附录:

functions

Lagrange

```
function y = fun_lagrangeInter(X, Y, x)
% Lagrange Interpolation
n = length(X);
up = 0;
low = 0;
for i = 1:n
  w = 1;
  for j = 1:n
    if j ~= i
      w = w/(X(i)-X(j));
    end
  end
  up = up + w*Y(i)./(x-X(i));
  low = low + w./(x-X(i));
end
y = up./low;
end
```

thomas

```
function x = thomas(varargin)
% THOMAS    Solves a tridiagonal linear system
%   x = THOMAS(a,b,c,d) where a is the diagonal, b is the upper diagonal, and c is
%       the lower diagonal of A also solves A*x = d for x. Note that a is size n
%       while b and c is size n-1.
[a,b,c,d] = parse_inputs(varargin{:});
m = zeros(size(a));
l = zeros(size(c));
y = zeros(size(d));
n = size(a,1);
m(1,:) = a(1,:);
y(1,:) = d(1,:);
for i = 2 : n
   i_1 = i-1;
   l(i_1,:) = c(i_1,:)./m(i_1,:);
   m(i,:) = a(i,:) - l(i_1,:).*b(i_1,:);

   y(i,:) = d(i,:) - l(i_1,:).*y(i_1,:);

end
x(n,:) = y(n,:)./m(n,:);
for i = n-1 : -1 : 1
   x(i,:) = (y(i,:) - b(i,:).*x(i+1,:))./m(i,:);
end
end
```

thomas2

```
function x = periodic(b,c,a,d)
% b = diag
% c = upper + A(end,1)
% a = lower + A(1,end)
% d = b
n = length(a);
x = zeros(n,1);
q = zeros(1,n+1);
p = q;
u = q;
for k = 2 : n+1
    p(k) = a(k-1)*q(k-1)+b(k-1);
    q(k) = -c(k-1)/p(k);
    u(k) = (d(k-1)-a(k-1)*u(k-1))/p(k);
end
t = zeros(1,n+1);
s = t;
v = t;
s(1) = 1;
t(n+1) = 1;
for k = 2 : n+1
    s(k) = -(a(k-1)*s(k-1))/p(k);
end
for k = n : -1 : 2
    t(k) = q(k)*t(k+1) + s(k);
    v(k) = q(k)*v(k+1) + u(k);
end
x(n) =(d(n)-c(n)*v(2)-a(n)*v(n))/(c(n)*t(2)+a(n)*t(n)+b(n));

for k = 1 : n-1
    x(k) = t(k+1)*x(n) + v(k+1);
end
```

Cubic Spline with Simply Supported Condition

```
function [ynew, polycoef] = cubicSplineSimplySupported(x, y, xnew, M0, Mn)
% Cubic Spline of 2D-points
% Simply-supported Boundary Condition
%

if nargin == 3
  M0 = 0;
  Mn = 0;
end
% formulate input data
if size(x,1)<size(x,2), x = x'; end
if size(y,1)<size(y,2), y = y'; end
if size(xnew,1)<size(xnew,2), xnew = xnew'; end

% linear equation
% tridiagonal matrix
n = length(x)-1;
x0 = x(1);
```

```
x = x(2:end);
% d -- diagonal
d = zeros(n-1,1);
d(1) = x(2)-x0;
d(2:end) = x(3:end)-x(1:end-2);
d = 2*d;
% u -- upper diagonal
u = x(2:end-1)-x(1:end-2);
% l -- lower diagonal -- l == u

% right hand items
y0 = y(1);
y = y(2:end);
temp = (y(2:end)-y(1:end-1))./(x(2:end)-x(1:end-1));
temp = [(y(1)-y0)/(x(1)-x0); temp];

rh = 6*(temp(2:end)-temp(1:end-1));
rh(1) = rh(1)-(x(1)-x0)*M0;
rh(end) = rh(end)-(x(end)-x(end-1))*Mn;
% Solve
M = thomas(d,u,u,rh);
% Interpolation polynomial
poly = zeros(n,4);
M = [M; Mn];
poly(1,1) = (1/6)*(M(1)-M0)./(x(1)-x0);
poly(2:end,1) = (1/6)*(M(2:end)-M(1:end-1))./(x(2:end)-x(1:end-1));
poly(:,2) = (1/2)*M;
poly(1,3) = (y(1)-y0)/(x(1)-x0)+(x(1)-x0)*(2*M(1)+M0)/6;
poly(2:end,3) = (y(2:end)-y(1:end-1))./(x(2:end)-...
  x(1:end-1))+(x(2:end)-x(1:end-1)).*(2*M(2:end)+M(1:end-1))/6;
poly(:,4) = y;
polycoef = poly;
% get new value
l = length(xnew);
ynew = zeros(l,1);
for k = 1:l
  for i = 1:n
    if xnew(k) < x(i)
      ynew(k) = polyval(poly(i,:), (xnew(k)-x(i)));
      break
    end
  end
end
```

Cubic Spline with Clamped condition

```
function [ynew, polycoef] = cubicSplineClamped(x, y, xnew, m0, mn)
% Cubic Spline of 2D-points
% Clamped Boundary Condition
%

if nargin == 3
  m0 = 0;
  mn = 0;
end
% formulate input data
if size(x,1)<size(x,2), x = x'; end
```

```
if size(y,1)<size(y,2), y = y'; end
if size(xnew,1)<size(xnew,2), xnew = xnew'; end

% linear equation

% tridiagonal matrix
n = length(x)-1;
x0 = x(1);
x = x(2:end);
% d -- diagonal
d = zeros(n-1,1);
d(1) = x(2)-x0;
d(2:end) = x(3:end)-x(1:end-2);
d = 2*d;
% u -- upper diagonal
u = zeros(n-2,1);
u(1) = x(1)-x0;
u(2:end) = x(2:end-2)-x(1:end-3);
% l -- lower diagonal
l = x(3:end)-x(2:end-1);

% right hand items
y0 = y(1);
y = y(2:end);
temp = (y(2:end)-y(1:end-1))./(x(2:end)-x(1:end-1));

rh = zeros(n-1,1);
rh(2:end) = 3*((x(2:end-1)-x(1:end-2)).*temp(2:end)+(x(3:end)-x(2:end-1)).*temp(1:end-1));
rh(1) = 3*(x(1)-x0)*(y(2)-y(1))/(x(2)-x(1))+3*(x(2)-x(1))*(y(1)-y0)/(x(1)-x0)-(x(2)-x(1))*m0;
rh(end) = rh(end)-(x(end-1)-x(end-2))*mn;

% Solve

m = thomas(d,u,l,rh);

% Interpolation polynomial
poly = zeros(n,4);
m = [m; mn];
poly(1,1) = (m0+m(1))/(x(1)-x0)^2+2*(y0-y(1))/(x(1)-x0)^3;
poly(2:end,1) = (m(1:end-1)+m(2:end))./((x(2:end)-x(1:end-1)).^2)...
  +2*(y(1:end-1)-y(2:end))./((x(2:end)-x(1:end-1)).^3);
poly(1,2) = 3*(y0-y(1))/(x(1)-x0)^2+(m0+2*m(1))/(x(1)-x0);
poly(2:end,2) = 3*(y(1:end-1)-y(2:end))./(x(2:end)-x(1:end-1)).^2+(m(1:end-1)+2*m(2:end))./(x(2:end)-x(1
poly(:,3) = m;
poly(:,4) = y;

polycoef = poly;
% get new value
l = length(xnew);
ynew = zeros(l,1);
for k = 1:l
  for i = 1:n
    if xnew(k) < x(i)
      ynew(k) = polyval(poly(i,:), (xnew(k)-x(i)));
      break
    end
  end
```

```
end
```

Cubic Spline Period Condition

```
function [ynew, polycoef] = cubicSplinePeriod(x, y, xnew)
% Cubic Spline of 2D-points
% Periodic Boundary Condition
%

% formulate input data
if size(x,1)<size(x,2), x = x'; end
if size(y,1)<size(y,2), y = y'; end
if size(xnew,1)<size(xnew,2), xnew = xnew'; end

% linear equation

% tridiagonal matrix
n = length(x)-1;
x0 = x(1);
x = x(2:end);
% d -- diagonal
d = zeros(n-1,1);
d(1) = x(2)-x0;
d(2:end) = x(3:end)-x(1:end-2);
d = [d; x(n)-x(n-1)+x(1)-x0];
d = 2*d;
% u -- upper diagonal
u = x(2:end-1)-x(1:end-2);
u = [u; x(end)-x(end-1)];
% l -- lower diagonal -- l == u

% right hand items
y0 = y(1);
y = y(2:end);
temp = (y(2:end)-y(1:end-1))./(x(2:end)-x(1:end-1));
temp = [(y(1)-y0)/(x(1)-x0); temp];

rh = 6*(temp(2:end)-temp(1:end-1));
rh = [rh; (y(1)-y0)/(x(1)-x0)-(y(end)-y(end-1))/(x(end)-x(end-1))];

% Solve
% A = diag(d)+diag(u,1)+diag(u,-1);
% A(end,1) = x(1)-x0;
% A(1,end) = x(1)-x0;
% b = rh;
% M = A\b;

M = periodic(d,[u;x(1)-x0],[u;x(1)-x0],rh);

% Interpolation polynomial

poly = zeros(n,4);
poly(1,1) = (1/6)*(M(1)-M(end))./(x(1)-x0);
poly(2:end,1) = (1/6)*(M(2:end)-M(1:end-1))./(x(2:end)-x(1:end-1));
poly(:,2) = (1/2)*M;
poly(1,3) = (y(1)-y0)/(x(1)-x0)+(x(1)-x0)*(2*M(1)+M(end))/6;
poly(2:end,3) = (y(2:end)-y(1:end-1))./(x(2:end)-...
```

```
  x(1:end-1))+(x(2:end)-x(1:end-1)).*(2*M(2:end)+M(1:end-1))/6;
poly(:,4) = y;
polycoef = poly;
% get new value

l = length(xnew);
ynew = zeros(l,1);
for k = 1:l
  for i = 1:n
    if xnew(k) < x(i)
      ynew(k) = polyval(poly(i,:), (xnew(k)-x(i)));
      break
    end
  end
end
```

Numerical Experiments

test01 ----- Equidistant Runge --------

```
f = @(x) 1./(1+12*x.^2);
xx = linspace(-1,1,100);
yy = f(xx);
figure(1)
plot(xx,yy)
axis([-1.1 1.1 -2 2])
grid on
% points set01: 5
x1 = linspace(-1,1,5);
y1 = f(x1);
yy1 = fun_lagrangeInter(x1, y1, xx);
figure(2)
plot(xx,yy,'b:','LineWidth',2)
axis([-1.1 1.1 -2 2])
hold on
plot(x1,y1,'ko','LineWidth',1)
plot(xx,yy1,'r','LineWidth',1)
grid on
% points set02: 15
x2 = linspace(-1,1,15);
y2 = f(x2);
yy2 = fun_lagrangeInter(x2, y2, xx);
figure(3)
plot(xx,yy,'b:','LineWidth',2)
axis([-1.1 1.1 -2 2])
hold on
plot(x2,y2,'ko','LineWidth',1)
plot(xx,yy2,'r','LineWidth',1)
grid on
% points set03: 25
x3 = linspace(-1,1,25);
y3 = f(x3);
yy3 = fun_lagrangeInter(x3, y3, xx);
figure(4)
plot(xx,yy,'b:','LineWidth',2)
```

```
axis([-1.1 1.1 -2 2])
hold on
plot(x3,y3,'ko','LineWidth',1)
plot(xx,yy3,'r','LineWidth',1)
grid on
```

test02 ----- chebyshev&cubic Runge ------

```
f = @(x) 1./(1+12*x.^2);
xx = linspace(-1,1,100);
yy = f(xx);
figure(1)
plot(xx,yy)
axis([-1.1 1.1 -2 2])
grid on

% points set03: 25
% chebyshev
xtemp = linspace(0,pi,25)';
x1 = cos(xtemp);
y1 = f(x1);
yy1 = fun_lagrangeInter(x1, y1, xx);
figure(2)
plot(xx,yy,'b:','LineWidth',2)
axis([-1.1 1.1 -2 2])
hold on
plot(x1,y1,'ko','LineWidth',1)
plot(xx,yy1,'r','LineWidth',1)
grid on

% points set03: 25
% cubicspline
x2 = linspace(-1,1,25);
y2 = f(x4);
yy2 = cubicSplineSimplySupported(x2, y2, xx);
figure(3)
plot(xx,yy,'b:','LineWidth',2)
axis([-1.1 1.1 -2 2])
hold on
plot(x2,y2,'ko','LineWidth',1)
plot(xx,yy2,'r','LineWidth',1)
grid on
```

test03 ------- Lebesgue constant -------

```
f = @(x) sin(x);
xx = linspace(-1,-0.95,10000);
yy = f(xx);
figure(1)
plot(xx,yy)
axis([-1.1 1.1 -2 2])
grid on

% points set04: 60
% chebyshev
```

```
xtemp = linspace(0,pi,25)';
x1 = cos(xtemp);
y1 = f(x1);
yy1 = fun_lagrangeInter(x1, y1, xx);
figure(2)
plot(xx,yy,'b:','LineWidth',2)
axis([-1 -0.97 -1 -0.7])
hold on
plot(xx,yy1,'r','LineWidth',1)
grid on

% points set03: 60
% cubicspline
x2 = linspace(-1,1,25);
y2 = f(x2);
yy2 = cubicSplineSimplySupported(x2, y2, xx);
figure(3)
plot(xx,yy,'b:','LineWidth',2)
axis([-1 -0.97 -1 -0.7])
hold on
plot(xx,yy3,'r','LineWidth',1)
grid on

% points set04: 60
x3 = linspace(-1,1,60);
y3 = f(x1);
yy3 = fun_lagrangeInter(x3, y3, xx);
figure(4)
plot(xx,yy,'b:','LineWidth',2)
axis([-1 -0.97 -1 -0.7])
hold on
plot(xx,yy3,'r','LineWidth',1)
grid on
```

test04 ---------- random noise ----------

```
epsilon = 1e-1;
f = @(x) 1./(1+12*x.^2);
xx = linspace(-1,1,10000);
yy = f(xx);
figure(1)
plot(xx,yy)
axis([-1.1 1.1 -2 2])
grid on

% points set03: 25
% cubicspline
x4 = linspace(-1,1,25)';
y4 = f(x4);
y4 = y4 + epsilon*randn(25,1);
yy4 = cubicSplineSimplySupported(x4, y4, xx);
figure(3)
plot(xx,yy,'b:','LineWidth',2)
axis([-1.1 1.1 -2 2])
hold on
plot(x4,y4,'ko','LineWidth',1)
plot(xx,yy4,'r','LineWidth',1)
```

```
grid on

err2 = max(yy4'-yy);
```

test05 ---------- boundary condition------

```
f = @(x) 1./(1+12*x.^2);
xx = linspace(-1,1,10000);
yy = f(xx);

% points set: 15
% cubicspline
M0 = 0;
Mn = 0;
x4 = linspace(-1,1,15)';
y4 = f(x4);
yy4 = cubicSplineSimplySupported(x4, y4, xx, M0, Mn);
figure(3)
plot(xx,yy,'b:','LineWidth',2)
axis([-1.1 1.1 -2 2])
hold on
plot(x4,y4,'ko','LineWidth',1)
plot(xx,yy4,'r','LineWidth',1)
grid on

temp = yy4'-yy;
err = max(temp(2001:8000))
```