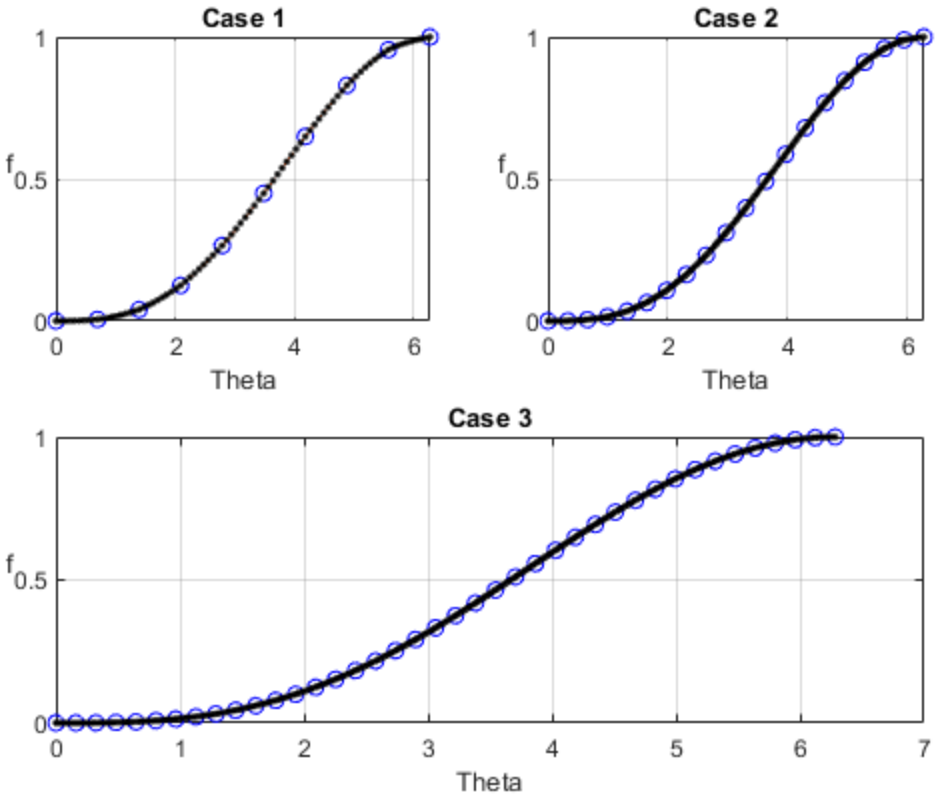
```
clear, clc;
close all;

% make data points
n1 = 10;
n2 = 20;
n3 = 40;

% exact case
xexact = linspace(0,2*pi(),100);
yexact = sin(xexact./4).^3;

%getting splines/x/y values
[s1,x1,y1] = naturalCubic(n1);
[s2,x2,y2] = naturalCubic(n2);
[s3,x3,y3] = naturalCubic(n3);

%Plot all 3 cases against each other
figure
subplot(2,2,1)
naturalCubicPlot(x1,y1,xexact,yexact,s1,n1, 'Case 1')
subplot(2,2,2)
naturalCubicPlot(x2,y2,xexact,yexact,s2,n2, 'Case 2')
subplot(2,2,[3 4])
naturalCubicPlot(x3,y3,xexact,yexact,s3,n3, 'Case 3')

%Funtion that take in number of data points and return splines with domain
%and range
function [splines,x,y] = naturalCubic(n)
    %making uniform grid based on given size n
    x = linspace(0,2*pi,n);
    y = sin(x./4).^3;

    % making tridiagonal for Thomas3 and solve for m1 to mn
    a = (1/6)*ones(1,n);
    b = (4/6)*ones(1,n);
    c = (1/6)*ones(1,n);
    e = ones(1,n);
    for i = 2:n-1
        e(i) = (y(i+1)-2*y(i)+y(i-1))/(x(i+1)-x(i));
    end
    e(1) = 0;
    e(n) = 0;
    m = THOMAS3(a,b,c,e,n);
    m(1) = 0; %first and last m set to 0
    m(n) = 0;

    %set up 10 points per subdivision
    %solve for splines
    splines = cell(n-1,1);
    for i = 1:n-1
        xspline = linspace(x(i),x(i+1),10);
```

```
        deltax = x(i+1)-x(i);
        a0 = m(i)/(6*deltax);
        a1 = m(i+1)/(6*deltax);
        a2 = y(i)/deltax-(m(i)*deltax)/6;
        a3 = y(i+1)/deltax-(m(i+1)*deltax)/6;
        yspline = a0.*(x(i+1)-xspline).^3 + a1.*(xspline-x(i)).^3 + a2.*(x(i
+1)-xspline) + a3.*(xspline-x(i));
        splines{i} = [xspline;yspline];
    end
end

%Plot function
%plot the original/exact function and data points
%plot splines
function naturalCubicPlot(x,y,xexact,yexact,splines,n,str)
    figure(1);
    plot(x, y, 'bo', xexact, yexact, '--')
    hold on; grid on;
    for i = 1:n-1
        plot(splines{i}(1,:),splines{i}(2,:),'.k')
    end
    title(str);
    xlabel('Theta');
    ylabel('f','Rotation',0);
end

%Sub-routine to solve for tridiagonal matrix system
function x = THOMAS3(a,b,c,d,n)

    %initial condition
    bbar(1) = b(1);
    cbar(1) = c(1);
    dbar(1) = d(1);

    %making upper triangle
    for i = 2:n
        multiplier = a(i)./bbar(i-1);
        abar(i) = a(i) - bbar(i-1).*multiplier;
        bbar(i) = b(i) - cbar(i-1).*multiplier;
        cbar(i) = c(i);
        dbar(i) = d(i) - dbar(i-1).*multiplier;
    end

    %initialize x of size n
    x = ones(1,n);

    %initialize end condition
    x(n) = dbar(n)/bbar(n);

    % Upward substitution AKA zip it up
    for i = n-1:-1:1
        x(i) = (dbar(i)-(cbar(i)*x(i+1)))/bbar(i);
    end
end
```



Published with MATLAB® R2021b