) % jacobi method
```

```
ans = 3×1
    0.2107
    0.0326
    0.0861
```

```
%b
gauss_seidel(A,b,k,x0) % gaussian seidel method
```

```
ans = 3×1
    0.2107
    0.0326
    0.0861
```

```
linsolve(A,b) % using matlab
```

```
ans = 3×1
    0.2107
    0.0326
    0.0861
```

Problem 5

```
% a two-point forward-difference
% f'(x) = 1/x
x = 1;
f = @(x) log(x);
```

```
h = 0.1;
diff_f_1 = (f(x+h) - f(x))/h % 2-point forward-difference
```

```
diff_f_1 = 0.9531
```

```
% Approximation error
error = abs(diff_f_1 - 1/x)
```

```
error = 0.0469
```

```
% three-point centered-difference formula
% f' = exp(x)
x = 0;
f = @(x) exp(x);
h = 0.1;
diff_f_1 = (f(x+h) - f(x-h))/(2*h) % 3-point centered-difference
```

```
diff_f_1 = 1.0017
```

```


% Aprroximation error
error = abs(diff_f_1 - exp(x))
```

```
error = 0.0017
```

```
function x=gaussian_partial_pivoting(A, b)
% This function apply gaussian elimination with partial pivoting
a = A;
n = length(b);
for j = 1 : n -1
    if abs ( a (j ,j )) < eps
        error ( "zero pivot encountered");
    end

    % Find the largest pivot
    r = j;
    for i = j+1:n
        if abs(a(i,j))>abs(a(r,j))
            r = i;
        end
    end

    % Exchange row in A
    temp_a = a(j, :);
    a(j,:) = a(r,:);
    a(r,:) = temp_a;

    % Exchange row in b
    temp_b = b(j);
    b(j) = b(r);
```

```
    b(r) = temp_b;

    % For ward elimination
    for i = j + 1 : n
        mult = a (i , j ) / a (j , j );
        a(i,:) = a(i,:) - mult * a (j,:);
        b ( i ) = b( i ) - mult * b ( j );

    end
end

% Backward substitution
x = zeros(n,1);
for i = n : -1 : 1
    for j = i +1 : n
        b ( i ) = b( i ) - a (i , j ) * x ( j );
    end
        x ( i ) = b( i ) / a (i , i );
end

end

% Gauss-Seidel method
% Estimates solution for equation Ax=b
% Input: coefficient matrix a,
%        right-hand-side vector b,
%        number of interations k,
%        initial guess x0
% Output: solution x.
function x = gauss_seidel(a,b,k,x0)
n=length(b);
d=diag(a);
l = tril(a, -1);
u = triu(a, 1);
x = x0.*ones(n,1);
xi = x;
for j=1:k
    for i=1:n
        xi(i) = (b(i) - u(i,:)*x - l(i,:)*xi) /d(i);
    end
    x = xi;
end
end

% Jacobi Method
% Estimates solution for equation Ax=b, uses zero vector as an initial
% guess
% Inputs: full or sparse matrix a,
%         right-hand-side vector b,
%         number of Jacobi iterations k,
```

```
%         initial guess x0
% Output: solution x
function x = jacobi(a,b,k,x0)
n=length(b);      % find n
d=diag(diag(a));        % extract diagonal of a
r=a-d;      % r is the remainder
x=x0.*ones(n,1);     % initialize vector x
for j=1:k         % loop for Jacobi iteration
  x=d\(b-r*x);
end               % End of Jacobi iteration loop
end
```