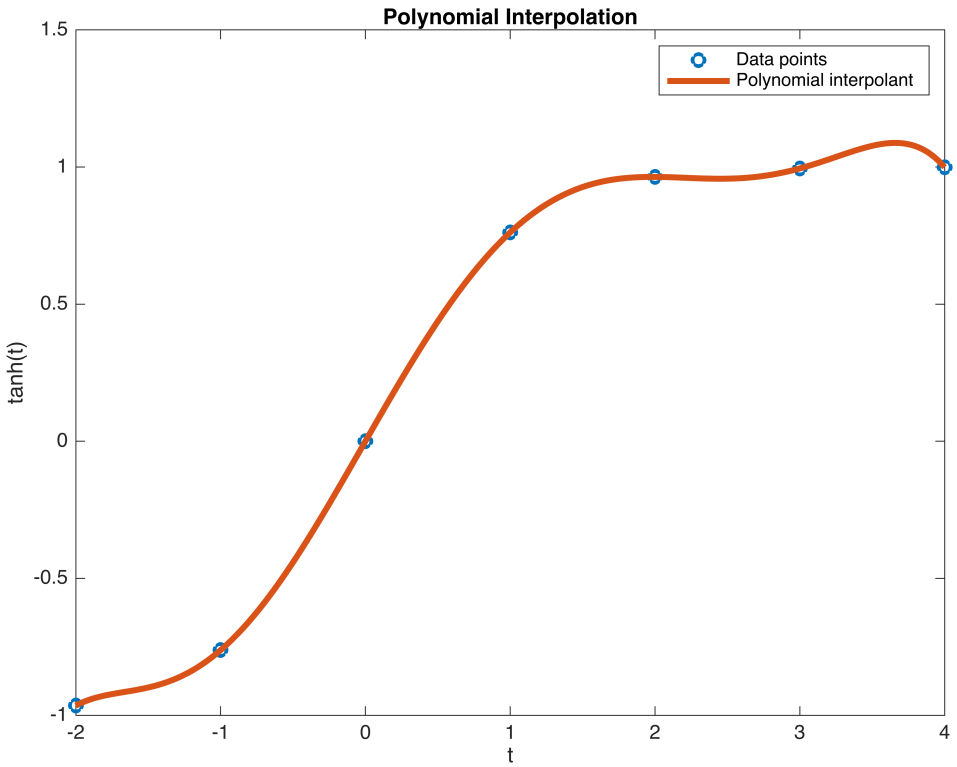
Muhideen Ogunlowo Math426 HW 9

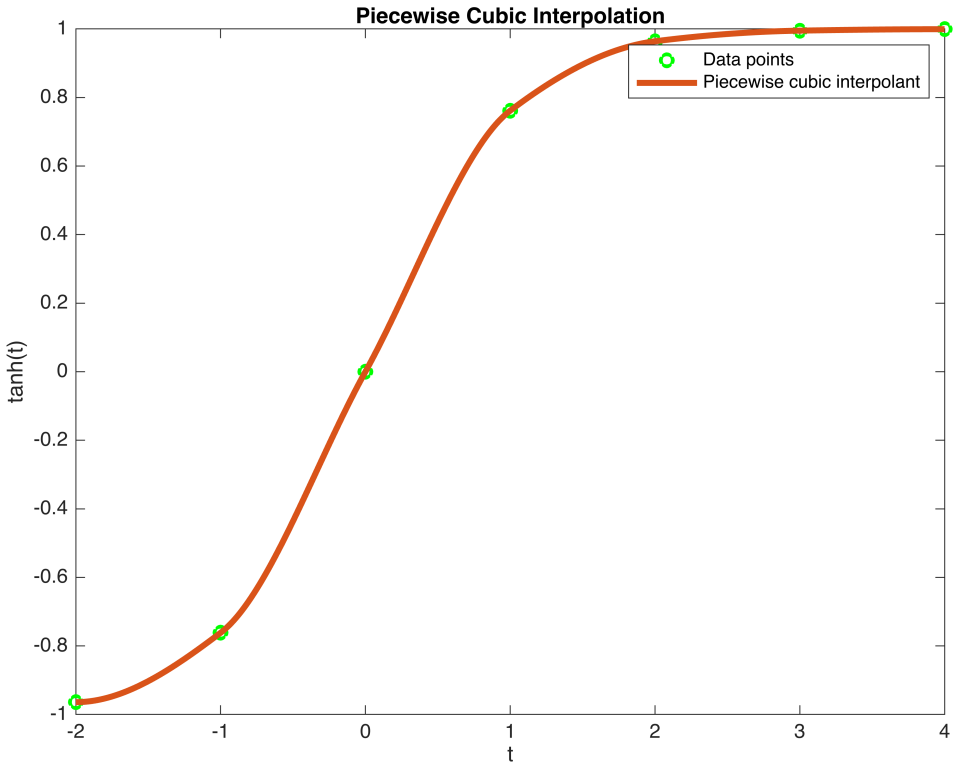
Exercise 5.1.1

```
t = (-2:4)'; %5.1.1
y = tanh(t);
p = polyfit(t, y, length(t) - 1); % Fit of polynomial of degree equal to
number of points minus one.

x = linspace(min(t), max(t), 1000); % Generates a finer grid for plotting
a = polyval(p, x); % Evaluates the polynomial at the grid points

plot(t, y, 'o', x, a, '-', 'linewidth', 3); % Plots both data points and
the polynomial
%legend and labeling
title('Polynomial Interpolation');
xlabel('t');
ylabel('tanh(t)');
legend('Data points', 'Polynomial interpolant');
```




```
t = (-2:4)';
y = tanh(t);
x = linspace(min(t), max(t), 1000); % Generates a finer grid for plotting
```

```
q = interp1(t, y, x, 'pchip'); % Piecewise cubic interpolating polynomial

plot(t, y, 'go', x, q, '-', 'linewidth', 3); % Plots both data points and
the piecewise cubic interpolant
%legend and labeling
title('Piecewise Cubic Interpolation');
xlabel('t');
ylabel('tanh(t)');
legend('Data points', 'Piecewise cubic interpolant');
```



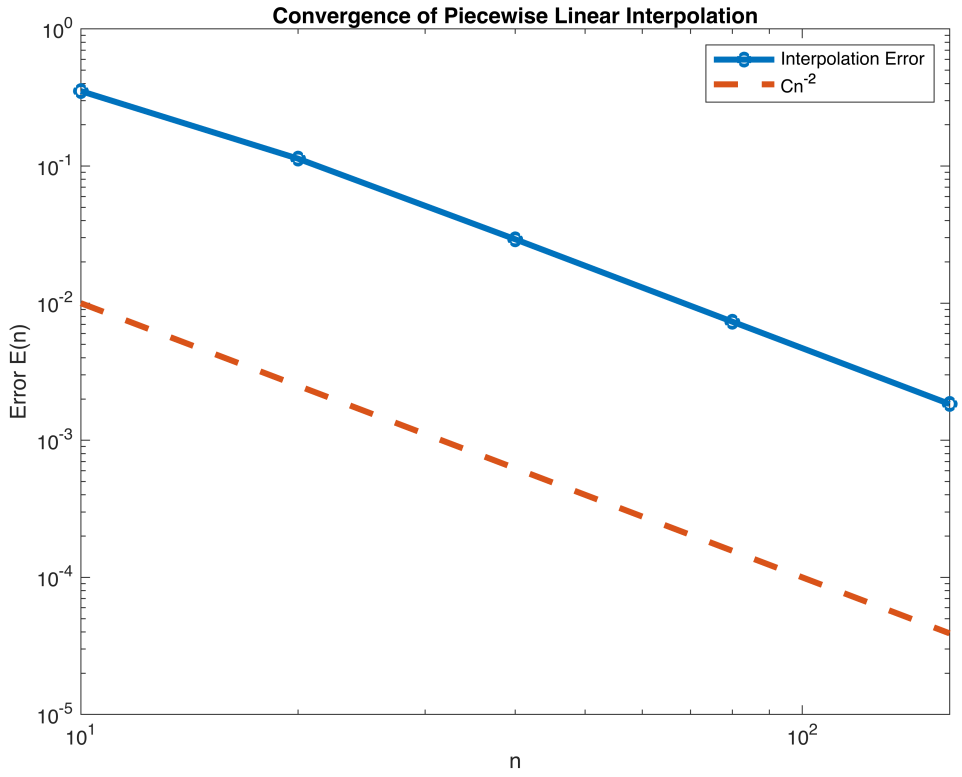
Exercise 5.2.1 a

```
% Define the function
f = @(x) cos(pi.^2 * x.^2);
x = linspace(0, 1, 1600)';
% Nodes n
n1 = 10*2.^(0:4)';

% Initialize array to store maximum errors
err_= 0*n;

% Loop over each n to perform interpolation and error estimation
for i = 1:length(n1);
    n = n1(i);

    % Equally spaced nodes and function values at those nodes
```

```
    t = linspace(0, 1, n+1)';
    y = f(t);

    % Interpolant
    p = plinterp(t, y);

    % Estimate the error
    err = max(abs(f(x) - p(x)));
    err_(i)= err;

end
```

```
n =
    10
n =
    20
n =
    40
n =
    80
n =
   160
```

```
C = 1; % This is an arbitrary value of C
n3 = linspace(min(n1), max(n1), 100);
E = C * n3.^(-2);

% Plotting
loglog(n1, err_, 'o-', n3, E, '--', 'linewidth', 3);
%legend and labeling
xlabel('n');
ylabel('Error E(n)');
legend('Interpolation Error', 'Cn^{-2}');
title('Convergence of Piecewise Linear Interpolation')
```



Exercise 5.2.4

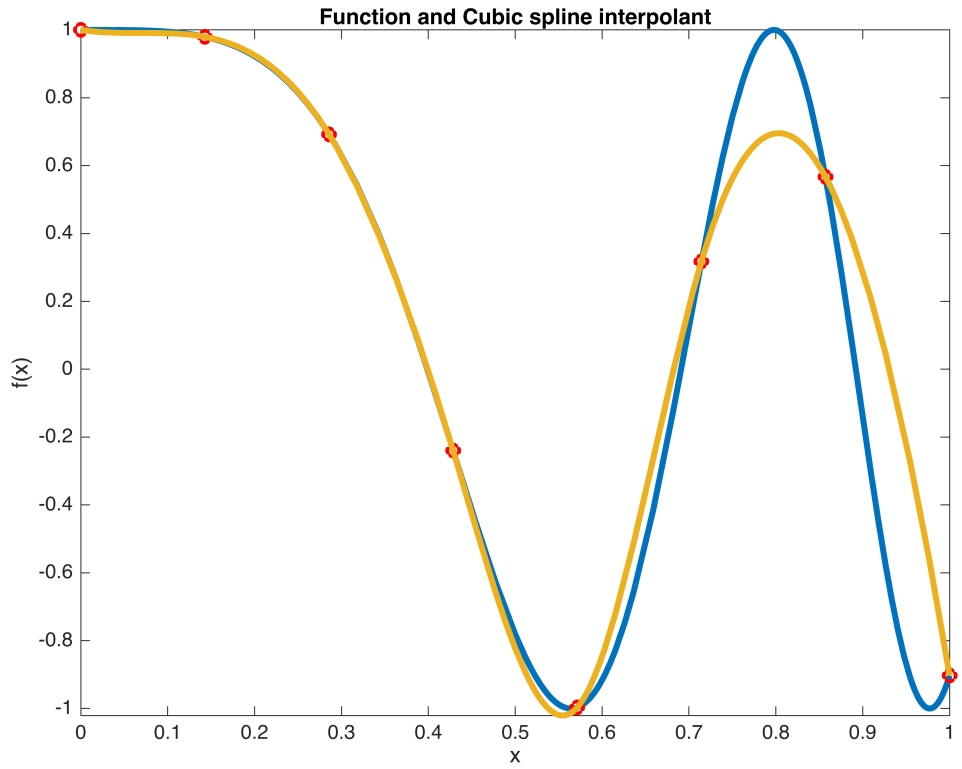
In the textbook, every interval only has two functions. Adding these hat functions from two intervals will always equal 1

For proof, the hat functions are split into two intervals t_k and t_{k+1} and then added together. These result is written out to be $\frac{x-t_k}{t_{k+1}-t_k}+\frac{t_{k+1}-x}{t_{k+1}-t_k}=\frac{t_{k+1}-t_k}{t_{k+1}-t_k}=1$

Exercise 5.3.3 a

```
format long
f= @(x) cos(pi.^2*x.^2); %Definition of function
fplot(f,[0,1],'linewidth', 3) %plot of function over 0,1
n=7; %number of interpolation points
t=linspace(0,1,n+1); %generates x-coordinates for interpolation.
y=f(t); %y values
hold on
plot(t,y ,'ro', 'linewidth', 3)
G=spinterp(t,y); %spline interpolation
hold off
hold on
fplot(G,[0,1], 'linewidth', 3)%plot of spline interpolation
```

```
hold off
%legend and labeling
xlabel('x'), ylabel('f(x)')
title('Function and Cubic spline interpolant')
```



Exercise 5.4.5

```
f=@(x) tan(2*x);%5.4.5 %functon definition
dfdx= @(x) 2*(sec(2*x)).^2; %first derivative
df2d2x= @(x) tan(2*x)*(8*(sec(2*x)).^2); %second derivative
exactValue=df2d2x(0.3) % f''(0.3)
```

```
exactValue =
   8.034738964488712
```

```
j= (-2:1:2)'; %values of j from -2..2
h=0.05; %h value of 0.05
t=(j.*h)+0.3;
w=fdweights(t-0.3,2); %call of finite diference weights function for x= 0.3
fdValue= w'*f(t) %finite difference value
```

```
fdValue =
   8.030596629020806
```

They are almost the same exact value, < 0.01 difference.

```
%Part b
```

```
newExValue= df2d2x(0.75) % f''(0.75)
```

```
newExValue =
     2.254535523463230e+04
```

```
tNew=(j.*h)+0.75;
newW= fdweights(tNew-0.75, 2);%call of finite diference weights function
for x=0.75
newFDvalue=newW'*f(tNew)% new finite difference value
```

```
newFDvalue =
    -2.913008276816167e+04
```

The reason why these finite differences are inaccurate is that the function tan(2x) has inflection points and also approaches asymptotes as x approaches $\frac{\pi}{4}$(or 0.7854) in radians) (which is close to the value of 0.75 chosen).

This can lead to large errors in finite difference approximations near these points. The step size h might be too large to capture the curvature of the function accurately at these points, leading to substantial numerical errors.

Exercise 5.5.3

$$f(a+h) = f(a) + h\,f'(a) + \frac{h^2}{2}f''(a) + \frac{h^3}{6}f'''(a)$$

This means :

$$f(a+h) - f(a) = h\,f'(a) + \frac{h^2}{2}f''(a) + \frac{h^3}{6}f'''(a)$$

Thus :

$$f(x+h) - f(x) = h\,f'(x) + \frac{h^2}{2}f''(x) + \frac{h^3}{6}f'''(x)$$

and :

$$f(x-h) - f(x) = -h\,f'(x) + \frac{(-h)^2}{2}f''(x) + \frac{(-h)^3}{6}f'''(x)$$

Which means

$$\frac{f(x) - f(x-h)}{h} = -\frac{1}{h}\left(-h\,f'(x) + \frac{(-h)^2}{2}f''(x) + \frac{(-h)^3}{6}f'''(x)\right)$$

$$\frac{f(x) - f(x-h)}{h} = f'(x) - \frac{h}{2}f''(x) + \frac{h^2}{6}f'''(x)$$

```
function S = spinterp (t,y)
% SPINTERP Cubic not-a-knot spline interpolation.
% Input:
% interpolation nodes (vector, length n+1)
% interpolation values (vector, length n+1)
```

```
% Output :
%S  not-a-knot cubic spline (function)
t= t(:); y=y(:); %ensure column vectors
n=length(t)-1;
h=diff(t);  %differences of all adjacent pairs
% Preliminary definitions.
Z = zeros (n);
I= eye(n); E=I(1:n-1,:);
J= I-diag(ones(n-1,1),1);
H= diag(h);

%Left endpoint interpolation:
AL=[I,Z,Z,Z];
vL= y(1:n);

%Right endpoint interpolation:
AR= [I, H, H^2, H^3];
vR= y(2:n+1);

% Continuity of first derivative:
A1= E*[Z, J, 2*H, 3*H^2];
v1= zeros(n-1,1);

% Continuity of the second derivative:
A2= E*[Z,Z,J 3*H];
v2= zeros(n-1,1);

%Not a knot conditions;
nakL= [zeros(1,3*n),[1,-1, zeros(1,n-2)]];
nakR= [zeros(1,3*n),[zeros(1,n-2), 1,-1]];

%Assemble and solve the full system
A= [AL;AR;A1;A2;nakL;nakR];
v=[vL;vR;v1;v2;0;0];
z= A\v;

%Break the coefficients into separate vectors.
rows = 1:n;
a= z(rows);
b= z(n+rows); c= z(2*n+rows); d= z(3*n+rows);
S= @evaluate;

%This function evalutes the spline when called with a value for x.
    function f = evaluate(x)
        f= zeros(size(x));
        for k=1:n %iterate over the pieces
            % Evaluate this piece's cubic at thepoints inside it.
            index=(x>=t(k)) & (x<=t(k+1));
            f(index) = polyval([d(k), c(k), b(k), a(k)], x(index)-t(k));
        end
```

```
    end
end
```

Function for Piecewise linear interpolation

```
function p = plinterp(t,y)
%PLINTERP: Piecewise Linear Interpolation
%Input:
% t: Interpolation nodes (vector, length n+1)
% y: interpolation values (vector, length n+1)
% Output:
% p: piecewise linear interpolant (function)
 n = length(t)-1
 p= @evaluate;
  %This function evaluates p when called.
function f = evaluate(x)
 f=0;
 for k=0:n
     f=f+y(k+1)*hatfun(x,t,k);
 end
end
end
```