Homework 4 - MSSC 6030: Spring 2020

Directions. All work is to be done in *complete sentences.* Assignments must be stapled with a printout of the assignment serving as the first page. Each problem must be on a *separate* sheet of paper. You are welcome to recycle paper, where one side is crossed out to avoid wasting paper, but your work MUST have **no more than one problem per page**. Each problem write-up must begin with the **full statement of the problem**. While you are encouraged to work through confusion with your classmates, your work must be written in your own words. **The assignment is due in dropbox on Wednesday, April 15, 2020 by 3:15pm.**

---

1. Use MC methods to compute the following

   a) the AREA trapped by $\sin(x)$ for x in $[0, 2\pi)$

   b) the integral of sin(x) from $x = 0$ to 2π

   c) (not using MC methods) Compare your results from a) and b) to the results you get with Calculus.

---

2. Use MC methods to compute the following

   a) the AREA trapped by $f(x) = x^2 + 2$ using the rectangular dartboard described in class.

   b) the AREA trapped by $f(x) = x^2 + 2$ using a trapezoid instead of a rectangle for your dartboard

   c) Compare your results from a) and b) to the results you get with Calculus.

---

3. Write a computer program to carry out the Monte Carlo Random Walk solution method for the following BVP

$$\begin{array}{lrcll} \text{PDE} & u_{xx} + u_{yy} & = & 0 & 0 < x < 1, \;\; 0 < y < 1 \\ \\ \text{BCs} & u(0, y) & = & 0 & 0 < y < 1 \\ & u(1, y) & = & 2y & 0 < y < 1 \\ & u(x, 0) & = & 0 & 0 < x < 1 \\ & u(x, 1) & = & 1 & 0 < x < 1 \end{array}$$

   You may use a uniform grid size $h = k$ or let the x grid size (h) and y grid size (k) be different. How accurate is your solution? How do you know? Also, code the problem using the *Finite Difference* method. Compare your solution to that from MC. Discuss.

---

4. Write a computer program to carry out the Monte Carlo Random Walk solution method for the following BVP

$$\begin{array}{lrcll} \text{PDE} & u_{xx} + x^2 u_{yy} & = & 0 & 0 < x < 1, \;\; 0 < y < 1 \\ \\ \text{BCs} & u(0, y) & = & 0 & 0 < y < 1 \\ & u(1, y) & = & 0 & 0 < y < 1 \\ & u(x, 0) & = & 5 & 0 < x < 1 \\ & u(x, 1) & = & 1 & 0 < x < 1 \end{array}$$

   You may use a uniform grid size $h = k$ or let the x grid size (h) and y grid size (k) be different. How accurate is your solution? How do you know? Also, code the problem using the *Finite Difference* method. Compare your solution to that from MC. Discuss.

5. Let $u(x) = \frac{1}{x+1}$ on [0,2]. Approximate u on $[0, 2]$ using n equally spaced subintervals for $n = 2, 5, 10$ using

   a) polynomial interpolation with the Vandermonde matrix and a polynomial of degree n

   b) polynomial interpolation with the Lagrange formula and a polynomial of degree n

   Plot your results and discuss.

---

6. For the fundamental B-spline discussed in the lecture, check that $B(t)$, $B'(t)$, and $B''(t)$ are continuous across the nodes (knots).

$$B(t) = \begin{cases} \frac{1}{6}t^3 & 0 \le t \le 1 \\ \frac{1}{6}\left[-3(t-1)^3 + 3(t-1)^2 + 3(t-1) + 1\right] & 1 \le t \le 2 \\ \frac{1}{6}\left[3(t-2)^3 - 6(t-2)^2 + 4\right] & 2 \le t \le 3 \\ \frac{1}{6}\left[-(t-3)^3 + 3(t-3)^2 - 3(t-3) + 1\right] & 3 \le t \le 4 \\ 0, & t \le 0 \quad \text{or} \quad t \ge 4. \end{cases}$$

   *Plot $B(t)$.

---

7. Let $u(x) = \frac{1}{x+1}$ on [0,2]. Approximate u on $[0, 2]$ using n equally spaced subintervals for $n = 2, 5, 10$ using

   a) n cubic splines with the polynomial approach

   b) n cubic splines with B-spline approach.

   c) n cubic splines using the spline command in Matlab.

   Plot your results and compare to those from H1. How do your results change if you use a 'natural' spline or a 'clamped' spline?

---

1. Use MC methods to compute the following

a) the AREA trapped by $\sin(x)$ for in $[0, 2\pi)$

b) the integral of $\sin(x)$ from $x = 0$ to 2π

c) (not using MC methods) Compare your results from a) and b) to the results you get with Calculus

a)

Using MC method calculus the area.

Since the area cannot be negative, we need to use the MC on the whole range $[0, 2\pi)$

We will drop darts on the interval of 0 to 2π and maximum, minimum of $\sin(x)$ which is 1 and -1.

Then test if the dart we drop is bounded by the sin(x) and x-axis.

Then we could count the number of darts that in the area and total darts we dropped.

Therefore we could calculate the area trapped by the line.

The specific step is following

step 1: intitialize count=0

step 2: loop from 1 to n

step 3: generate random num x: $0 \le x \le 2\pi$,

y: $-1 \le y \le 1$.

step 4: calculate $\mathrm{fx} = \sin(x)$

step 5: if $0 \le y \le \mathrm{fx}$ or $\mathrm{fx} \le y \le 0$ increase count by 1. Otherwise goes to step 3

step 6: area= $2 \times 2\pi \times$ count/n

```
clear
%step 1 initial count=0
count = 0;
num = 10000;%set n=10000

for j = 1:num
    %step 2 loop from 1 to n
    x = 2*pi*rand;%step 3 generate random number of x from 0 to 2*pi
    y = -1+(1-(-1))*rand;%step 3 generate random number of y from -1 to 1
    fx = sin(x);%step 4 calculate fx
    if (0<=y) && (y<=fx) || (fx<=y) && (y<=0)
        %step 5 test if y is bounded by fx and x-axis
        count=count+1;%count increase by 1
    end
end
%step 6 calculate the total area.
```

```
Areafx=2*pi*2*count/num
```

```
Areafx = 4.0677
```

So the area trapped by the line is about 4

b)

$$\int_0^{2\pi} \sin(x)\mathrm{dx} = 0$$

c)

From the integral we coud see that, the integral of sin(x) from 0 to 2π is 0. But if we calculate

$$\left|\int_0^{\pi} \sin(x)\mathrm{dx}\right| + \left|\int_{\pi}^{2\pi} \sin(x)\mathrm{dx}\right|$$

$$\left|\int_0^{\pi} \sin(x)\mathrm{dx}\right| + \left|\int_{\pi}^{2\pi} \sin(x)\mathrm{dx}\right| = 2 + 2 = 4$$

Which the result from MC method is similar with the true value.

2. Use MC methods to compute the following

a) the AREA trapped by $f(x) = x^2 + 2$ using the rectangular dartboard described in class

b) the AREA trapped by $f(x) = x^2 + 2$ using a trapezoid instead of a rectangle for your dartboard

c) Compare your results from a) and b) to the results you get with Calculus

a)

Let the range of the area trapped from x=0 to x=4

So the maximum of f(x) is $4^2 + 18$, and minimum of f(x) is 0.

step 1: intitialize count=0

step 2: loop from 1 to n

step 3: generate random num x: $0 \le x \le 4$,

y: $0 \le y \le 18$

step 4: calculate $\mathrm{fx} = x^2 + 2$

step 5: if $y \le \mathrm{fx}$ increase count by 1. Otherwise goes to step 3

step 6: area=4*18*count/n

```
clear
%step 1 initial count=0
count = 0;
num = 10000;%set n=10000

for j = 1:num
    %step 2 loop from 1 to n
    x = 4*rand;%step 3 generate random number of x from 0 to 4
    y = 18*rand;%generate random number of y from 0 to 18
    fx = x^2+2;%step 4: calculate fx
    if y <= fx
        %test if y is less than fx
        count = count + 1;%count increase by 1
    end
end
%step 6
AreaFx=18*4*(count/num)
```

```
AreaFx = 29.4552
```

So as result the area under $f(x) = x^2 + 2$ by using MC method is around 29

b)

Since $\mathrm{fx} = x^2 + 2$ is curve up so the trapezoid created by (0,0),(4,0),(0,2),(4,18)

The line of (0,2),(4,18) is

$y = 4x + 2$

So in step 3 generate random number of y $0 \le y \le 4x + 2$

The area of trapezoid is (2+18)*4/2=40

And in step 6: area=40*count/n

```
numt = 10000;
countt = 0;
for jt = 1:num
    xt = 4*rand;
    yt = (4*xt + 2)*rand;%modified step 3
    fxt = xt^2+2;
    if yt <= fxt
        countt = countt+1;
    end
end

AreaFxt=40*(countt/numt)
```

```
AreaFxt = 27.4240
```

So as result the area under $f(x) = x^2 + 2$ by using MC method with trapezoid is around 27

c)

$$\int_0^4 x^2 + 2 = \left(\frac{1}{3}x^3 + 2x\right)_0^4 = \frac{1}{3}4^3 + 8 = 29.33$$

So the true value of the area is 29.33, which means that the normal MC has more accuraccy.

3. Write a computer program to carry out the Monte Carlo Random Walk solution method for following BVP

PDE $u_{xx} + u_{yy} = 0 \qquad 0 < x < 1, 0 < y < 1$

BCs $u(0, y) = 0 \qquad 0 < y < 1$

$u(1, y) = 2y \qquad 0 < y < 1$

$u(x, 0) = 0 \qquad 0 < x < 1$

$u(x, 1) = 1 \qquad 0 < x < 1$

You may use a uniform grid size h=k or let the x grid size (h) and y grid size (k) be diffierent. How accurate is your solution? How do you know? Also, code the problem using the Finite Difference method. Compare your solution to that from MC. Discuss.

By using Monte Carlo Random Walk method we need to calculate the probability that random walk attach to the boundary which starts from each point inside the boundary.

So for each point A and each boundary point p_k $P_A(p_k) = \frac{\text{Number of times reached } p_k \text{ from } A}{\text{Total number of random walk}}$ (1)

And compute the each interior point by adding the 'expectation value' together:

$u(A) = \sum_k g_k P_A(p_k)$ (2)

Where g_k is the boundary condition.

Then in order to do the random walk since x and y has same grid, the probability that the walks goes to each way are the same. So we will generate a random number with $w \sim U(0, 1)$ and

If $0 \le w < 0.25$ move from (i,j) to (i+1,j), which is North direction

If $0.25 \le w < 0.5$ move from (i,j) to (i-1,j), which is Sorth direction

If $0.5 \le w < 0.75$ move from (i,j) to (i,j+1), which is East direction

If $0.75 \le w < 1$ move from (i,j) to (i,j-1), which is West direction

The specific step is:

Step 0: Choose a finite number of simulations to run (i.e. random walks to complete). Call this number "numWalks". Initialize all necessary components.

(LOOP) For n=1,2,3,... total number of interior points do the following

Step 1: Choose an interior point (i,j) (for which the solution u(x_j,y_i) is still unknown). Let's call it A=(i,j)

for the time being. Initialize COUNTER=0.

Step 2a: Generate a random number w in [0,1] from the uniform distribution. Then start walking by following the method state above.

Step 2b: For each walk test if it attach the boundary. If not, repeat Step 2a. If so, stop the walk and record the boundary point you ended up at.

Step 3. Return to your chosen interior point A and repeat Step 2 until you have reached the total

number of walks to complete "numWalks".

Step 4: For your chosen interior point A, compute the fraction $P_A(p_k)$

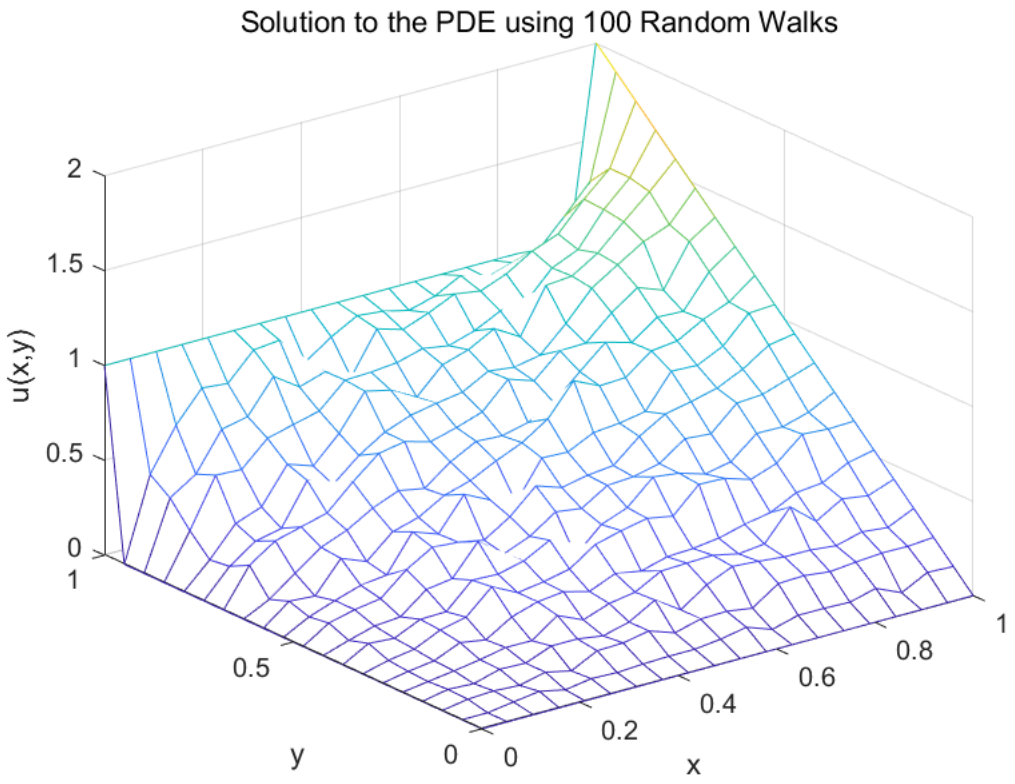
Step 5: Compute the approximate solution at the chosen interior point $u(A) = \sum_k g_k P_A(p_k)$

Step 6: Repeat Steps 1-5 for the remaining interior points.

End loop.

```
clear
% Step 0:

% Inputs
numWalks     = 100;%set numWaks
numPts       = 20; % number of discretization points in each direction on the board.

% Discretization
h            = (1-0)/(numPts-1);
x            = 0:h:1;
y            = 0:h:1;

% Set up the entire grid:
[xx,yy]      = meshgrid(x,y);

% Initialize:
u_sol        = zeros(size(xx));


% Enforce the BC u(x,1)=1;
u_sol(end,:) = 1; %BC, If y=1, the value of u(x,y)=1.
u_sol(:,end) = 2*y;%BC, u(1,y)=2y
u        = u_sol;
% Determine how many boundary points there are (not counting the corners of the board)
numBndry     = 4*(numPts-2);

% Loop over all of the interior points
for ii = 2:(numPts-1)
    for jj = 2:(numPts-1)
        % Initialize counter for the given point
        % to track how many walks end up at each boundary point.
        counter = zeros(size(xx));

        for nn = 1:numWalks % Walks loop
            % Step 1: Choose point A=(ii,jj)
            xPt = jj;
```

```
            yPt = ii;

            % Step 2:
            while xPt~=1 && xPt~=numPts && yPt~=1 && yPt~=numPts
                % Step 2b test if the walk attach to the boundary point
                w = rand;
                % Step 2a generate w~U(0,1)
                % WALK
                if w>=0 && w<0.25
                    yPt = yPt + 1; % walk north
                elseif w>=0.25 && w<0.50
                    yPt = yPt - 1; % walk south
                elseif w>=.5 && w<0.75
                    xPt = xPt + 1; % walk east
                elseif w>=0.75 && w<=1
                    xPt = xPt - 1; % walk west
                end
            end

            % Once the walk attach the boundary, Record number
            counter(yPt,xPt) = counter(yPt,xPt) + 1;
            % Step 4 compute the fraction
            counter_frac = counter/numWalks;

            % Step 5: Compute the approximate solution at the point A=(ii,jj)
            u(ii,jj)=sum(u_sol.*counter_frac,'all');
        end
    end
end

%plot
mesh(xx,yy,u);
xlabel('x'); ylabel('y');zlabel('u(x,y)')
title(['Solution to the PDE using ' num2str(numWalks) ' Random Walks'])
```

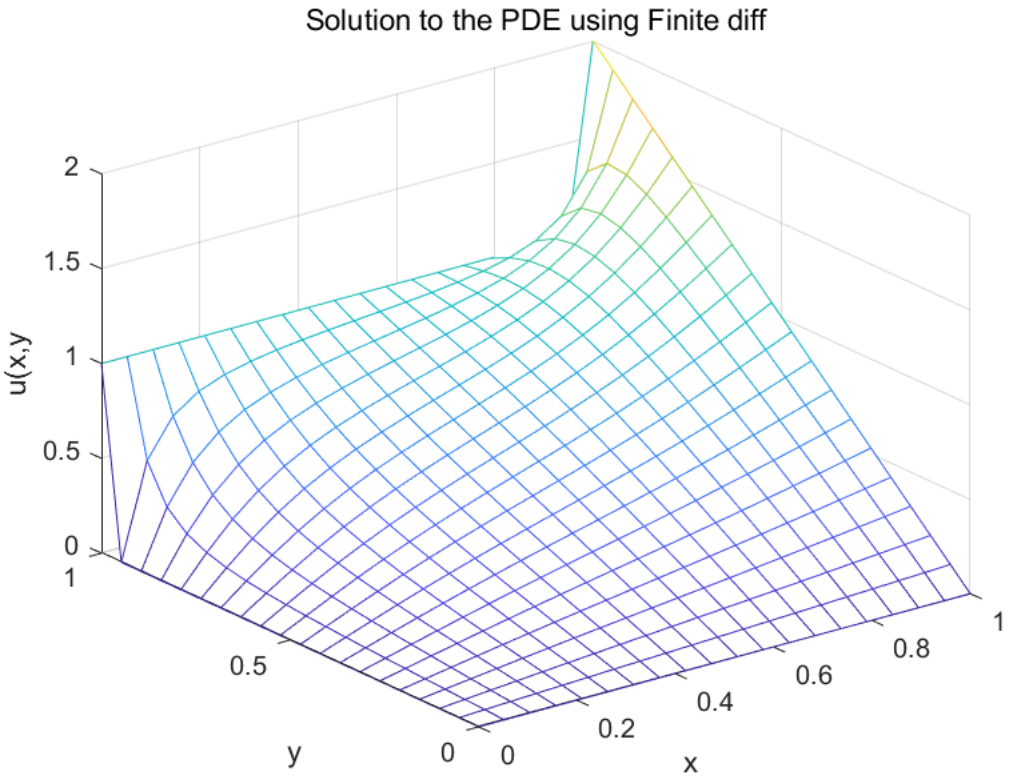


From the plot above we could see that, the surface is rough due to the method and number of random walk we use. with low number of random walk it will be hard to an accurate surface and solution we want.

Then using Finite Difference method

The method is same as the problem 4 from last homework, with different BCs.

```
[uk]=FD(numPts,numPts,u_sol);
%plot
mesh(xx,yy,uk)
xlabel('x'); ylabel('y');zlabel('u(x,y')
title(['Solution to the PDE using Finite diff'])
```



From both plot above, we could see that the shape of these two plots are almost same, Finite difference method will give us more smooth curve.

By accutual compute the difference between these two method

```
Total_diff = sum(u-uk,'all')
```

```
ans = 0.0579
```

```
Total_abs_diff=sum(abs(u-uk),'all')
```

```
ans = 10.9501
```

```
function [uk] = FD(nx, ny, u)
%function of processing finite difference
%set k
    k = [0 -1 0;
        -1 4 -1;
        0 -1 0];
    V = [];
    b = [];
%step 1 loop from (2,2) to (nx-1),(ny-1)
    for j = 2:ny - 1
        for l = 2:nx - 1
            %step 2 set all 0 matrix
            ut = zeros(ny, nx);
            %step 2 change the points around to k
```

```
            ut(j - 1:j + 1, l - 1:l + 1) = k;
            %step 3 cut the matrix from (2,2) to (nx-1),(ny-1)
            uc = ut(2:nx - 1, 2:ny - 1);
            %step 4 flat the cut matrix to 1 row and store it to V
            V = [V;reshape(uc', 1, (nx - 2)*(ny - 2))];
            %step 5 calculate b by adding the around point from IC matrix
            %together
            b = [b;u(j + 1, l) + u(j, l - 1) + u(j - 1, l) + u(j, l + 1)];
        end
    end
    uin = V\b;%step 6 calculate the result.
    u(2:nx - 1, 2:ny - 1) = reshape(uin, (nx - 2), (ny - 2))';
    %put the solution back into IC matrix.
    uk = u;
end
```

4. Write a computer program to carry out the Monte Carlo Random Walk solution method for following BVP

PDE $u_{xx} + x^2 u_{yy} = 0 \qquad 0 < x < 1, 0 < y < 1$

BCs $u(0, y) = 0 \qquad 0 < y < 1$

$u(1, y) = 0 \qquad 0 < y < 1$

$u(x, 0) = 5 \qquad 0 < x < 1$

$u(x, 1) = 1 \qquad 0 < x < 1$

You may use a uniform grid size h=k or let the x grid size (h) and y grid size (k) be diffierent. How accurate is your solution? How do you know? Also, code the problem using the Finite Difference method. Compare your solution to that from MC. Discuss.

Since the PDE has been changed, the the probability that the walks goes to each way are different

Consider

$$u_{xx} = \frac{u_{i,j+1} - 2u_{ij} + u_{i,j-1}}{h^2}$$

$$u_{yy} = \frac{u_{i+1,j} - 2u_{ij} + u_{i-1,j}}{h^2}$$

$$x^2 = x_j^2$$

$$\frac{u_{i,j+1} - 2u_{ij} + u_{i,j-1}}{h^2} + x_j^2 \frac{u_{i+1,j} - 2u_{ij} + u_{i-1,j}}{h^2} = 0$$

If we solve that

$$u_{i,j} = \frac{u_{i,j+1} + u_{i,j-1} + x_j^2 [u_{i+1,j} + u_{i-1,j}]}{2(1 + x_j^2)} \qquad (3)$$

So according to the coefficients of each point, the probability could be determined by the coefficients:

with $P_N = \frac{x_j^2}{2(1 + x_j^2)}$,move from (i,j) to (i+1,j), which is North direction

with $P_S = \frac{x_j^2}{2(1 + x_j^2)}$,move from (i,j) to (i-1,j), which is Sorth direction

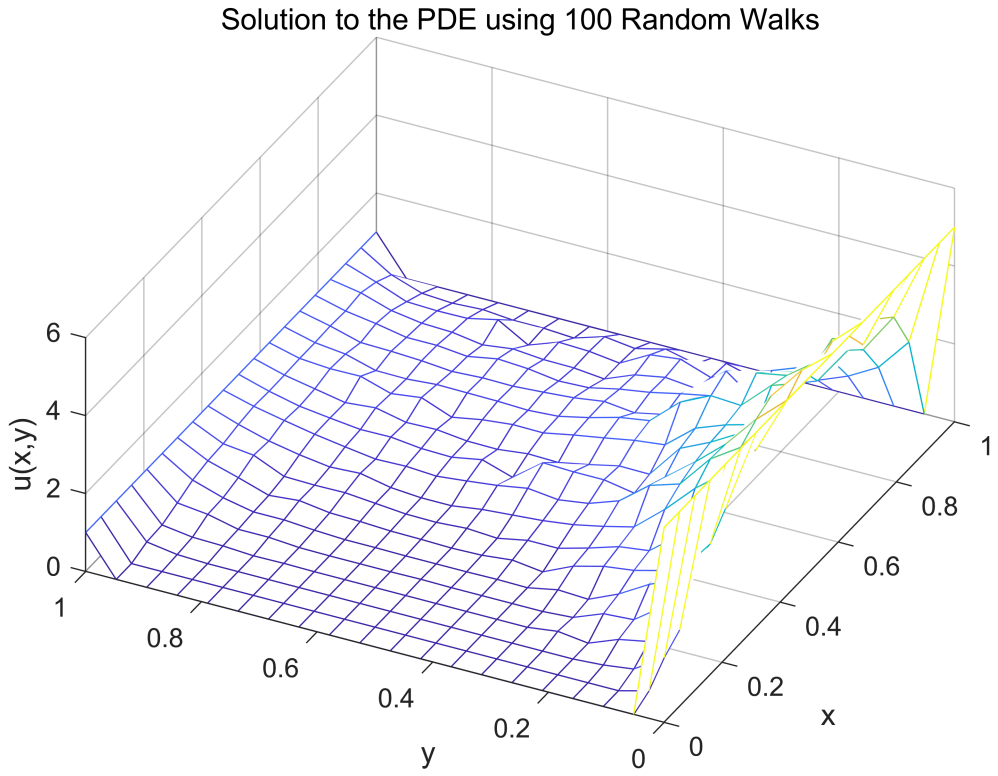
with $P_E = \frac{1}{2(1 + x_j^2)}$,move from (i,j) to (i,j+1), which is East direction

with $P_W = \frac{1}{2(1 + x_j^2)}$,move from (i,j) to (i,j-1), which is West direction

```
clear
```

```
% Step 0:

% Inputs
numWalks     = 100;%set numWaks
numPts       = 20; % number of discretization points in each direction on the board.

% Discretization
h            = (1-0)/(numPts-1);
x            = 0:h:1;
y            = 0:h:1;

% Set up the entire grid:
[xx,yy]      = meshgrid(x,y);

% Initialize:
u_sol        = zeros(size(xx));


% Enforce the BC u(x,1)=1;
u_sol(end,:) = 1; % If y=1, the value of u(x,y)=1.
u_sol(1,:) = 5;%BC, u(x,0)=5
u        = u_sol;
ufe=u_sol;

% Determine how many boundary points there are (not counting the corners of the board)
numBndry     = 4*(numPts-2);

% Loop over all of the interior points
for ii = 2:(numPts-1)
    for jj = 2:(numPts-1)
        % Initialize counter for the given point
        % to track how many walks end up at each boundary point.
        counter = zeros(size(xx));

        for nn = 1:numWalks % Walks loop
            % Step 1: Choose point A=(ii,jj)
            xPt = jj;
            yPt = ii;

            %set probability of each direction
            N=x(jj)^2/(2*(1+x(jj)^2));%P_n
            S=N;%P_S
            E=1/(2*(1+x(jj)^2));%P_E
            W=E;%P_W

            % Step 2:
            while xPt~=1 && xPt~=numPts && yPt~=1 && yPt~=numPts
                % Step 2b test if the walk attach to the boundary point
                w = rand; % random number w in [0,1] from the uniform distribution
                % Step 2a generate w~U(0,1)
                % WALK
                if w>=0 && w<N
                    %test if w between 0 and P_N
```

```
                    yPt = yPt + 1; % walk north
                elseif w>=N && w<N+S
                    %test if w between P_N and P_N+P_S
                    yPt = yPt - 1; % walk south
                elseif w>=N+S && w<N+S+E
                    %test if w between P_N+P_S and P_N+P_S+P_E
                    xPt = xPt + 1; % walk east
                elseif w>=N+S+E && w<=1
                    %test if w between P_N+P_S+P_E and 1
                    xPt = xPt - 1; % walk west
                end
            end

            % Once the walk attach the boundary, Record number
            counter(yPt,xPt) = counter(yPt,xPt) + 1;
            % Step 4 compute the fraction
            counter_frac = counter/numWalks;

            % Step 5: Compute the approximate solution at the point A=(ii,jj)
            u(ii,jj)=sum(u_sol.*counter_frac,'all');
        end
    end
end

%plot
mesh(xx,yy,u);
xlabel('x'); ylabel('y');zlabel('u(x,y)')
title(['Solution to the PDE using ' num2str(numWalks) ' Random Walks'])
view([-63.33 55.19])
```



From the plot above we could see that, the surface is rough due to the method and number of random walk we use. with low number of random walk it will be hard to an accurate surface and solution we want.

Then using Finite Difference method

The method is same as the problem 5 from last homework, with different BCs.

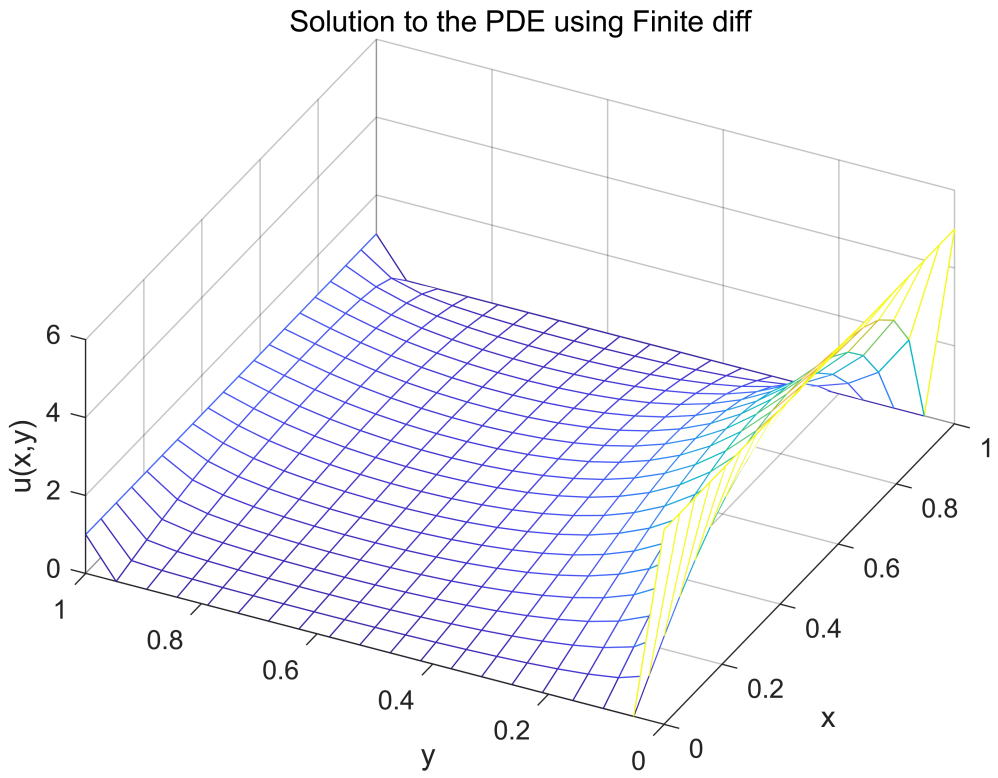
From equaiton (3)

$u_{i,j}$is the weighted sumation of the point around it.

We could adjust

$$k = \begin{bmatrix} 0 & -x_j^2 & 0 \\ -1 & 2\left(1+x_j^2\right) & -1 \\ 0 & -x_j^2 & 0 \end{bmatrix} \qquad (4)$$

```
[uk]=FD(numPts,numPts,u_sol,x);
%plot
mesh(xx,yy,uk)
xlabel('x'); ylabel('y');zlabel('u(x,y)')
title(['Solution to the PDE using Finite diff'])
view([-63.33 55.19])
```



From both plot above, we could see that the shape of these two plots are almost same, Finite difference method will give us more smooth curve.

By accutual compute the difference between these two method

```
Total_diff = sum(u-uk,'all')
```

```
Total_diff = -17.8126
```

```
Total_abs_diff = sum(abs(u-uk),'all')
```

```
Total_abs_diff = 61.5390
```

With the different PDE and BCs the accuracy also decreased.

```
function [uk]=FD(nx,ny,u,x)
    V=[];
    b=[];
    for j=2:ny-1
        for l=2:nx-1
            k=[0 -(x(l))^2 0;
                -1 2*(1+(x(l))^2) -1;
                0 -(x(l))^2 0];
            %new k from equation(4)
            ut=zeros(ny,nx);
            ut(j-1:j+1,l-1:l+1)=k;
            uc=ut(2:nx-1,2:ny-1);
```

```
            V=[V;reshape(uc', 1,(nx-2)*(ny-2))];
            b=[b;(x(l))^2*u(j+1,l)+u(j,l-1)+(x(l))^2*u(j-1,l)+u(j,l+1)];
            %each term in b is the weighted sumation of the point around it
        end
    end
    uin=V\b;
    u(2:nx-1,2:ny-1)=reshape(uin,(nx-2),(ny-2))';
    uk=u;
end
```

5. Let $u(x) = \frac{1}{x+1}$. Approximate u on [0,2] using n equally spaced subintervals for n=2,5,10 using

a) polynomial interpolation with the Vandermonde matrix and a polynomial of degree n

b) polynomial interpolation with the the Lagrange formula and a polynomial of degree n

Plot your results and discuss.

a) By using the Vandermonde martix

In order to find polynomial that pas throug all the point are given, we need to first set up a polynomial:

$$P(x) = a_0 + a_1 x + a_2 x^2 + \ldots + a_k x^k \qquad (4)$$

Then plug all the points given into equation (4)

Suppose the points are:

$$(x_0, y_0)(x_2, y_2)(x_3, y_3)\ldots(x_k, y_k)$$

As a result we will get:

$$\begin{cases} P(x_0) = a_0 + a_1 x_0 + a_2 {x_0}^2 + \ldots + a_k {x_0}^k = y_0 \\ P(x_1) = a_0 + a_1 x_1 + a_2 {x_1}^2 + \ldots + a_k {x_1}^k = y_1 \\ \qquad\qquad\qquad\qquad \vdots \\ P(x_k) = a_0 + a_1 x_k + a_2 {x_k}^2 + \ldots + a_k {x_k}^k = y_k \end{cases}$$

And if we write them as a matrix:

$$\begin{bmatrix} 1 & x_0 & x_0^2 & x_0^3 & \cdots & x_0^k \\ 1 & x_1 & x_1^2 & x_1^3 & \cdots & x_1^k \\ 1 & x_2 & x_2^2 & x_2^3 & \cdots & x_2^k \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_k & x_k^2 & x_k^3 & \cdots & x_k^k \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ \vdots \\ a_k \end{bmatrix} = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ \vdots \\ y_k \end{bmatrix} \qquad (5)$$

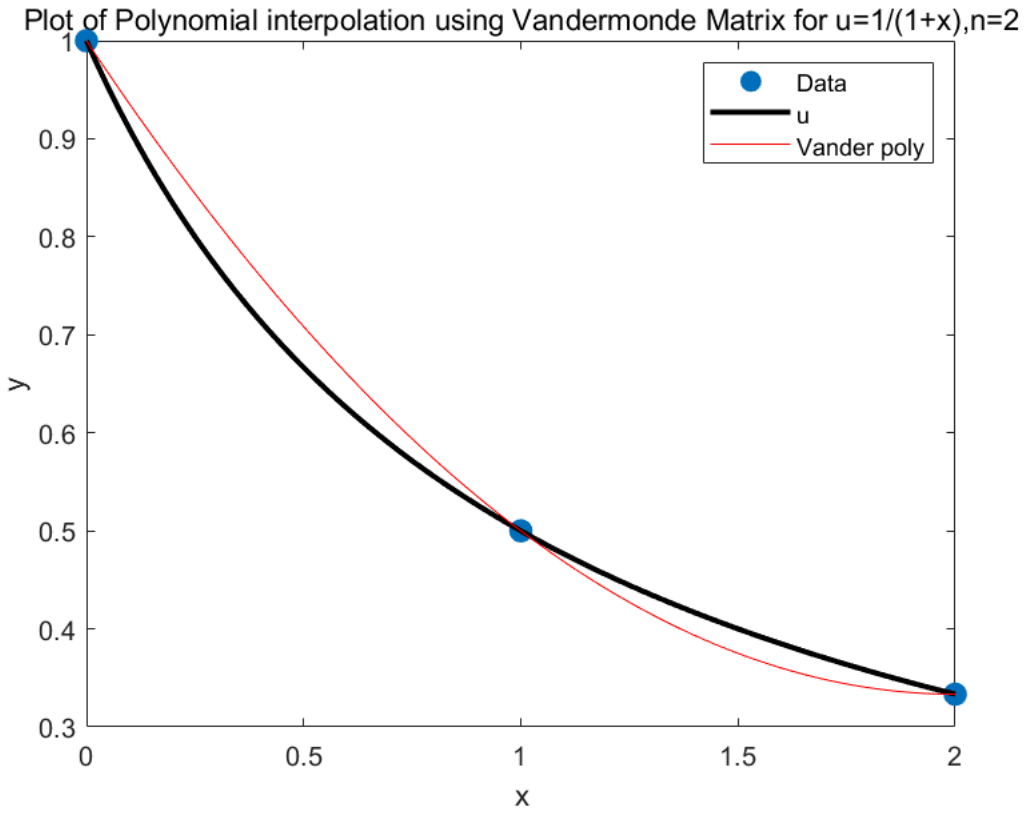
Let

$$V = \begin{bmatrix} 1 & x_0 & x_0^2 & x_0^3 & \cdots & x_0^k \\ 1 & x_1 & x_1^2 & x_1^3 & \cdots & x_1^k \\ 1 & x_2 & x_2^2 & x_2^3 & \cdots & x_2^k \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ \vdots & \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_k & x_k^2 & x_k^3 & \cdots & x_k^k \end{bmatrix}, \; a = \begin{bmatrix} a_0 \\ a_1 \\ a_2 \\ \vdots \\ \vdots \\ a_k \end{bmatrix} \text{ and } y = \begin{bmatrix} y_0 \\ y_1 \\ y_2 \\ \vdots \\ \vdots \\ y_k \end{bmatrix}$$

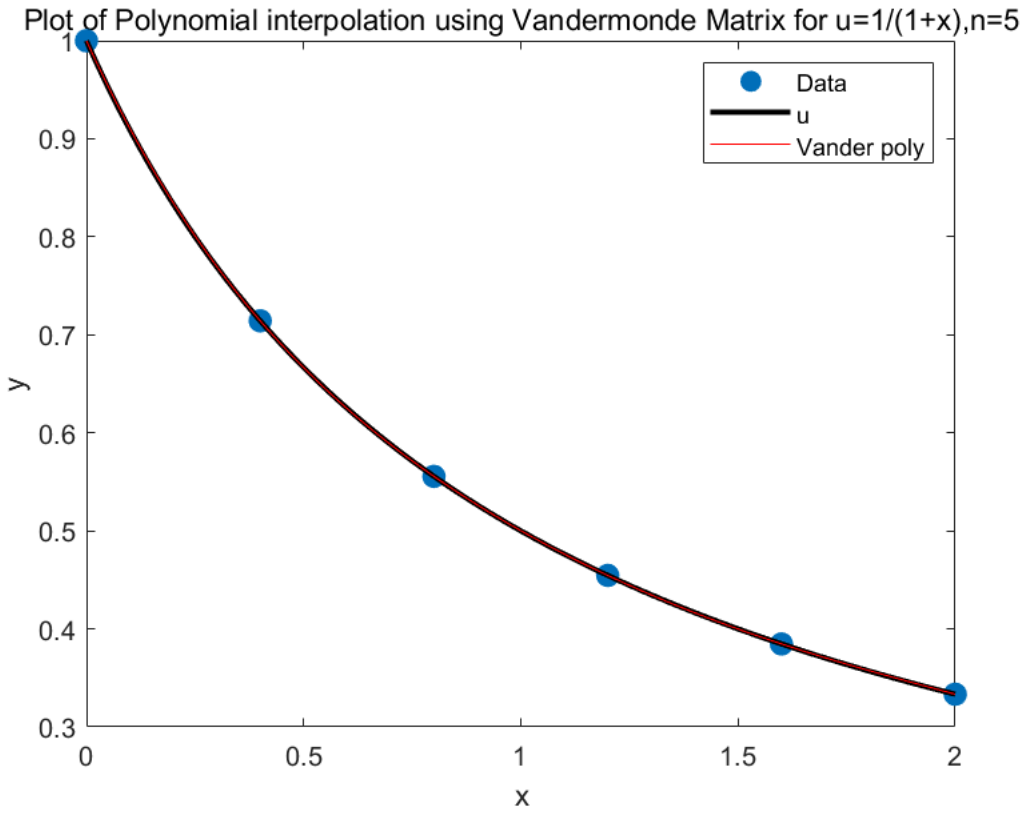
equation (5) can be written as $\mathrm{Va} = y$

V is called Vandermonde

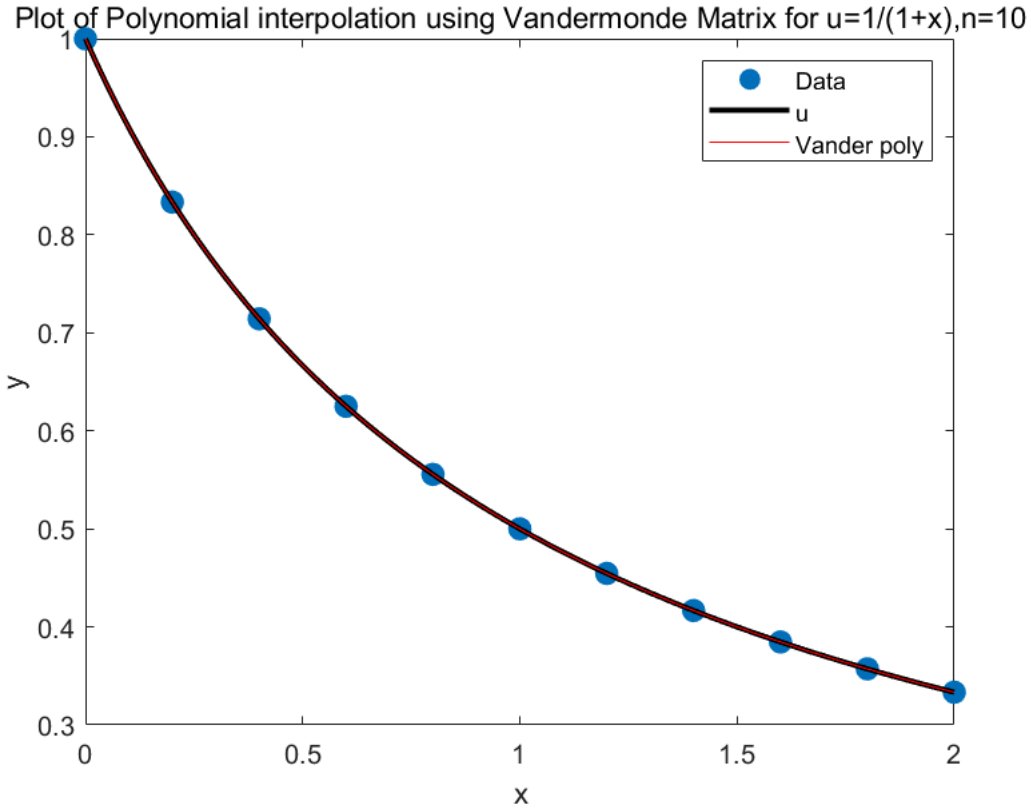
```
clear
%define the interval
a=0;
b=2;

%Vandermonde matrix
n = 2; % number of intervals
[x2V,y2V,xfine2V,u_Vander_fine2V,a_vec2V]=van(a,b,n);
plotA(x2V,y2V,xfine2V,u_Vander_fine2V,n)
```




```
n = 5; % number of intervals
[x5V,y5V,xfine5V,u_Vander_fine5V,a_vec5V]=van(a,b,n);
plotA(x5V,y5V,xfine5V,u_Vander_fine5V,n)
```




```
n = 10; % number of intervals
[x10V,y10V,xfine10V,u_Vander_fine10V,a_vec10V]=van(a,b,n);
plotA(x10V,y10V,xfine10V,u_Vander_fine10V,n)
```



From the graph above we could see as the interval point increasing the polynomial are almost become coincide with the original function u(x).

b)By using the Lagrange fomular

The Lagrange fomular is:

$$y_k = p(x_k) \qquad (6)$$

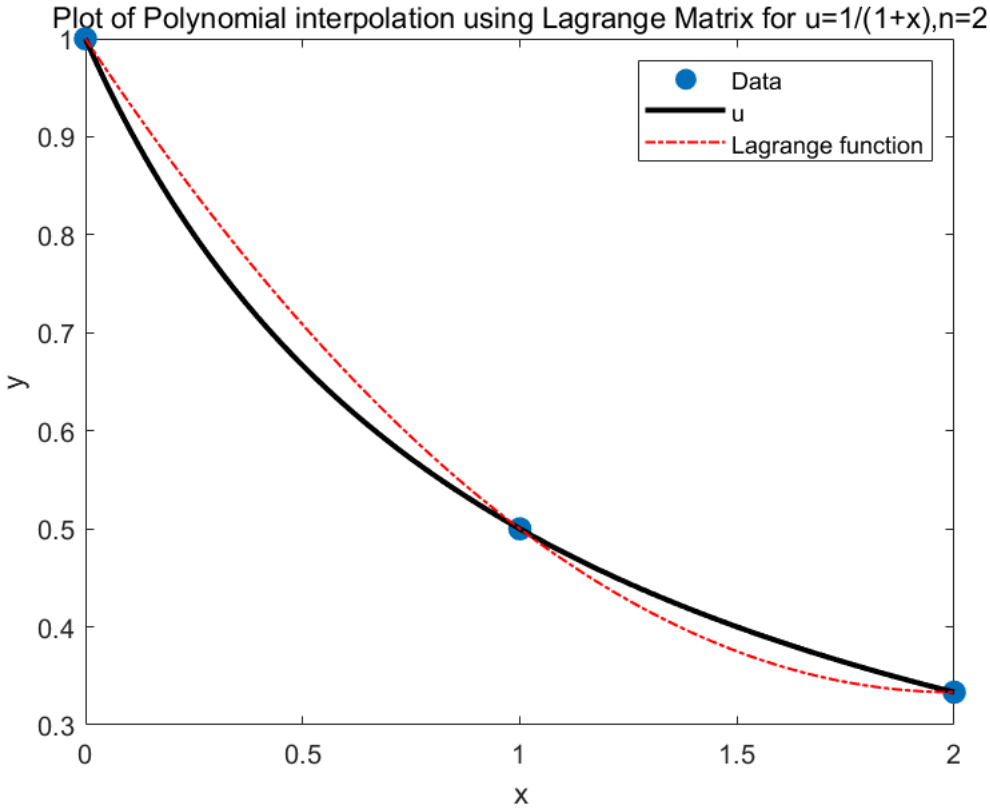
$$p(x_k) = y_0\ell_0(x) + y_1\ell_1(x) + \cdots + y_n\ell_n(x) \qquad (7)$$

$$\ell_0(x) = \frac{(x-x_0)(x-x_1)\cdots(x-x_{k-1})(x-x_{k+1})\cdots(x-x_n)}{(x_k-x_0)(x_k-x_1)\cdots(x_k-x_{k-1})(x_k-x_{k+1})\cdots(x_k-x_n)}$$

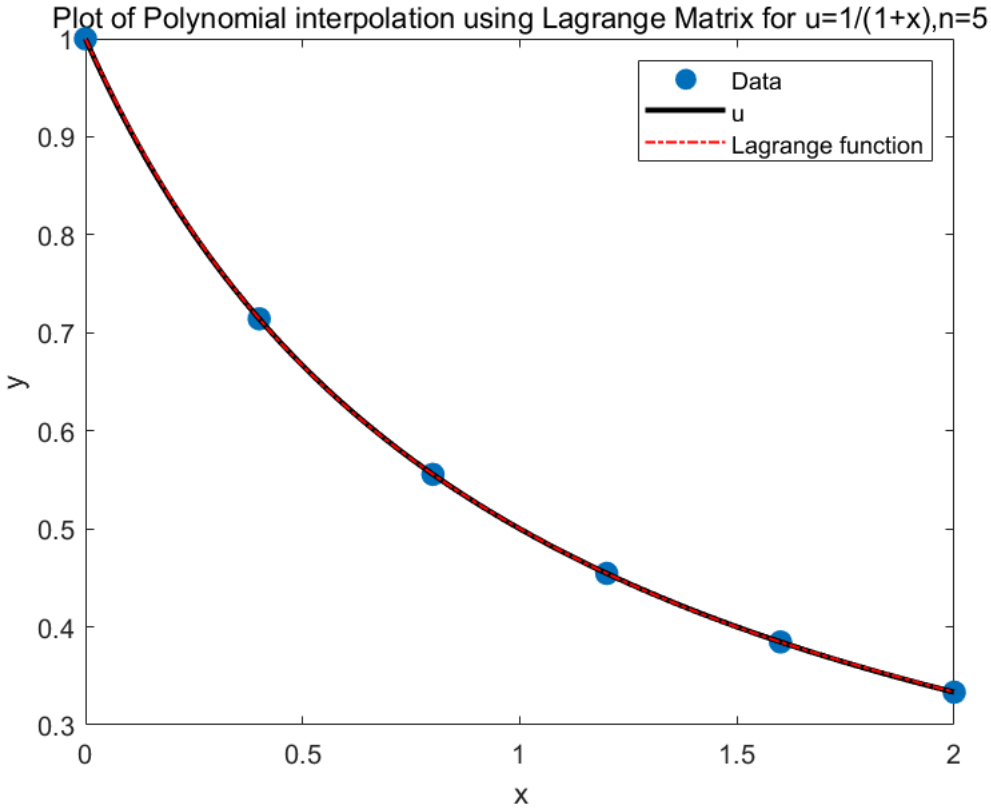
where can be write as

$$\ell_j(x) = \prod_{k=0,k\neq j}^{n} \frac{(x-x_k)}{(x_j-x_k)} \qquad (8)$$

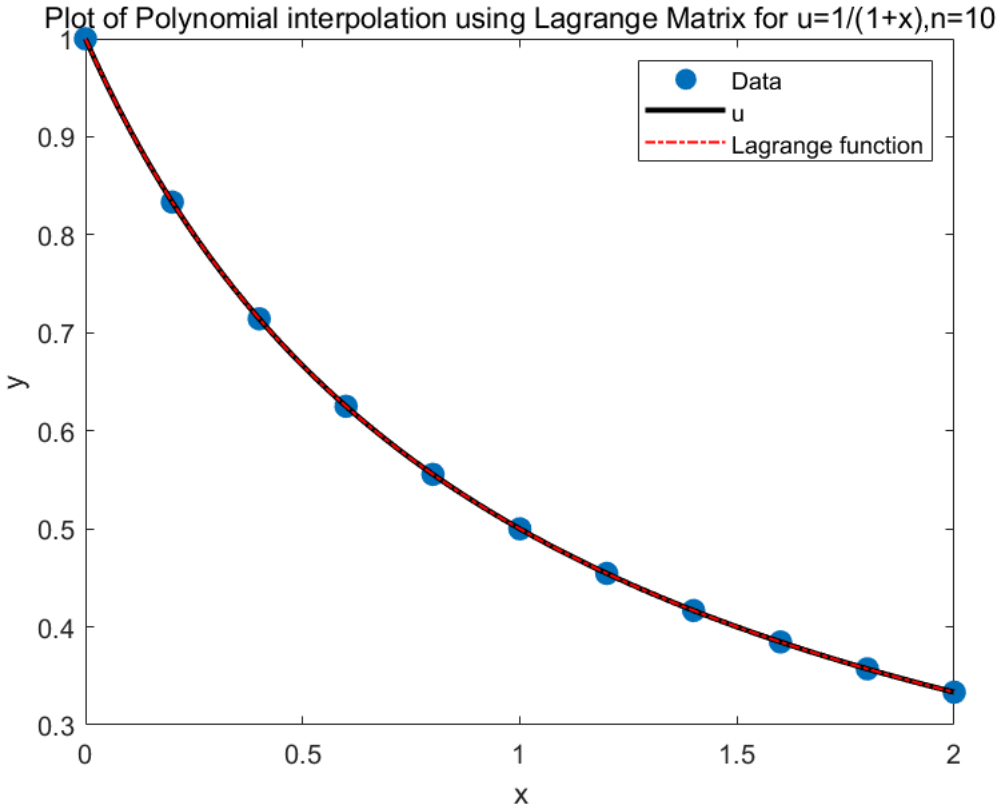
```
%Lagrange fomular
n = 2; % number of intervals
[x2L,y2L,xfine2L,u_Lagrange2L]=lan(a,b,n);
plotAN(x2L,y2L,xfine2L,u_Lagrange2L,n)
```




```
n = 5; % number of intervals
[x5L,y5L,xfine5L,u_Lagrange5L]=lan(a,b,n);
plotAN(x5L,y5L,xfine5L,u_Lagrange5L,n)
```




```
n = 10; % number of intervals
[x10L,y10L,xfine10L,u_Lagrange10L]=lan(a,b,n);
plotAN(x10L,y10L,xfine10L,u_Lagrange10L,n)
```



From the graph above we could see as that the polynomial compute from Lagrange method is also looks like coinside with the true value;

By compute the error of them

```
err_Vandermonde_n_2 = sum(abs(u_Vander_fine2V-u(xfine2V)),'all')
```

```
err_Vandermonde_n_2 = 4.5646
```

```
err_Vandermonde_n_5 = sum(abs(u_Vander_fine5V-u(xfine5V)),'all')
```

```
err_Vandermonde_n_5 = 0.0643
```

```
err_Vandermonde_n_10 = sum(abs(u_Vander_fine10V-u(xfine10V)),'all')
```

```
err_Vandermonde_n_10 = 1.3101e-04
```

```
err_Lagrange_n_2 = sum(abs(u_Lagrange2L-u(xfine2L)),'all')
```

```
err_Lagrange_n_2 = 4.5646
```

```
err_Lagrange_n_5 = sum(abs(u_Lagrange5L-u(xfine5L)),'all')
```

```
err_Lagrange_n_5 = 0.0643
```

```
err_Lagrange_n_10 = sum(abs(u_Lagrange10L-u(xfine10L)),'all')
```

```
err_Lagrange_n_10 = 1.3101e-04
```

As result, we could find that, as n increading the error of both approximation decreased.

```
function [x,y,xfine,u_Vander_fine,a_vec]=van(a,b,n)
%function of computing the Vandermonde martix and correspounding
%coefficients
%take input a,b,and n
%out put x: the x coodinate of data point according to n
%out put y: the y coodinate of data point that compute from x
%xfine: a finer grid of x
%u_Vander_fine: approximate the polynomial using the coefficient compute from V and y
%a_vec: vector a
    h = (b-a)/n;
    x = a + (0:n)*h; % define the x values
    y = u(x); % define the corresponding y values as y_j = u(x_j)
    Vmat = zeros(length(x));
    for ii=1:length(x)
        % loop over the data points
        for jj=1:length(x)
            % loop over the columns of the Vandermonde martix
            Vmat(ii,jj) = x(ii)^(jj-1);
            % since matlab indices starts at 1, so we need to subtract 1
        end
    end
    rr_form     = rref([Vmat,y(:)]);
    a_vec       = rr_form(:,end); % peeling off the last column
    xfine         = a:0.01:b;%define a finer grid x
    u_Vander_fine = a_vec(1)*ones(size(xfine));
    for jj=1:n
        %compute the polynomial from vector a
        u_Vander_fine = u_Vander_fine + a_vec(jj+1)*xfine.^(jj);
    end
end



function [x,y,xfine,u_Lagrange]=lan(a,b,n)
%function of compute the Lagrange fomular
%input a, b and n
%outpout
%out put x: the x coodinate of data point according to n
%out put y: the y coodinate of data point that compute from x
%xfine: a finer grid of x
%u_Lagrange:  approximate the polynomial
% using the coefficient compute Lagrange fomular
    h = (b-a)/n;
    x = a + (0:n)*h; % define the x values
    y = u(x); % define the corresponding y values as y_j = u(x_j)
    N          = length(x);
    xfine       = a:0.01:b; % define the finer x values
    u_Lagrange  = zeros(size(xfine));
    for k = 1:N
        %loop each k from equation (6)
```

```
        w = ones(size(xfine));
        for j = [1:k-1 k+1:N] % This loop the index from equation (8)
            w = (xfine-x(j))./(x(k)-x(j)).*w;
        end
        u_Lagrange = u_Lagrange + w*y(k);
        % This adds up each y_k*l_k(x) term, equation (7)
    end
end

function []=plotA(x,y,xfine,u_Vander_fine,n)
%function of plot the polynomial from Vandermonde martix
%take input:
%x: the x coodinate of data point according to n
%y: the y coodinate of data point that compute from x
%xfine: a finer grid of x
%u_Vander_fine: approximate the polynomial using the coefficient compute from V and y
%n
    plot(x,y,'.','MarkerSize', 30)%plot the data
    hold on;
    plot(xfine,u(xfine),'k', 'LineWidth',2)%plot the true value
    plot(xfine, u_Vander_fine, 'r')%plot the polynomial
    xlabel('x');ylabel('y');
    legend('Data','u', 'Vander poly')
    title(['Plot of Polynomial interpolation using ' ...
        'Vandermonde Matrix for u=1/(1+x),n=',num2str(n)])
    hold off
end

function []=plotAN(x,y,xfine,u_Lagrange,n)
%function of plot the polynomial from Lagrange fomular
%take input:
%x: the x coodinate of data point according to n
%y: the y coodinate of data point that compute from x
%xfine: a finer grid of x
%u_Lagrange: approximate the polynomial using the coefficient compute Lagrange
%n
    plot(x,y,'.','MarkerSize',30);%plot the data
    hold on;
    plot(xfine,u(xfine),'k', 'LineWidth',2);%plot the true value
    plot(xfine,u_Lagrange,'-.r','LineWidth',1)%plot the polynomial
    xlabel('x');ylabel('y');
    legend('Data','u','Lagrange function')
    title(['Plot of Polynomial interpolation using' ...
        ' Lagrange Matrix for u=1/(1+x),n=',num2str(n)])
    hold off
end

function [ut]=u(x)
%true function
    ut=1./(x+1);
end
```

6. For the fundemental B-spline discussed in the lecture, check that $B(t), B'(t)$ and $B''(t)$ are continous across the nodes(knots).

$$B(t) = \begin{cases} \frac{1}{6}t^3 & 0 \le t \le 1 \quad (9) \\ \frac{1}{6}[-3(t-1)^3 + 3(t-1)^2 + 3(t-1) + 1] & 1 \le t \le 2 \quad (10) \\ \frac{1}{6}[3(t-2)^3 - 6(t-2)^2 + 4] & 2 \le t \le 3 \quad (11) \\ \frac{1}{6}[-(t-3)^3 + 3(t-3)^2 - 3(t-3) + 1] & 3 \le t \le 4 \quad (12) \\ 0 & t \le 0 \text{ or } t \ge 4 \end{cases}$$

PLot $B(t)$

$$B_1(1) = \frac{1}{6}$$

$$B_2(1) = \frac{1}{6}$$

$$B_2(2) = \frac{1}{6}[-3(1)^3 + 3(1)^2 + 3(1) + 1] = \frac{1}{6}[-3 + 3 + 3 + 1] = \frac{1}{6}[4] = \frac{4}{6}$$

$$B_3(2) = \frac{1}{6}[3(t-2)^3 - 6(t-2)^2 + 4] = \frac{4}{6}$$

$$B_3(3) = \frac{1}{6}[3(1)^3 - 6(1)^2 + 4] = \frac{1}{6}[3 - 6 + 4] = \frac{1}{6}[1] = \frac{1}{6}$$

$$B_4(3) = \frac{1}{6}[-(t-3)^3 + 3(t-3)^2 - 3(t-3) + 1] = \frac{1}{6}$$

$$B_4(4) = \frac{1}{6}[-(1)^3 + 3(1)^2 - 3(1) + 1] = \frac{1}{6}[-1 + 3 - 3 + 1] = \frac{1}{6}[0] = 0$$

$$B_5(4) = 0$$

$$B_5(0) = 0$$

$$B_1(0) = 0$$

So $B(t)$ is continous across the nodes

$$B'(t) = \begin{cases} \frac{1}{2}t^2 & 0 \le t \le 1 \\ \frac{1}{6}[-9(t-1)^2 + 6(t-1) + 3] & 1 \le t \le 2 \\ \frac{1}{6}[9(t-2)^2 - 12(t-2)] & 2 \le t \le 3 \\ \frac{1}{6}[-3(t-3)^2 + 6(t-3) - 3] & 3 \le t \le 4 \\ 0 & t \le 0 \text{ or } t \ge 4 \end{cases}$$

$B_1'(1) = \frac{1}{2}$

$B_2'(1) = \frac{1}{3}$

$B_2'(2) = \frac{1}{6}[-9 + 6 + 3] = \frac{1}{6}[0] = 0$

$B_3'(2) = 0$

$B_3'(3) = \frac{1}{6}[9 - 12] = -\frac{1}{2}$

$B_4'(3) = -\frac{1}{2}$

$B_4'(4) = \frac{1}{6}[-3 + 6 - 3] = 0$

$B_5'(4) = 0$

$B_5'(0) = 0$

$B_1'(0) = 0$

So $B'(t)$ is continous across the nodes

$$B''(t) = \begin{cases} t & 0 \le t \le 1 \\ \frac{1}{6}[-18(t-1) + 6] & 1 \le t \le 2 \\ \frac{1}{6}[18(t-2) - 12] & 2 \le t \le 3 \\ \frac{1}{6}[-6(t-3) + 6] & 3 \le t \le 4 \\ 0 & t \le 0 \text{ or } t \ge 4 \end{cases}$$

$B_1''(1) = 1$

$B_2''(1) = \frac{1}{6}[6] = 1$

$$B_2''(2) = \frac{1}{6}[-18 + 6] = \frac{1}{6}[-12] = -2$$

$$B_3''(2) = \frac{1}{6}[-12] = -2$$

$$B_3''(3) = \frac{1}{6}[18 - 12] = \frac{1}{6}[6] = 1$$

$$B_4''(3) = \frac{1}{6}[6] = 1$$

$$B_4''(4) = \frac{1}{6}[-6 + 6] = 0$$

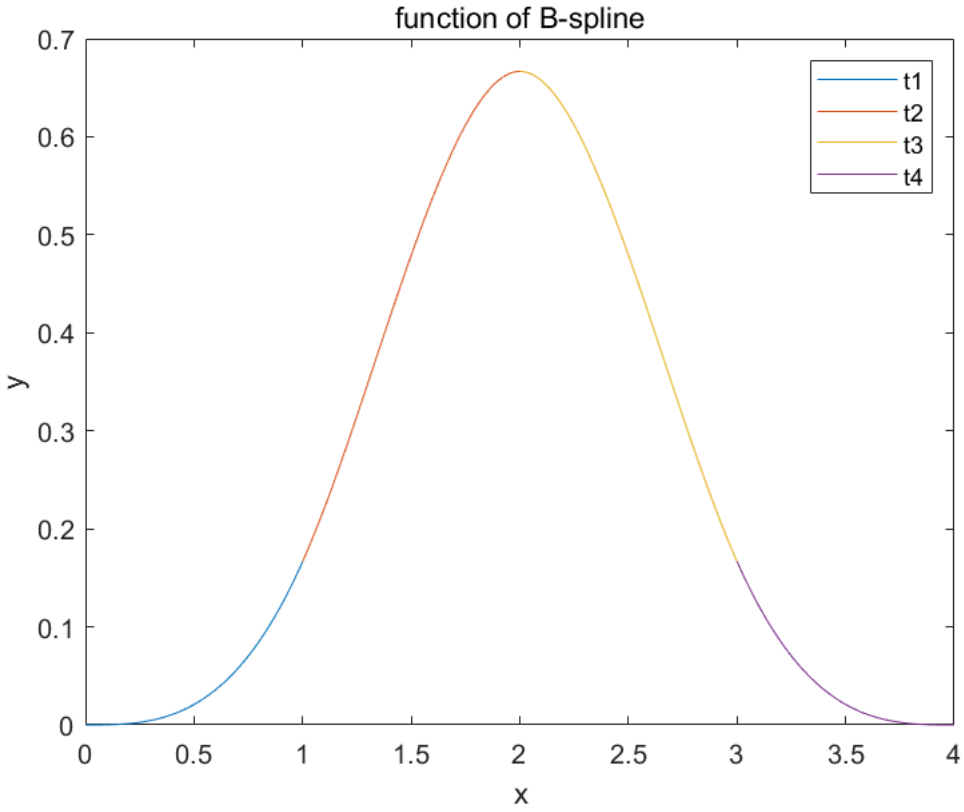
$$B_5''(4) = 0$$

$$B_5''(0) = 0$$

$$B_1''(0) = 0$$

So $B''(t)$ is continous across the nodes

```
clear
%define n
n=0.01;
%define t1,t2,t3 and t4
t1=0:n:1;
t2=1:n:2;
t3=2:n:3;
t4=3:n:4;

%plot
plot(t1,B1(t1))
hold on
plot(t2,B2(t2))
plot(t3,B3(t3))
plot(t4,B4(t4))
xlabel('x')
ylabel('y')
title('function of B-spline')
legend('t1','t2','t3','t4')
hold off
```




```
function [Bt1]=B1(t)
%function of equation (9)
    Bt1=1/6*t.^3;
end

function [Bt2]=B2(t)
%function of equation (10)
    Bt2=1/6*(-3*(t-1).^3+3*(t-1).^2+3*(t-1)+1);
end

function [Bt3]=B3(t)
%function of equation (11)
    Bt3=1/6*(3*(t-2).^3-6*(t-2).^2+4);
end

function [Bt4]=B4(t)
%function of equation (12)
    Bt4=1/6*(-(t-3).^3+3*(t-3).^2-3*(t-3)+1);
end
```

7. Let $u(x) = \frac{1}{x+1}$ on [0,2]. Approximate u on [0,2] using n equally spaced subintervals for n=2,5,10 using.

a) n cubic splines with the polynomial approach

b) n cubic splines with B-spline approach

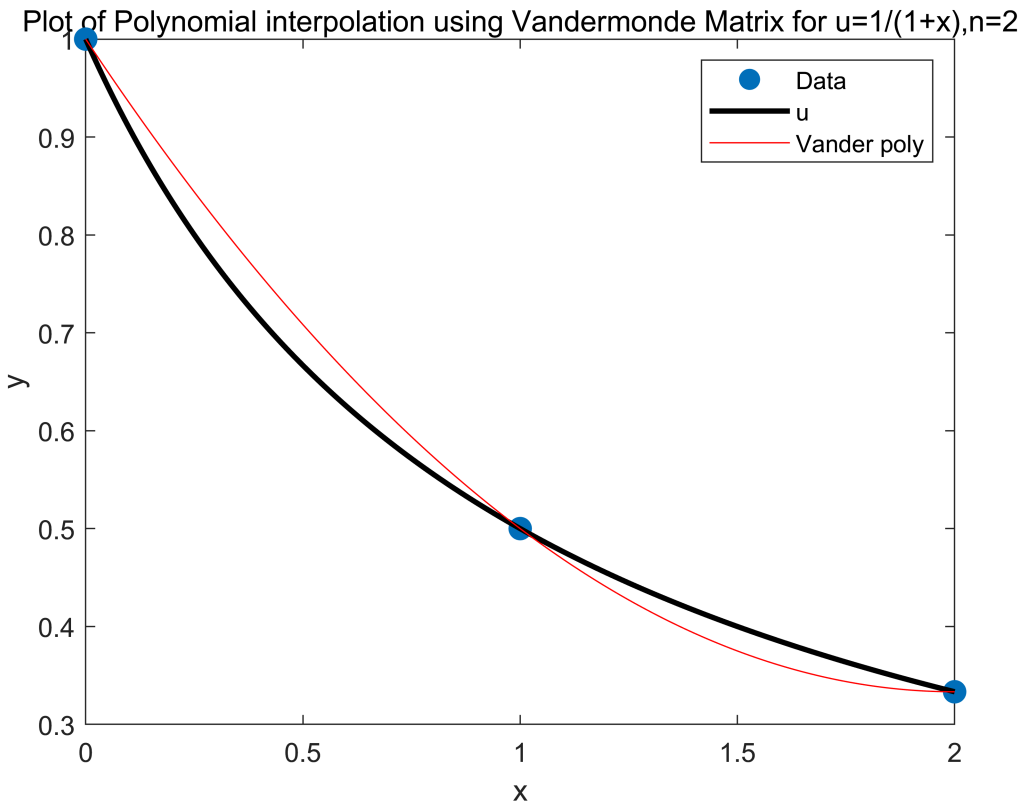
c) n cubic using the spline command in matlab.

Plot your results and compare to those from H1. How do your results change if you use a 'natural' spline or a 'clamped' spline.

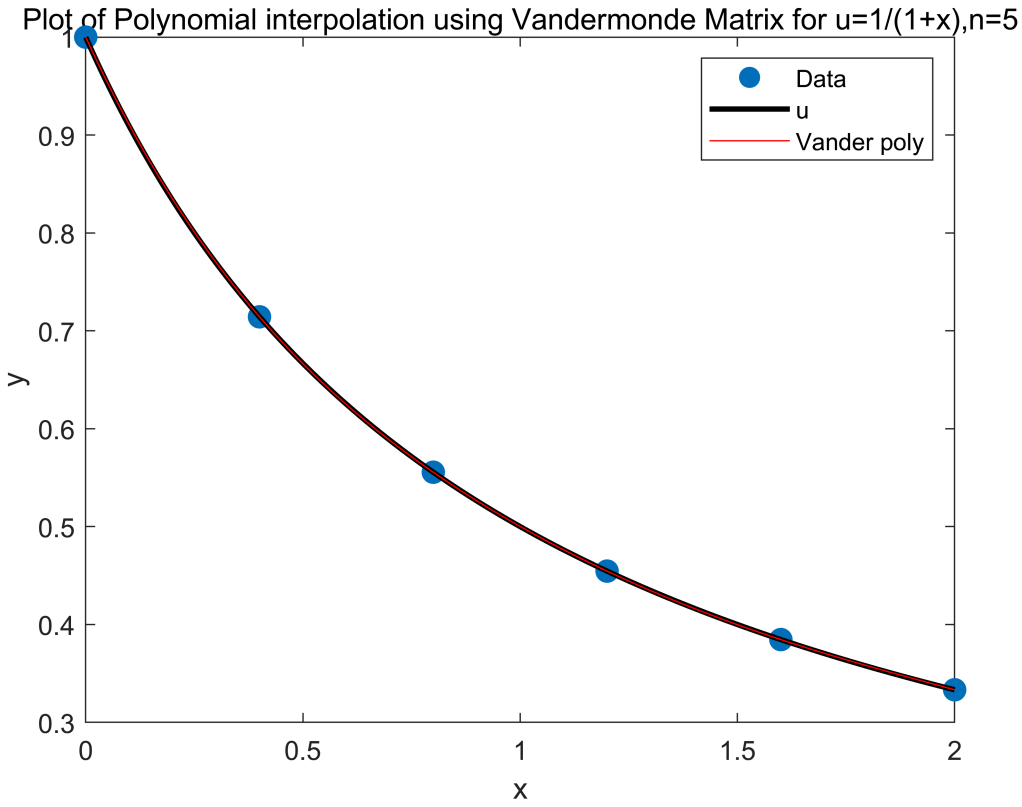
a)

We will use the same method with problem 5.

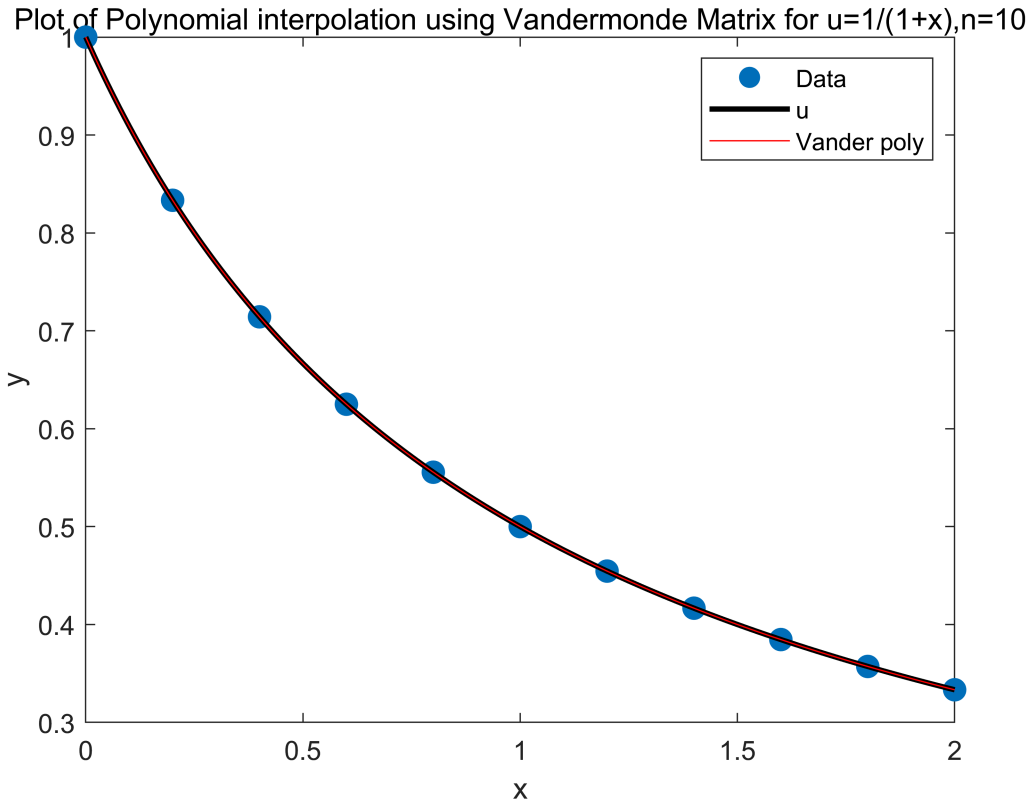
```
clear
a=0;
b=2;
x=a:0.01:b;
ut=u(x);%real

%a)
%Vandermonde matrix
n = 2; % number of intervals
[x2V,y2V,xfine2V,u_Vander_fine2V,a_vec2V]=van(a,b,n);
plotA(x2V,y2V,xfine2V,u_Vander_fine2V,n)
```



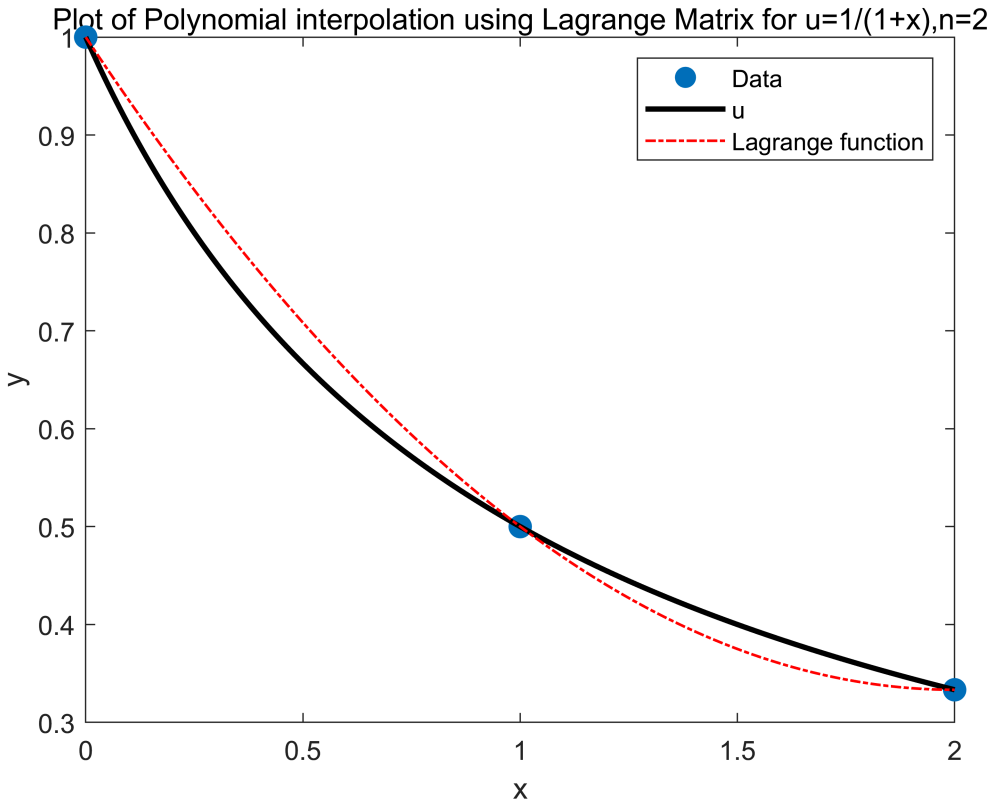
```
n = 5; % number of intervals
[x5V,y5V,xfine5V,u_Vander_fine5V,a_vec5V]=van(a,b,n);
plotA(x5V,y5V,xfine5V,u_Vander_fine5V,n)
```



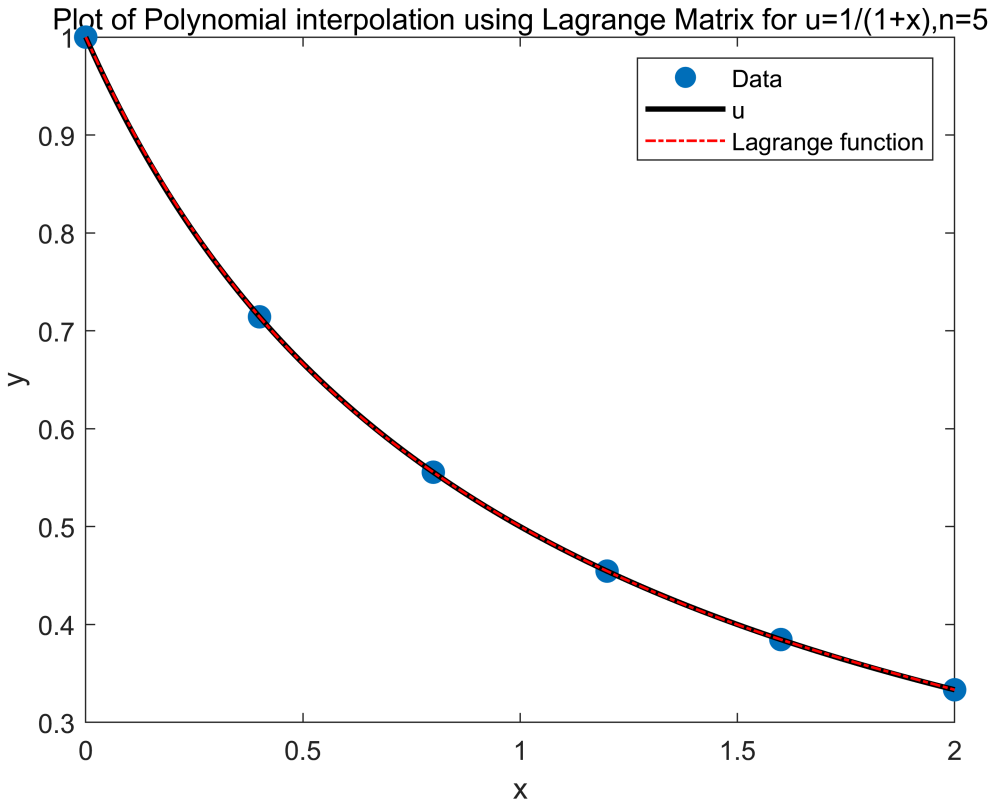

```
n = 10; % number of intervals
[x10V,y10V,xfine10V,u_Vander_fine10V,a_vec10V]=van(a,b,n);
plotA(x10V,y10V,xfine10V,u_Vander_fine10V,n)
```



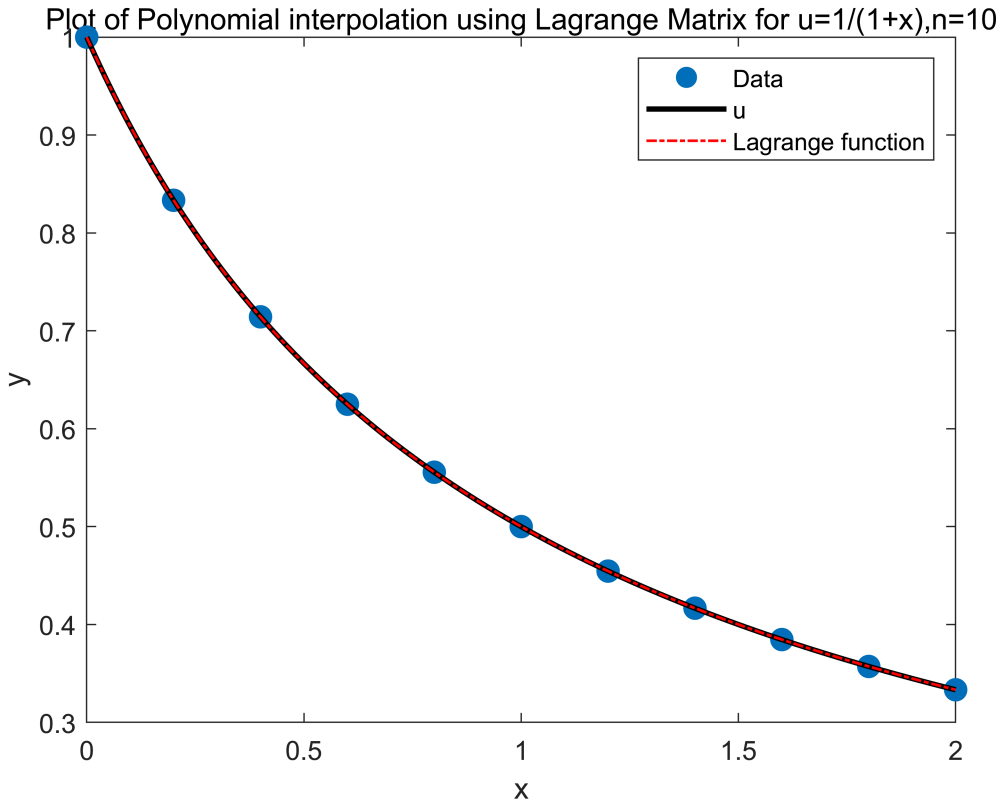

```
%Lagrange fomular
n = 2; % number of intervals
[x2L,y2L,xfine2L,u_Lagrange2L]=lan(a,b,n);
plotAN(x2L,y2L,xfine2L,u_Lagrange2L,n)
```




```
n = 5; % number of intervals
[x5L,y5L,xfine5L,u_Lagrange5L]=lan(a,b,n);
plotAN(x5L,y5L,xfine5L,u_Lagrange5L,n)
```




```
n = 10; % number of intervals
[x10L,y10L,xfine10L,u_Lagrange10L]=lan(a,b,n);
plotAN(x10L,y10L,xfine10L,u_Lagrange10L,n)
```



b)

With B-spline approach

In B-spline method we will use multiple B function

$$B(t)=\begin{cases}\frac{1}{6}t^3 & 0\le t\le 1 \quad (13)\\ \frac{1}{6}[-3(t-1)^3+3(t-1)^2+3(t-1)+1] & 1\le t\le 2 \quad (14)\\ \frac{1}{6}[3(t-2)^3-6(t-2)^2+4] & 2\le t\le 3 \quad (15)\\ \frac{1}{6}[-(t-3)^3+3(t-3)^2-3(t-3)+1] & 3\le t\le 4 \quad (16)\\ 0 & t\le 0 \text{ or } t\ge 4\end{cases}$$

For each B function $B_0, B_1, .. B_n$,

Each of B_j are shifted by h with other function

The B_0 ends at a+h

And B_n starts at b-h

B function has length=4

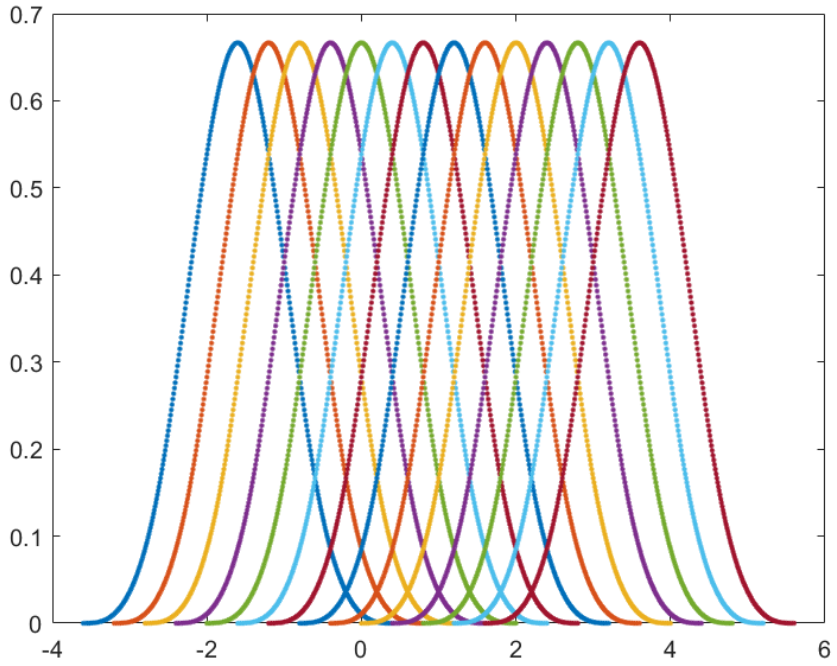
So B_0 starts at a+h-4

And B_n end at b-h+4

So if we graph it

For example n=5,a=0 and b=2

$$h = \frac{(b-a)}{n} = \frac{2}{5} = 0.4$$



So in total, the number of B funtion will be

$$n_b = \frac{(b-h)-(a+h-4)}{h} + 1 = \frac{2-0.4-0-0.4+4}{0.4} + 1 = \frac{5.2}{0.4} + 1 = 14$$

$$n_b = \frac{(b-h)-(a+h-4)}{h} + 1 \qquad (17)$$

The linear combination of B functions are:

$$s(x) = \sum_j^{n_b} \alpha_j B_j \qquad (18)$$

So for $j = 0, 1, 2, \ldots, n_b$

$$u(0+\mathrm{ih}) = \sum_j^{n_b} \alpha_j B_j(0+\mathrm{ih}) \qquad (19)$$

for i=0,1,2,...n+1

And we also have

$$u'(a) = \sum_j^{n_b} \alpha_j B'_j(a) \qquad (20)$$

and

$$u'(b) = \sum_j^{n_b} \alpha_j B'_j(b) \qquad (21)$$

we could compute the B'_j first

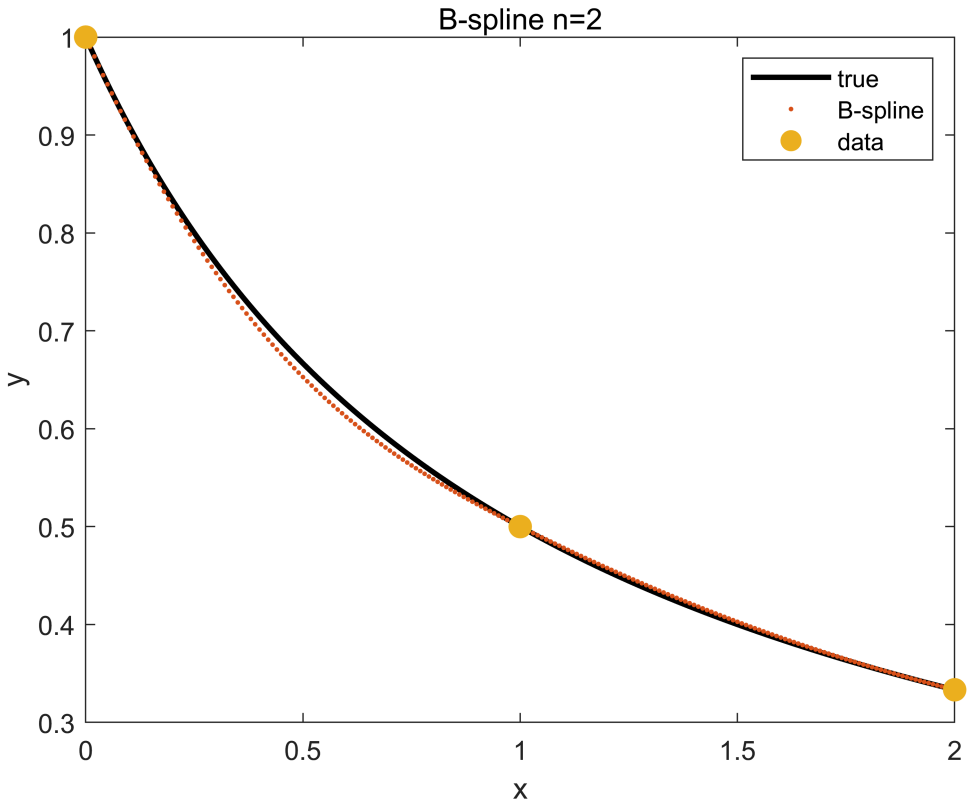
$$B'(t) = \begin{cases} \frac{1}{2}t^2 & 0 \le t \le 1 \quad (22) \\ \frac{1}{6}[-9(t-1)^2 + 6(t-1) + 3] & 1 \le t \le 2 \quad (23) \\ \frac{1}{6}[9(t-2)^2 - 12(t-2)] & 2 \le t \le 3 \quad (24) \\ \frac{1}{6}[-3(t-3)^2 + 6(t-3) - 3] & 3 \le t \le 4 \quad (25) \\ 0 & t \le 0 \text{ or } t \ge 4 \end{cases}$$

therefore we can write it as matrix

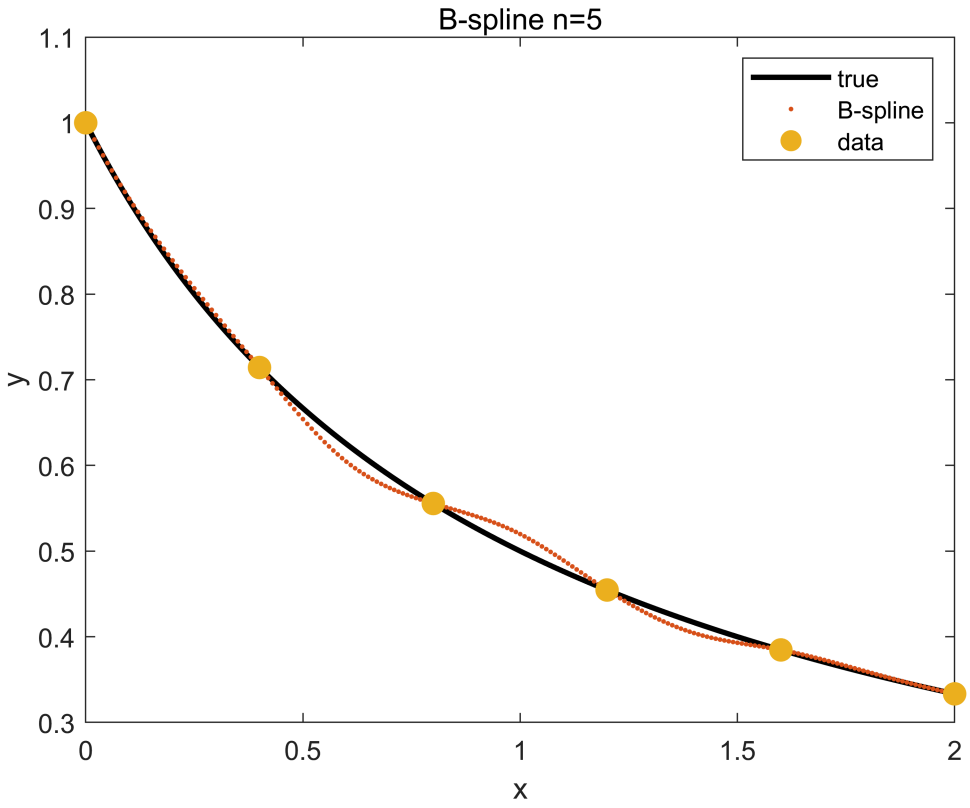
$$\begin{bmatrix} B_0(0+0\text{h}) & B_1(0+0h) & \cdots & B_{n_b}(0+0h) \\ B_0(0+1h) & B_1(0+1h) & \cdots & B_{n_b}(0+1h) \\ \vdots & \vdots & \ddots & \vdots \\ B_j(0+(n+1)h) & B_j(0+(n+1)h) & \cdots & B_j(0+(n+1)h) \\ B'_0(a) & B'_1(a) & \cdots & B'_{n_b}(a) \\ B'_0(b) & B'_1(b) & \cdots & B'_{n_b}(b) \end{bmatrix} \begin{bmatrix} \alpha_0 \\ \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_{n_b} \end{bmatrix} = \begin{bmatrix} u(0+0h) \\ u(0+1h) \\ \vdots \\ u(0+(n+1)h) \\ u'(a) \\ u'(b) \end{bmatrix}$$

Then we could compute *α* vector and approch the true value by using equation (18)

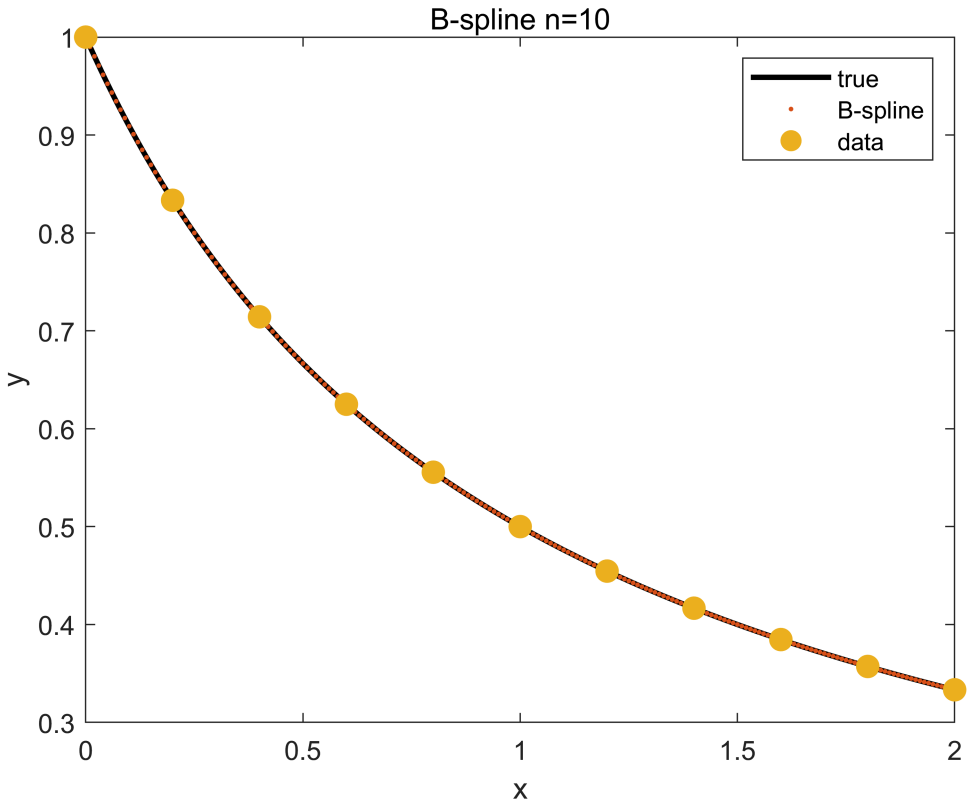
```
n=2;
[bc2,vv2,v2,t2]=b_clam(a,b,n);
[Y2B]=plotB(bc2,vv2,v2,x,ut,n,t2);
```




```
n=5;
[bc5,vv5,v5,t5]=b_clam(a,b,n);
[Y5B]=plotB(bc5,vv5,v5,x,ut,n,t5);
```




```
n=10;
[bc10,vv10,v10,t10]=b_clam(a,b,n);
[Y10B]=plotB(bc10,vv10,v10,x,ut,n,t10);
```



c)

Using clamped-spline

We will cut the function according to the numbers of data

So we define the funtion for n given data:

$$s = s(x) = \begin{cases} P_1(x) & x \in [x_0, x_1] \\ P_2(x) & x \in [x_1, x_2] \\ \vdots & \vdots \\ P_n(x) & x \in [x_{n-1}, x_n] \end{cases}$$

And each one could be written as

$$P_j(x) = a_j + b_j x + c_j x^2 + d_j x^3 \qquad (26)$$

Therefore we will get the equation below

$$\begin{aligned} &P_1(x_0) = u(x_0) \\ &P_2(x_1) = u(x_1) \\ &\vdots \\ &P_n(x_{n-1}) = u(x_{n-1}) \end{aligned} \qquad \text{(27a) and}$$

$$
\begin{aligned}
&P_1(x_1) = u(x_1)\\
&P_2(x_2) = u(x_2)\\
&\vdots\\
&P_n(x_n) = u(x_n)
\end{aligned}
\qquad \text{(27b)}
$$

Since we also need the first derivative to be equall

$$
\begin{aligned}
&P'_1(x_1) = P'_2(x_1)\\
&P_2'(x_2) = P_3'(x_2)\\
&\vdots\\
&P_{n-1}'(x_{n-1}) = P_n'(x_{n-1})
\end{aligned}
\qquad \text{(28)}
$$

Since we also need the second derivative to be equall

$$
\begin{aligned}
&P''_1(x_1) = P''_2(x_1)\\
&P_2''(x_2) = P_3''(x_2)\\
&\vdots\\
&P_{n-1}''(x_{n-1}) = P_n''(x_{n-1})
\end{aligned}
\qquad \text{(29)}
$$

So in total there are $4n$ of unknowns and only $n + n + n - 1 + n - 1 = 4n - 2$ equations

So we need clamped splines:

$$u'(a) = P_1'(a) = P_1'(x_0) \qquad \text{(30)}$$

$$u'(b) = P_n'(b) = P_n'(x_n) \qquad \text{(31)}$$

if we plug the (26) into (28) and (29)

$$P_j'(x) = 0 + b_j + 2c_j x + 3d_j x^2$$

$$P_j''(x) = 0 + 0 + 2c_j + 6d_j x$$

So for (28)

$$
\begin{aligned}
&P'_1(x_1) = P'_2(x_1) && 0 + b_1 + 2c_1x_1 + 3d_1x_1^2 = 0 + b_2 + 2c_2x_1 + 3d_2x_1^2\\
&P_2'(x_2) = P_3'(x_2) && 0 + b_2 + 2c_2x_2 + 3d_2x_2^2 = 0 + b_3 + 2c_3x_2 + 3d_3x_2^2\\
&\vdots && \vdots\\
&P_{n-1}'(x_{n-1}) = P_n'(x_{n-1}) \Rightarrow && 0 + b_{n-1} + 2c_{n-1}x_{n-1} + 3d_{n-1}x_{n-1}^2 = 0 + b_n + 2c_nx_{n-1} + 3d_nx_{n-1}^2
\end{aligned}
$$

$$
\begin{aligned}
&0 + b_1 + 2c_1x_1 + 3d_1x_1^2 - 0 - b_2 - 2c_2x_1 - 3d_2x_1^2 = 0\\
&0 + b_2 + 2c_2x_2 + 3d_2x_2^2 - 0 - b_3 - 2c_3x_2 - 3d_3x_2^2 = 0\\
&\vdots\\
\Rightarrow\ &0 + b_{n-1} + 2c_{n-1}x_{n-1} + 3d_{n-1}x_{n-1}^2 - 0 - b_n - 2c_nx_{n-1} - 3d_nx_{n-1}^2 = 0
\end{aligned}
\qquad \text{(32)}
$$

and (29)

$$
\begin{aligned}
&P''_1(x_1) = P''_2(x_1) && 0+0+2c_1+6d_1x_1 = 0+0+2c_2+6d_2x_1\\
&P_2''(x_2) = P_3''(x_2) && 0+0+2c_2+6d_2x_2 = 0+0+2c_3+6d_3x_2\\
&\vdots && \vdots\\
&P_{n-1}''(x_{n-1}) = P_n''(x_{n-1}) \Rightarrow && 0+0+2c_{n-1}+6d_{n-1}x_{n-1} = 0+0+2c_n+6d_nx_{n-1}
\end{aligned}
$$

$$
\begin{aligned}
&0+0+2c_1+6d_1x_1-0-0-2c_2-6d_2x_1=0\\
&0+0+2c_2+6d_2x_2=0-0-2c_3-6d_3x_2=0\\
&\vdots\\
\Rightarrow\ &0+0+2c_{n-1}+6d_{n-1}x_{n-1}-0-0-2c_n-6d_nx_{n-1}=0 \Rightarrow
\end{aligned}
\qquad (33)
$$

so if we write (27a) (27b) (32) (33) into a marix:

$$
\begin{aligned}
&P_1(x_0) = u(x_0)\\
&P_2(x_1) = u(x_1)\\
&\vdots\\
&P_n(x_{n-1}) = u(x_{n-1})
\end{aligned}
\qquad \text{(27a) and}
$$

$$
\begin{aligned}
&P_1(x_1) = u(x_1)\\
&P_2(x_2) = u(x_2)\\
&\vdots\\
&P_n(x_n) = u(x_n)
\end{aligned}
\qquad (27b)
$$

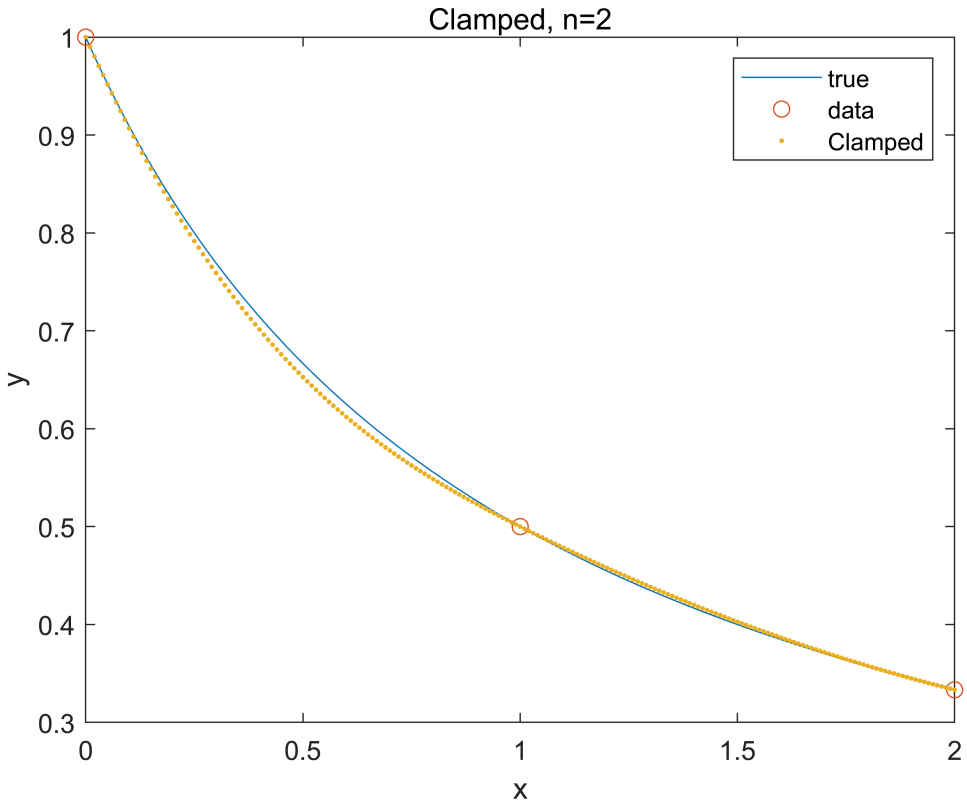
and (30) and (31)

$$V = \begin{bmatrix}
1 & x_0 & x_0^2 & x_0^3 & 0 & & & & & & & \cdots & & & & & & & 0 \\
0 & & & & 1 & x_1 & x_1^2 & x_1^3 & & & & \cdots & & & & & & & 0 \\
0 & & & & & & & & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & & & & 0 \\
0 & & & & & & & & & & & \cdots & & & & & 1 & x_{n-1} & x_{n-1}^2 & x_{n-1}^3 \\
1 & x_1 & x_1^2 & x_1^3 & & & & & & & & \cdots & & & & & & & & 0 \\
0 & & & & 1 & x_2 & x_2^2 & x_2^3 & & & & \cdots & & & & & & & & 0 \\
0 & & & & & & & & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & \cdots & & & & 0 \\
0 & & & & & & & & & & & \cdots & & & & & 1 & x_n & x_n^2 & x_n^3 \\
0 & 1 & 2x_1 & 3x_1^2 & 0 & -1 & -2x_1 & -3x_1^2 & & & & \cdots & & & & & & & & 0 \\
0 & & & & 0 & 1 & 2x_2 & 3x_2^2 & 0 & -1 & -2x_2 & -3x_2^2 & \cdots & & & & & & & 0 \\
0 & & & & & & & & & & & \cdots & & & & & & & & 0 \\
0 & & & & & & & & & & & \cdots & 0 & 1 & 2x_{n-1} & 3x_{n-1}^2 & 0 & -1 & -2x_{n-1} & -3x_{n-1}^2 \\
0 & 0 & 2 & 6x_1 & 0 & 0 & -2 & -6x_1 & & & & \cdots & & & & & & & & 0 \\
0 & & & & 0 & 0 & 2 & 6x_2 & 0 & 0 & -2 & -6x_2 & \cdots & & & & & & & 0 \\
0 & & & & & & & & & & & \cdots & & & & & & & & 0 \\
0 & & & & & & & & & & & \cdots & 0 & 0 & 2 & 6x_{n-1} & 0 & 0 & -2 & -6x_{n-1} \\
0 & 1 & 2x_0 & 3x_0^2 & & & & & & & & \cdots & & & & & & & & 0 \\
0 & & & & & & & & & & & \cdots & & & & & 0 & 1 & 2x_n & 3x_n^2
\end{bmatrix}$$

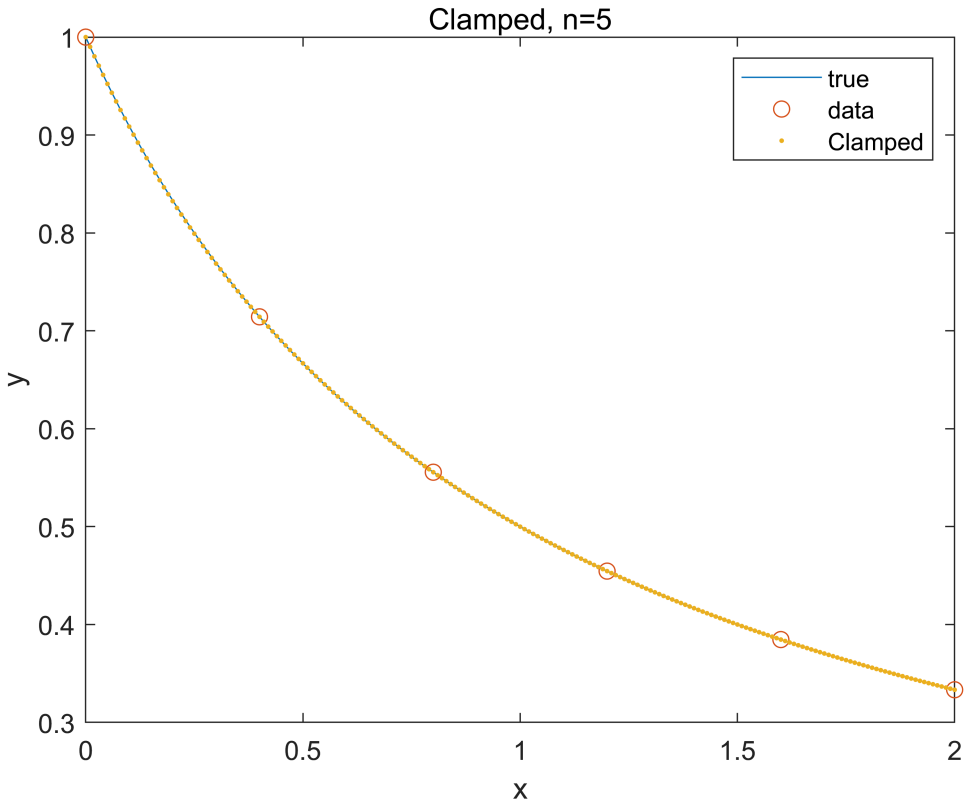
$$a = \begin{bmatrix} a_1 \\ b_1 \\ c_1 \\ d_1 \\ a_2 \\ b_2 \\ c_2 \\ d_2 \\ \vdots \\ a_n \\ b_n \\ c_n \\ d_n \end{bmatrix} \text{ and } u = \begin{bmatrix} u(x_0) \\ u(x_1) \\ \vdots \\ u(x_{n-1}) \\ u(x_1) \\ u(x_2) \\ \vdots \\ u(x_n) \\ 0 \\ \vdots \\ 0 \\ 0 \\ \vdots \\ 0 \\ u'(a) \\ u'(b) \end{bmatrix} \text{ ,where } \mathrm{Va} = u$$

Then we can compute a from V and u.

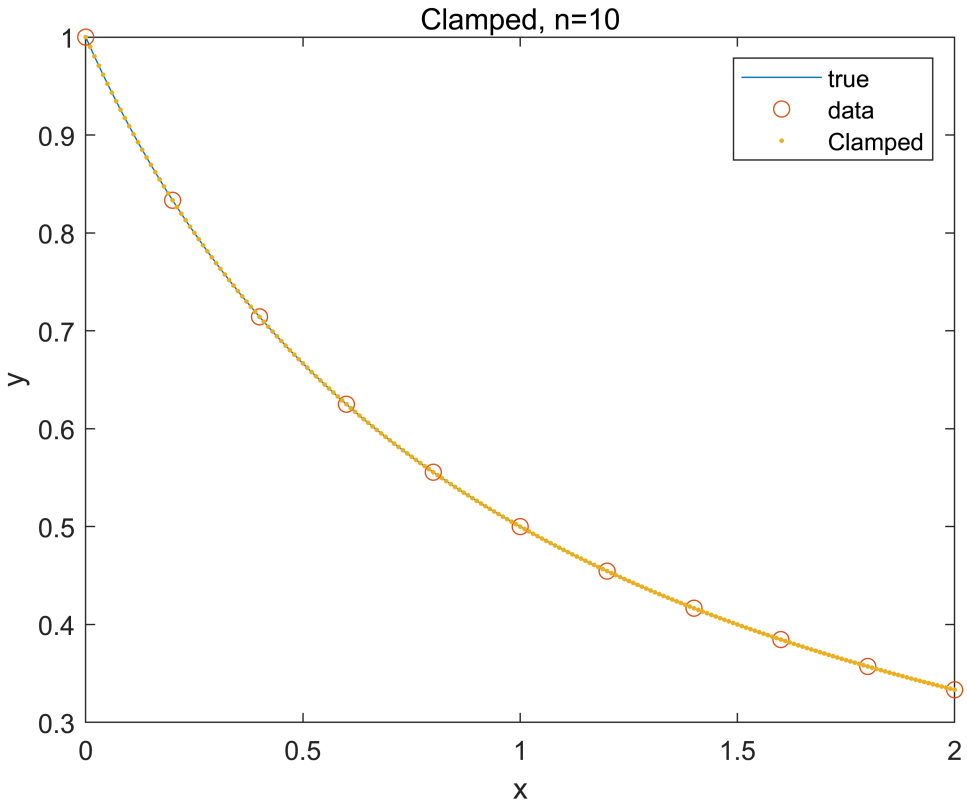
```
n=2;
[X_coeC2]=clam(a,b,n);
[Y2C]=plotCC(x,ut,a,b,X_coeC2,n);
```




```
n=5;
[X_coeC5]=clam(a,b,n);
[Y5C]=plotCC(x,ut,a,b,X_coeC5,n);
```




```
n=10;
[X_coeC10]=clam(a,b,n);
[Y10C]=plotCC(x,ut,a,b,X_coeC10,n);
```



from the plot we could conclude that , the more n we take the more closer two curves will be.

For natrual spline

instand (30) and (31) we will use

$P_1''(a)=P_1''(x_0)=0$ (34)

$P_n''(a)=P_n''(x_n)=0$ (35)

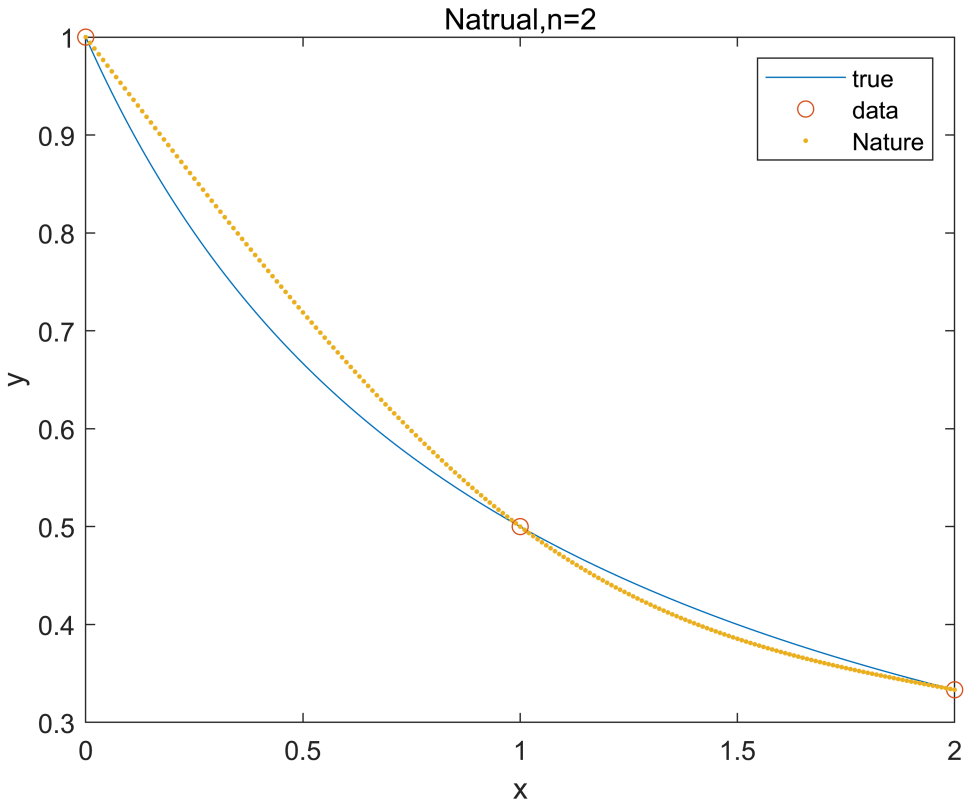
The only things that need to be changed is

last 2 row of the matrix V

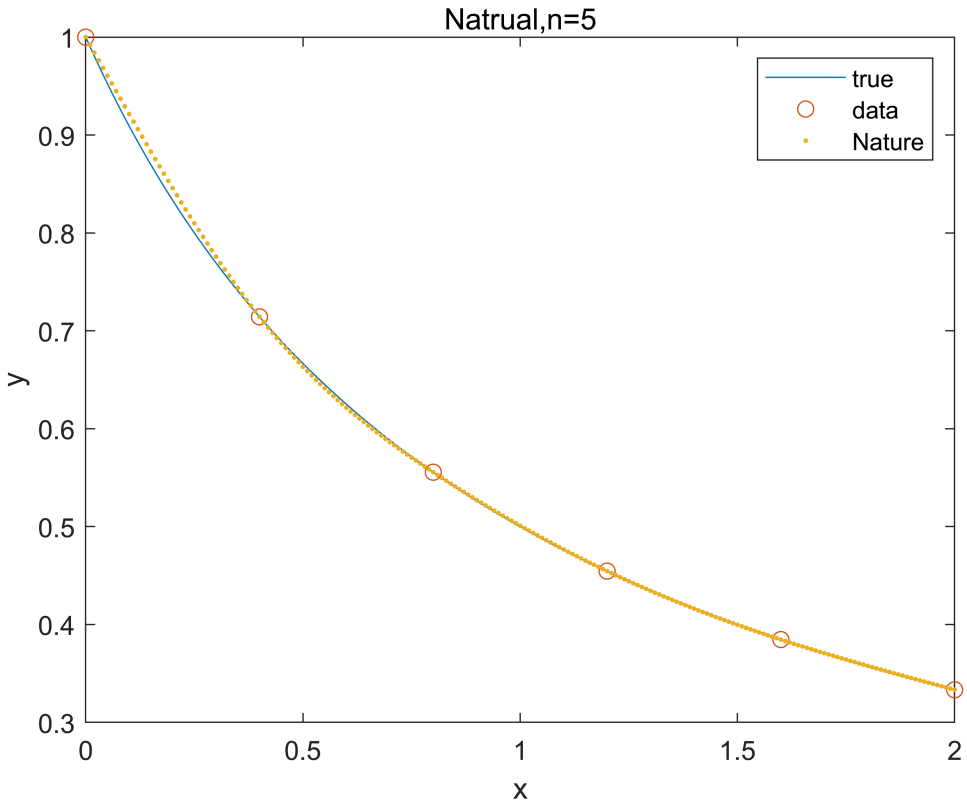
$$\begin{bmatrix} 0 & 0 & 2 & 6a & \cdots & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & \cdots & 0 & 0 & 2 & 6b \end{bmatrix} \quad (36)$$

and last 2 terms of u becomes 0.

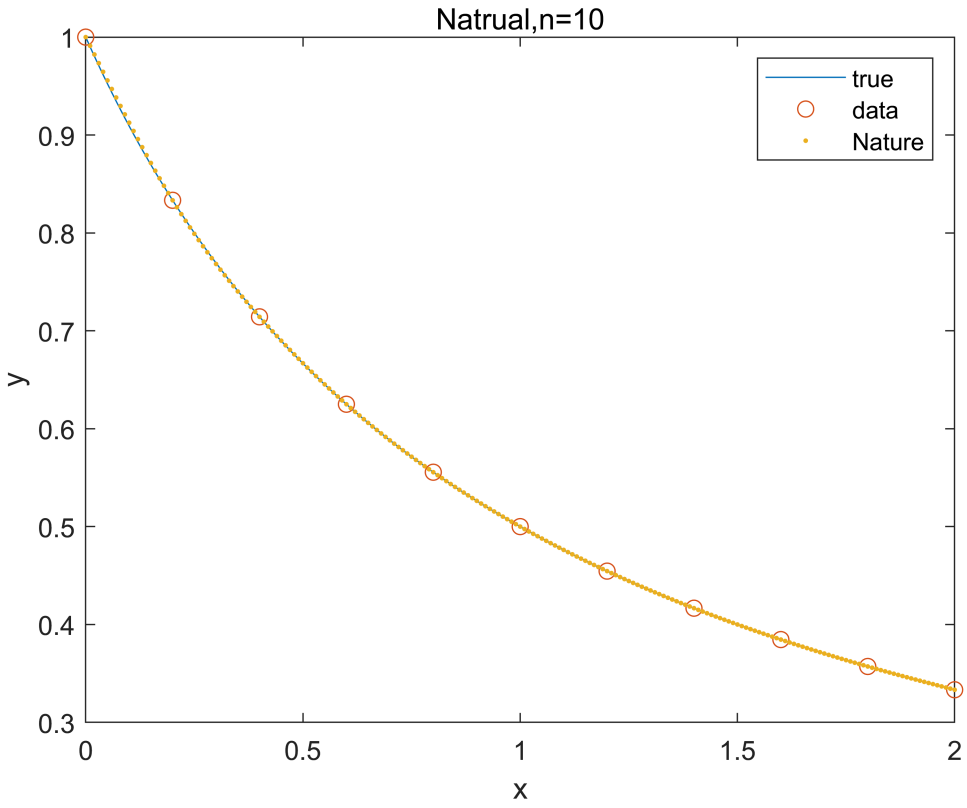
```
n=2;
[X_coeN2]=nat(a,b,n);
[Y2N]=plotCN(x,ut,a,b,X_coeN2,n);
```




```
n=5;
[X_coeN5]=nat(a,b,n);
[Y5N]=plotCN(x,ut,a,b,X_coeN5,n);
```




```
n=10;
[X_coeN10]=nat(a,b,n);
[Y10N]=plotCN(x,ut,a,b,X_coeN10,n);
```



we could find that when n is small the nature spline work bad compare with the clamped.

To specific compute

```
err_Vandermonde_n_2 = sum(abs(u_Vander_fine2V-u(xfine2V)),'all')
```

```
err_Vandermonde_n_2 = 4.5646
```

```
err_Vandermonde_n_5 = sum(abs(u_Vander_fine5V-u(xfine5V)),'all')
```

```
err_Vandermonde_n_5 = 0.0643
```

```
err_Vandermonde_n_10 = sum(abs(u_Vander_fine10V-u(xfine10V)),'all')
```

```
err_Vandermonde_n_10 = 1.3101e-04
```

```
err_Lagrange_n_2 = sum(abs(u_Lagrange2L-u(xfine2L)),'all')
```

```
err_Lagrange_n_2 = 4.5646
```

```
err_Lagrange_n_5 = sum(abs(u_Lagrange5L-u(xfine5L)),'all')
```

```
err_Lagrange_n_5 = 0.0643
```

```
err_Lagrange_n_10 = sum(abs(u_Lagrange10L-u(xfine10L)),'all')
```

```
err_Lagrange_n_10 = 1.3101e-04
```

```
err_B_spline_2=sum(abs(Y2B(:)-ut(:)),"all")
```

```
err_B_spline_2 = 0.9904
```

```
err_B_spline_5=sum(abs(Y5B(:)-ut(:)),"all")
```

```
err_B_spline_5 = 1.5040
```

```
err_B_spline_10=sum(abs(Y10B(:)-ut(:)),"all")
```

```
err_B_spline_10 = 0.0013
```

```
err_Cliamped_2=sum(abs(Y2C(:)-ut(:)),'all')
```

```
err_Cliamped_2 = 0.9904
```

```
err_Cliamped_5=sum(abs(Y5C(:)-ut(:)),'all')
```

```
err_Cliamped_5 = 0.0247
```

```
err_Cliamped_10=sum(abs(Y10C(:)-ut(:)),'all')
```

```
err_Cliamped_10 = 0.0013
```

```
err_Nature_2=sum(abs(Y2N(:)-ut(:)),'all')
```

```
err_Nature_2 = 4.5646
```

```
err_Nature_5=sum(abs(Y5N(:)-ut(:)),'all')
```

```
err_Nature_5 = 0.4788
```

```
err_Nature_10=sum(abs(Y10N(:)-ut(:)),'all')
```

```
err_Nature_10 = 0.0660
```

So as result, when n is high, vanermode and lagrange will has better result,

and when n is smal, B-spline and clamped will have better result.

```
function [X_coe]=clam(a,b,n)
%function of Clamped spline
%out put the coefficient a,b,c,d
    A=zeros(4*n,4*n);%initial Matrix V
    xx=linspace(a,b,n+1);%general grid
    x=xx(2:end);
    for i=1:n-1
        %general row 1 to n-1 of equation (27a) (32) (33)
        %general row 2 to n of equaiton (27b)
        A(i,(1+(i-1)*4:(i+1)*4))=[1 x(i) x(i)^2 x(i)^3 0 0 0 0];%(27a)
        A(i+n,(1+(i-1)*4:(i+1)*4))=[0 0 0 0 1 x(i) x(i)^2 x(i)^3];%(27b)
```

```
        A(i+n+n-1,(1+(i-1)*4:(i+1)*4))=[0 1 2*x(i) 3*x(i)^2 0 -1 -2*x(i) -3*x(i)^2];%(32)
        A(i+n+n-1+n-1,(1+(i-1)*4:(i+1)*4))=[0 0 2 6*x(i) 0 0 -2 -6*x(i)];%(33)
    end
    A(n,(1+(n-1)*4:n*4))=[1 x(n) x(n)^2 x(n)^3];%the n th row of (27a)
    A(n+n-1+n-1+n-1+1,1)=1;%the first row of (27b)
    A(n+n-1+n-1+n-1+1+1,1:4)=[0 1 2*a 3*a^2];%(30)
    A(n+n-1+n-1+n-1+1+1+1,(4*(n-1)+1:4*(n-1)+4))=[0 1 2*b 3*b^2];%(31)
    rhs=zeros(1,4*n);%initial the u vector
    s=u(x);
    rhs(1:n)=s;%(27a)
    rhs(n+1:n+n-1)=s(1:n-1);%(27b)
    rhs(n+n-1+n-1+n-1+1)=u(a);
    rhs(4*n-1)=uprim(a);%(30)
    rhs(4*n)=uprim(b);%(31)
    X_coe=A\rhs';
end

function [X_coe]=nat(a,b,n)
%function of Clamped spline
%out put the coefficient a,b,c,d
    A=zeros(4*n,4*n);
    xx=linspace(a,b,n+1);
    x=[xx(2:end)];
    for i=1:n-1
        A(i,(1+(i-1)*4:(i+1)*4))=[1 x(i) x(i)^2 x(i)^3 0 0 0 0];
        A(i+n,(1+(i-1)*4:(i+1)*4))=[0 0 0 0 1 x(i) x(i)^2 x(i)^3];
        A(i+n+n-1,(1+(i-1)*4:(i+1)*4))=[0 1 2*x(i) 3*x(i)^2 0 -1 -2*x(i) -3*x(i)^2];
        A(i+n+n-1+n-1,(1+(i-1)*4:(i+1)*4))=[0 0 2 6*x(i) 0 0 -2 -6*x(i)];
    end
    A(n,(1+(n-1)*4:n*4))=[1 x(n) x(n)^2 x(n)^3];
    A(n+n-1+n-1+n-1+1,1)=1;
    A(n+n-1+n-1+n-1+1+1,1:4)=[0 0 2 6*a];%(34)
    A(n+n-1+n-1+n-1+1+1+1,(4*(n-1)+1:4*(n-1)+4))=[0 0 2 6*b];%(35)
    rhs=zeros(1,4*n);
    s=u(x);
    rhs(1:n)=s;
    rhs(n+1:n+n-1)=s(1:n-1);
    rhs(n+n-1+n-1+n-1+1)=u(a);
    X_coe=A\rhs';
end

function [X,Y]=hua(a,b,X_coe,n)
%funtion of processing the data
%input:
%X_coe: the coefficient computed from above
%ouutput:
%X,Y: x coordinate and Y coordinate of the approximate funtion.
    h=(b-a)/n;
    dx=0.01;
    Y=[];
    X=[];
    for i=1:n
        %loop of n
        x=a+(i-1)*h:dx:a+i*h-dx;%compute each part of x coodinate
```

```
        %compute each part of y coodinate according to the coefficient
        y=X_coe(1+(i-1)*4)+ ...
        X_coe(2+(i-1)*4)*x+ ...
        X_coe(3+(i-1)*4)*x.^2+ ...
        X_coe(4+(i-1)*4)*x.^3;
        Y=[Y y];
        X=[X x];
    end
    %last point of Y
    Y=[Y X_coe(4*n-3)+X_coe(4*n-2)*b+X_coe(4*n-1)*b^2+X_coe(4*n)*b^3];
    %last point of X
    X=[X b];
end

function [datax,datay]=dian(a,b,n)
%funtion of compute the data
    datax=linspace(a,b,n+1);
    datay=u(datax);
end

function [ut]=u(x)
%function real function
    ut=1./(x+1);
end

function [up]=uprim(x)
%function of uprime
    up=-(x+1).^(-2);
end

function [bc,vv,v,t]=b_clam(a,b,n)
%function of forming the matrix, u vector and compute the alpha vector
%output
%bc:alpha vector
%vv:index of B function
%v:number of B funciton
    h=(b-a)/n;
    vv=(-4+h:h:2-h);%compute the index of B function equation(17)
    v=length(vv);%compute the number of B function
    Ab=zeros(n+3);%initial the matrix
    t=a:h:b;%define t
    for i=1:n+1
        %loop of i
        for j=1:v
            %loop of j
            Ab(i,j)=BBj(t(i),vv(j));%fill the matrix with equation(19)
        end
    end
    for j=1:v
        %fill the matrix with equation (20) and (21)
        Ab(n+2,j)=BBjprime(a,vv(j));%euqaiton(20)
        Ab(n+3,j)=BBjprime(b,vv(j));%euqaiton(21)
    end
    %general the u vector
```

```
    yb=zeros(n+3,1);
    yb(1:n+1)=u(t);%fill the first n+1 terms with u(t)
    yb(n+2)=uprim(a);%euqaiton(20)
    yb(n+3)=uprim(b);%euqaiton(21)
    bc=Ab\yb;%compute the alpha vector
end

function [Bt1]=B1(t)
%equation (13)
    Bt1=1/6*t.^3;
end

function [Bt2]=B2(t)
%equation (14)
    Bt2=1/6*(-3*(t-1).^3+3*(t-1).^2+3*(t-1)+1);
end

function [Bt3]=B3(t)
%equation (15)
    Bt3=1/6*(3*(t-2).^3-6*(t-2).^2+4);
end

function [Bt4]=B4(t)
%equation (16)
    Bt4=1/6*(-(t-3).^3+3*(t-3).^2-3*(t-3)+1);
end

function [Bt1p]=B1prime(t)
%equation (22)
    Bt1p=1/2*t.^2;
end

function [Bt2p]=B2prime(t)
%equation (23)
    Bt2p=1/6*(-9*(t-1).^2+6*(t-1)+3);
end

function [Bt3p]=B3prime(t)
%equation (24)
    Bt3p=1/6*(9*(t-2).^2-12*(t-2));
end

function [Bt4p]=B4prime(t)
%equation (25)
    Bt4p=1/6*(-3*(t-3).^2+6*(t-3)-3);
end

function [k]=BBj(t,j)
%function of combina B1, B2, B3, B4 together
%j means the shifts take placed on B
    if j<=t && t<=j+1
        k=B1(t-j);
    elseif j+1<=t && t<=j+2
        k=B2(t-j);
```

```
    elseif j+2<=t && t<=j+3
        k=B3(t-j);
    elseif j+3<=t && t<=j+4
        k=B4(t-j);
    elseif t>=j+4 || t<=j
        k=0;
    end
end

function [k]=BBjprime(t,j)
%function of combina B1prime, B2prime, B3prime, B4prime together
%j means the shifts take placed on Bprime
    if j<=t && t<=j+1
        k=B1prime(t-j);
    elseif j+1<=t && t<=j+2
        k=B2prime(t-j);
    elseif j+2<=t && t<=j+3
        k=B3prime(t-j);
    elseif j+3<=t && t<=j+4
        k=B4prime(t-j);
    elseif t>=j+4 || t<=j
        k=0;
    end
end

function [Y]=plotCC(x,ut,a,b,X_coe,n)
%funtion of plot clamped spline
%out put the approximate y value
    [X,Y]=hua(a,b,X_coe,n);%compute the y coordinate
    [datax,datay]=dian(a,b,n);%input the data
    plot(x,ut)%real value
    hold on
    plot(datax,datay,'o')%data
    plot(X,Y,'.')%Clamped
    xlabel('x');ylabel('y')
    legend('true','data','Clamped')
    title(['Clamped, n=',num2str(n)])
    hold off
end

function [Y]=plotCN(x,ut,a,b,X_coe,n)
%funtion of plot nature spline
%out put the approximate y value
    [X,Y]=hua(a,b,X_coe,n);
    [datax,datay]=dian(a,b,n);
    plot(x,ut)
    hold on
    plot(datax,datay,'o')
    plot(X,Y,'.')
    xlabel('x');ylabel('y')
    legend('true','data','Nature')
    title(['Natrual,n=',num2str(n)])
    hold off
end
```

```
function [Y]=plotB(bc,vv,v,x,ut,n,t)
%funtion of ploting the B-spline
%input:
%bc:alpha vector
%vv:index of B function
%v:number of B funciton
%x:grid
%ut:real value
%yb:u vector
%n
    num=length(x);
    Y=zeros(num,1);
    for i=1:v
        for j=1:num
            %compute the linear compbination from equation (18)
            Y(j)=Y(j)+bc(i)*BBj(x(j),vv(i));
        end
    end
    plot(x,ut,'k', 'LineWidth',2)%true value
    hold on
    plot(x,Y,'.')%approximate value
    plot(t,u(t),'.','MarkerSize', 30)%data
    xlabel('x');ylabel('y')
    legend('true','B-spline','data')
    title(['B-spline n=',num2str(n)])
    hold off
end

function [x,y,xfine,u_Vander_fine,a_vec]=van(a,b,n)
%function of computing the Vandermonde martix and correspounding
%coefficients
%take input a,b,and n
%out put x: the x coodinate of data point according to n
%out put y: the y coodinate of data point that compute from x
%xfine: a finer grid of x
%u_Vander_fine: approximate the polynomial using the coefficient compute from V and y
%a_vec: vector a
    h = (b-a)/n;
    x = a + (0:n)*h; % define the x values
    y = u(x); % define the corresponding y values as y_j = u(x_j)
    Vmat = zeros(length(x));
    for ii=1:length(x)
        % loop over the data points
        for jj=1:length(x)
            % loop over the columns of the Vandermonde martix
            Vmat(ii,jj) = x(ii)^(jj-1);
            % since matlab indices starts at 1, so we need to subtract 1
        end
    end
    rr_form     = rref([Vmat,y(:)]);
    a_vec       = rr_form(:,end); % peeling off the last column
    xfine         = a:0.01:b;%define a finer grid x
    u_Vander_fine = a_vec(1)*ones(size(xfine));
```

```
    for jj=1:n
        %compute the polynomial from vector a
        u_Vander_fine = u_Vander_fine + a_vec(jj+1)*xfine.^(jj);
    end
end

function [x,y,xfine,u_Lagrange]=lan(a,b,n)
%function of compute the Lagrange fomular
%input a, b and n
%outpout
%out put x: the x coodinate of data point according to n
%out put y: the y coodinate of data point that compute from x
%xfine: a finer grid of x
%u_Lagrange:  approximate the polynomial
% using the coefficient compute Lagrange fomular
    h = (b-a)/n;
    x = a + (0:n)*h; % define the x values
    y = u(x); % define the corresponding y values as y_j = u(x_j)
    N          = length(x);
    xfine       = a:0.01:b; % define the finer x values
    u_Lagrange  = zeros(size(xfine));
    for k = 1:N
        %loop each k from equation (6)
        w = ones(size(xfine));
        for j = [1:k-1 k+1:N] % This loop the index from equation (8)
            w = (xfine-x(j))./(x(k)-x(j)).*w;
        end
        u_Lagrange = u_Lagrange + w*y(k);
        % This adds up each y_k*l_k(x) term, equation (7)
    end
end

function []=plotA(x,y,xfine,u_Vander_fine,n)
%function of plot the polynomial from Vandermonde martix
%take input:
%x: the x coodinate of data point according to n
%y: the y coodinate of data point that compute from x
%xfine: a finer grid of x
%u_Vander_fine: approximate the polynomial using the coefficient compute from V and y
%n
    plot(x,y,'.','MarkerSize', 30)%plot the data
    hold on;
    plot(xfine,u(xfine),'k', 'LineWidth',2)%plot the true value
    plot(xfine, u_Vander_fine, 'r')%plot the polynomial
    xlabel('x');ylabel('y');
    legend('Data','u', 'Vander poly')
    title(['Plot of Polynomial interpolation using ' ...
        'Vandermonde Matrix for u=1/(1+x),n=',num2str(n)])
    hold off
end

function []=plotAN(x,y,xfine,u_Lagrange,n)
%function of plot the polynomial from Lagrange fomular
%take input:
```

```
%x: the x coodinate of data point according to n
%y: the y coodinate of data point that compute from x
%xfine: a finer grid of x
%u_Lagrange: approximate the polynomial using the coefficient compute Lagrange
%n
    plot(x,y,'.','MarkerSize',30);%plot the data
    hold on;
    plot(xfine,u(xfine),'k', 'LineWidth',2);%plot the true value
    plot(xfine,u_Lagrange,'-.r','LineWidth',1)%plot the polynomial
    xlabel('x');ylabel('y');
    legend('Data','u','Lagrange function')
    title(['Plot of Polynomial interpolation using' ...
        ' Lagrange Matrix for u=1/(1+x),n=',num2str(n)])
    hold off
end
```