interpolation

November 7, 2022

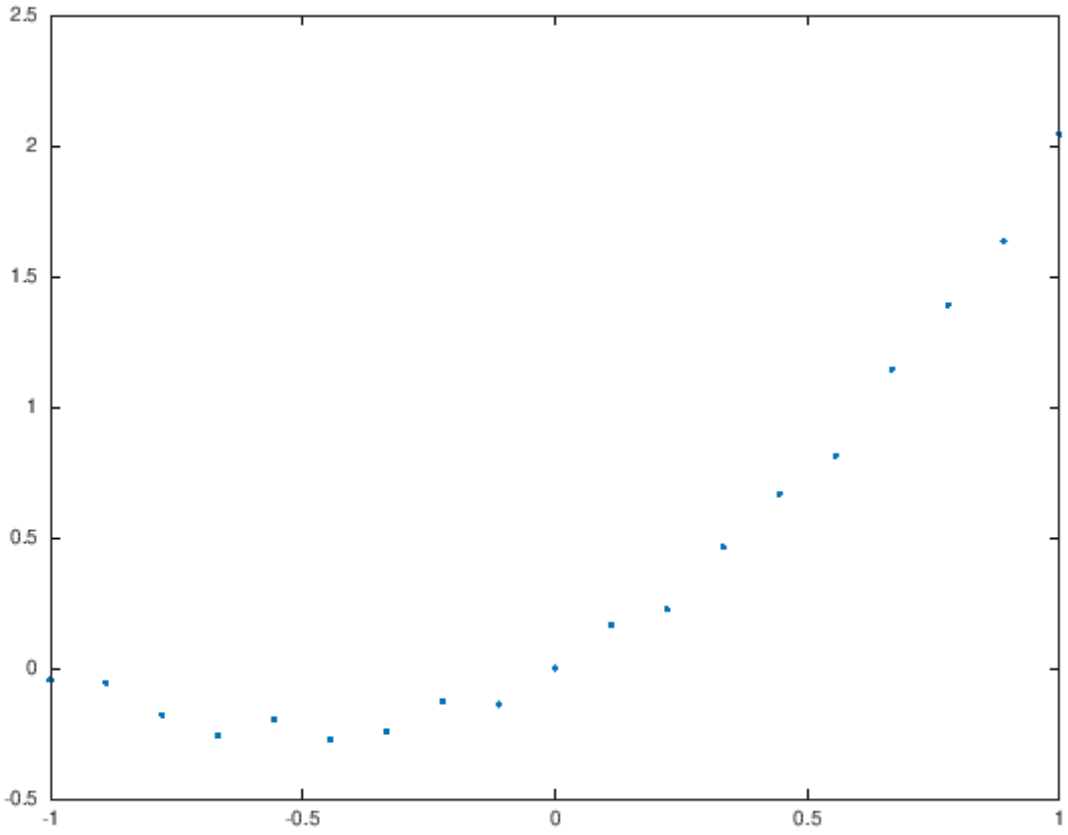
1 Interpolation

Before we go into specifics let us tackle the difference between interpolation and regression, since these are two different topics that often cause confusion when first learning about them.

Both techniques are about fitting a curve to a dataset, however one key difference is that in regression it is assumed that the points in the dataset are the true values of the function, while in regression you believe that the data set is noisy and there maybe an error. With that in mind in regression the approximated function does not have to match the data points exactly, while in regression it does. So in a sense while regression looks for minimization of the error between the points and the dataset and the curve value, interpolation is more of a connect the dots game. Of course there are many different ways to connect those dots, one of the first things we may think about is using a polynomial

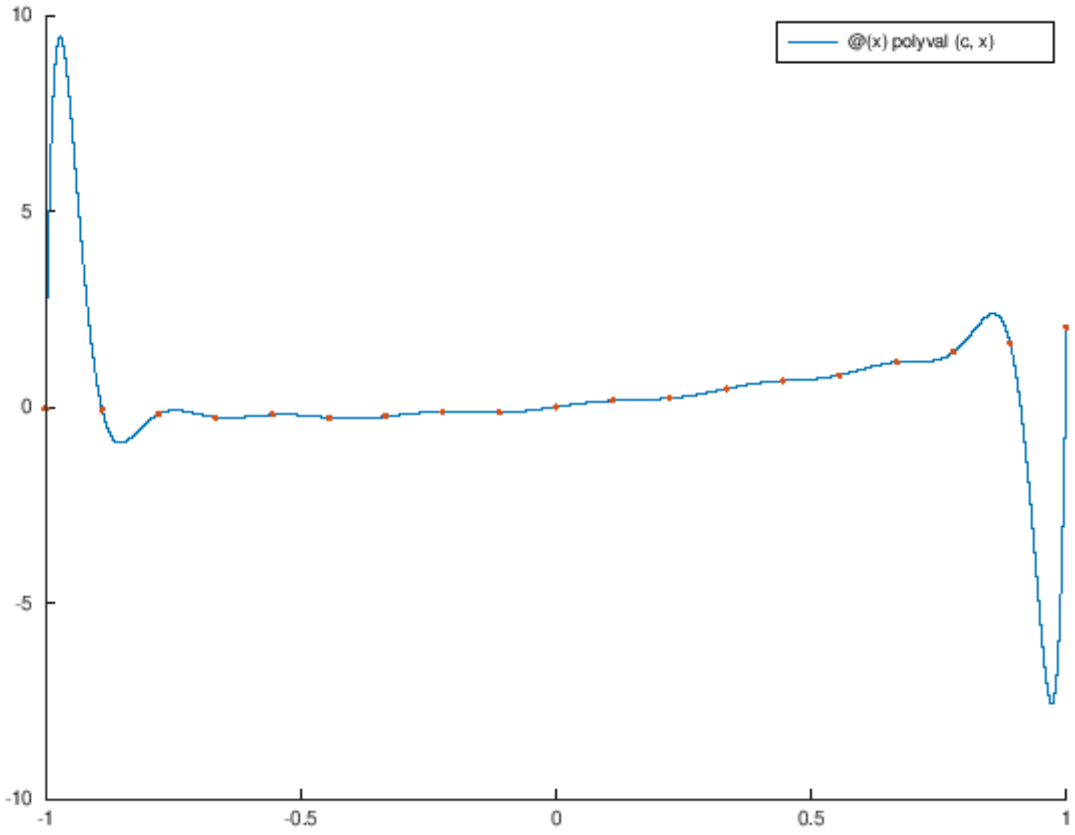
First let us create an artificial data set

[12]:
```
n=18; # number of points
t=linspace(-1,1,n+1); #interval
y= t.^2 .+ t+ 0.05*sin(20*t); # data
plot(t,y,'.')
```



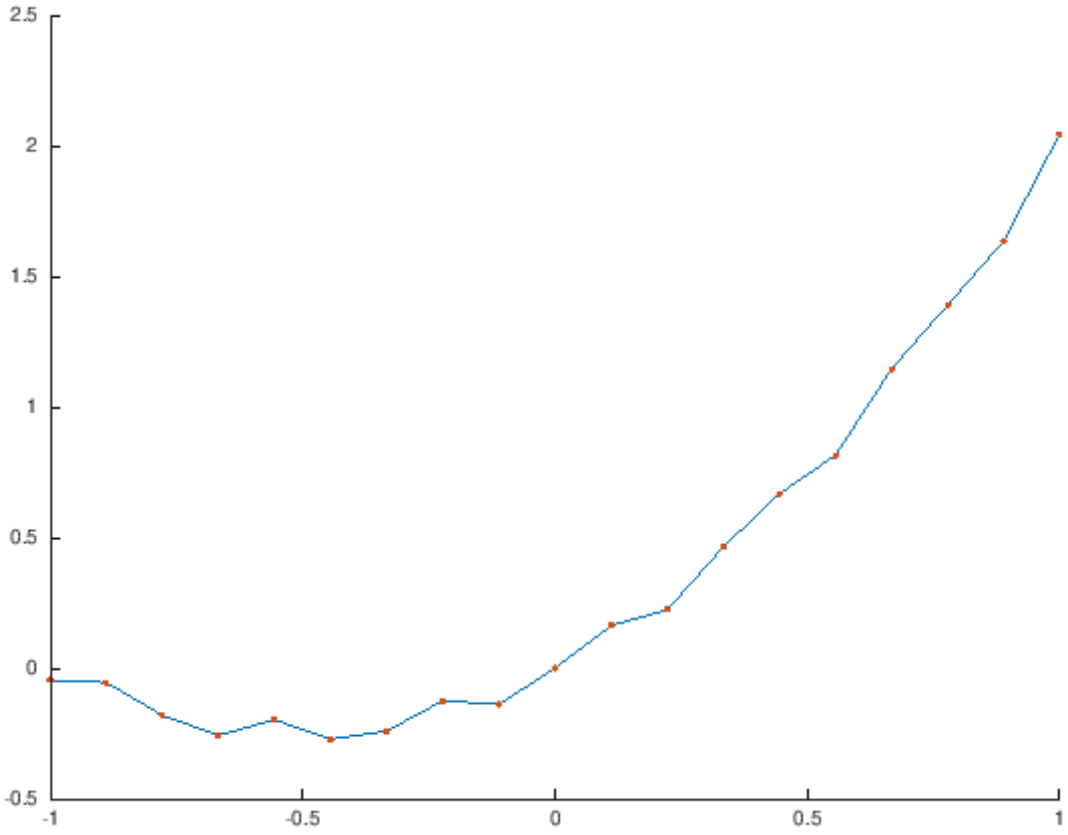
Now octave comes with polyfit which will gives us a polynomial interpolation of nth order, for now let us take n to be the same number of data points, we will use the polyval function to evaluate the polynomial and store its coefficients in c

[13]:
```
c=polyfit(t,y,n);
p=@(x) polyval(c,x);
hold on,fplot(p,[-1,1])
plot(t,y,'.')
```



Now, something is going wrong we can see that there are two huge bumps where there are no points, eventhough were passing through all of the points such bumps are undesirable they point to the interpolation being really sensitive to the presence of a point which is a feature we don't want. Let us push that thought aside for a minute and look at other ways of interpolation, perhaps just connecting each pair of points by a straight line (which is called piecewise linear interpolation) will work better. In octave it can be done using the interp1(t,y,x) function it takes t,y from the dataset and x as the interval where the connections are made

[14]:
```
x=linspace(-1,1,400);
hold on,plot(x,interp1(t,y,x))
plot(t,y,'.')
```

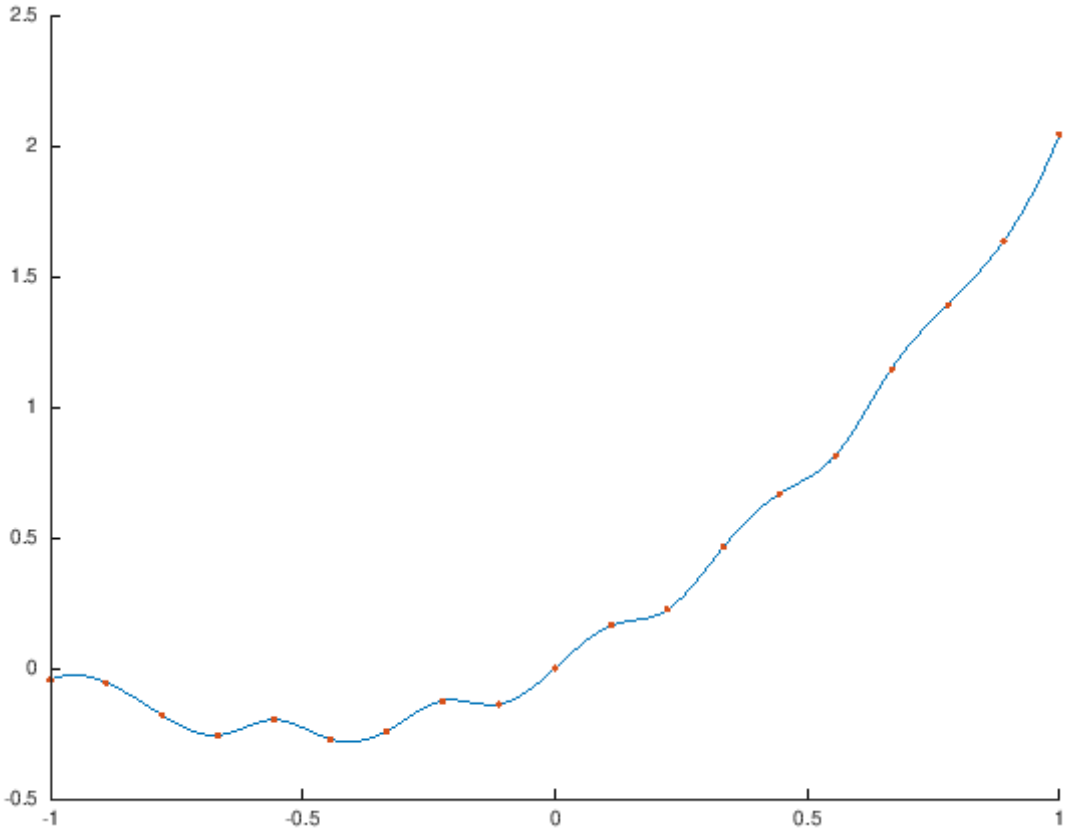


This interpolation looks nicer but it is not as smooth as we would like it to be, this is mainly to the points being too far apart, if they were closer together this would be a much better interpolation (You want to perform interpolation in large dense datasets for the most part)

Task see how this graph changes if you double the number of points? would this sort of interpolation be suitable for curve plotting?

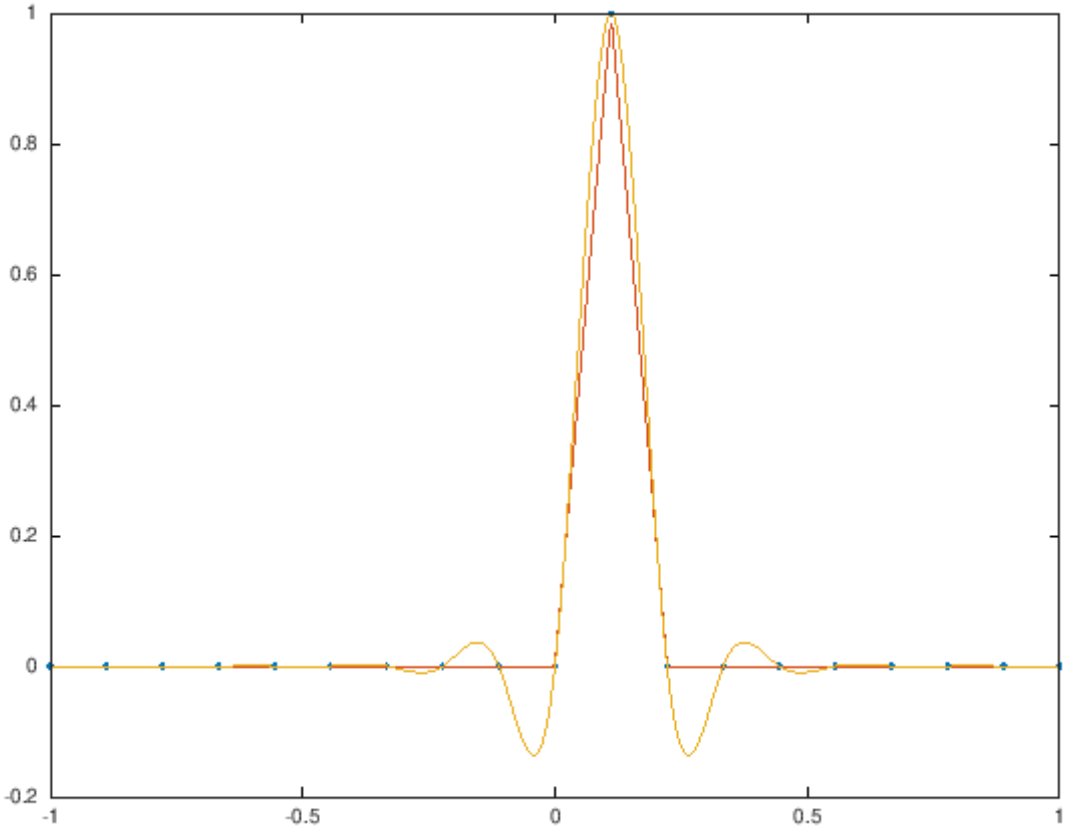
Another popular interpolation is connecting the dots with piecewise polynomials instead of lines splines, interp1 offers cubic polynomial piecewise interpolation if we pass the keyword spline as a last argument. This sort of interpolation is preferred when the dataset is not large/dense since the outcome will be much smother than using lines

[15]:
```
x=linspace(-1,1,400);
hold on,plot(x,interp1(t,y,x,'spline'))
plot(t,y,'.')
```

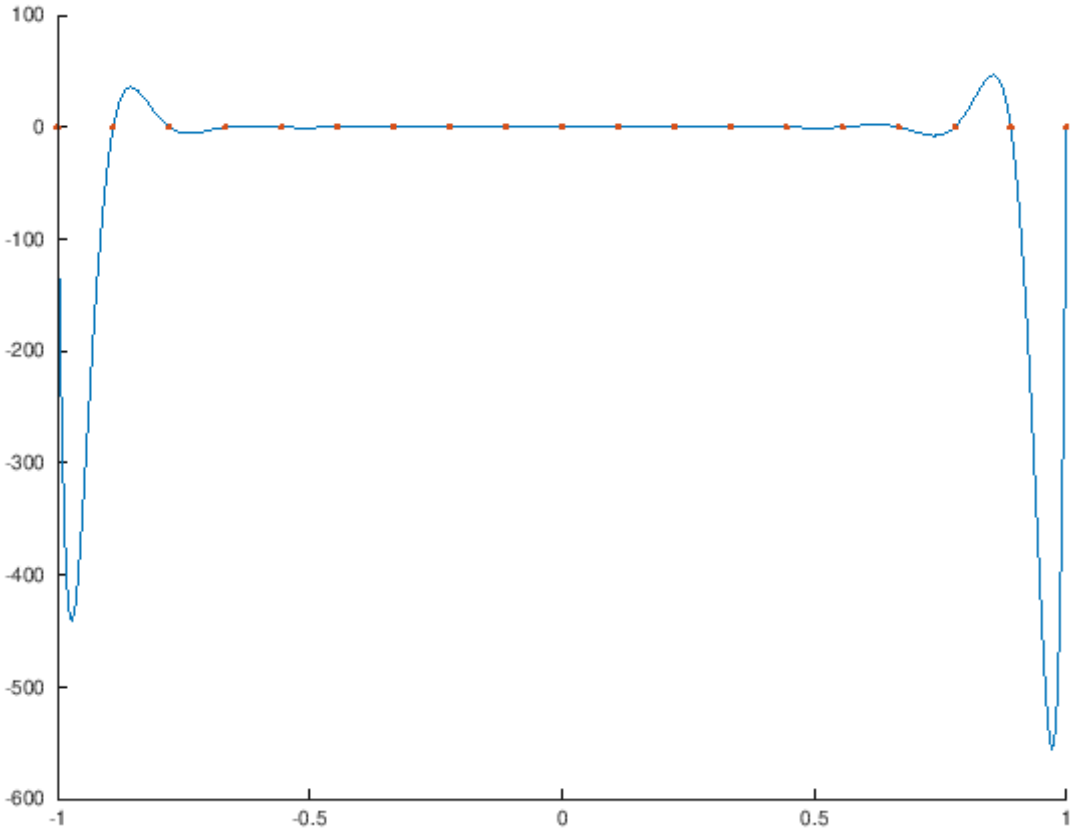


Just as condition numbers where an indicator of how bad the result of a given calculation might be depending on a matrix, there is also such a concept for interpolation functions, the idea translates in the so called cardinal functions, From a practial point of view such functions translate into evaluating interpolation with all of the data points 0 except the k^{th} point which we set to one, the result of said interpolation is the k^{th} cardinal function

[16]:
```
y=[zeros(10,1);1;zeros(n-10,1)]; %11th cardinal function
plot(t,y,'.'),hold on
plot(x,interp1(t,y,x)) %10th cardinal function for piecewise linear
hold on, plot(x,interp1(t,y,x,'spline')) % for splines
```



[17]:
```
c=polyfit(t,y',n);
p=@(x) polyval(c,x);
hold on,plot(x,p(x))  % 11th cardinal function for nth order polynomyal
plot(t,y,'.')
```



The condition number which again has the same sort of meaning, in this case it is proportional to the maximum of the cardinal functions, which we want to be of order one (same as for numerical linear algebra). The higher it is the most sensitive to individual points the interpolation is

Task: Check that for piecewise linear and splines it is always proportional to 1, explain why thats the case

Task

Given the two datasets

x1=[0 0.51 0.96 1.06 1.29 1.55 1.73 2.13 2.61]

y1=[0 0.16 0.16 0.43 0.62 0.48 0.19 0.18 0]

and

x2=[0 0.58 1.04 1.25 1.56 1.76 2.19 2.61];

y2=[0 -0.16 -0.15 -0.3 -0.29 -0.12 -0.12 0];

Find an adequate interpolation for each of the datasets, what shape do they form ?

Hint: if the shape is not clear try another interpolation

1.1 Piecewise linear interpolation

We did this at the begginning and saw that in a way, it is a matter of connecting the dots through a straight line. let us assume that the points are given in order so that $t_0 < t_1 < .... < t_n$. The interpolant function p(x) is then given by the line connecting y_{k+1} and y_k

$$p(x) = y_k + \frac{y_{k+1} - y_k}{t_{k+1} - t_k}(x - t_k) \tag{1}$$

on each interval $[t_k, t_{k+1}]$

However instead of basing our implementation on the equation above, we will use the so called hat functions, so that the interpoland can be written as a linear combination of a preselected finite set of functions, the hat functions which are defined as

$$H_x = \begin{cases} \frac{x-t_{k-1}}{t_k-t_{k-1}} & x \in [t_{k-1}, t_k] \\ \frac{t_{k+1}-x}{t_{k+1}-t_k} & x \in [t_k, t_{k+1}] \\ 0 & otherwise \end{cases} \tag{2}$$

The hat functions are a basis of the set of functions that are continuous and piecewise linear (relative to t). So any continuous piecewise linear function can be written as a linear combination of hat functions

$$f(x) = \sum_{k=0}^{n} c_k H_k(x) \tag{3}$$

At each x, $H_k(x)$ is the larger of to options zero, or the smaller of the two linear functions

Task Complete the definition of the hat function

[59]:
```
function H=hatfun(x,t,k)
%HATFUN the piecewise linear basis function.
% Input:
% x the evaluation points
% the nodes (data points)
%k node index
%Output: values of the kth hat function
n=size(t)(1)-1;
k=k+1; % so we start counting from one
%We define a fake node for dealing with the first and last funcs H_{0},H_{n}
t=[2*t(1) .- t(2);t(:);2*t(n+1) .- t(n)];
k=k+1;% adjust index for this ficticius node

H1=(x .- t(k-1))/(t(k) .- t(k-1)); %upward slope
H2=(t(k+1) .- x)/(t(k+1) .- t(k)); %downword


H=min(H1,H2);
```

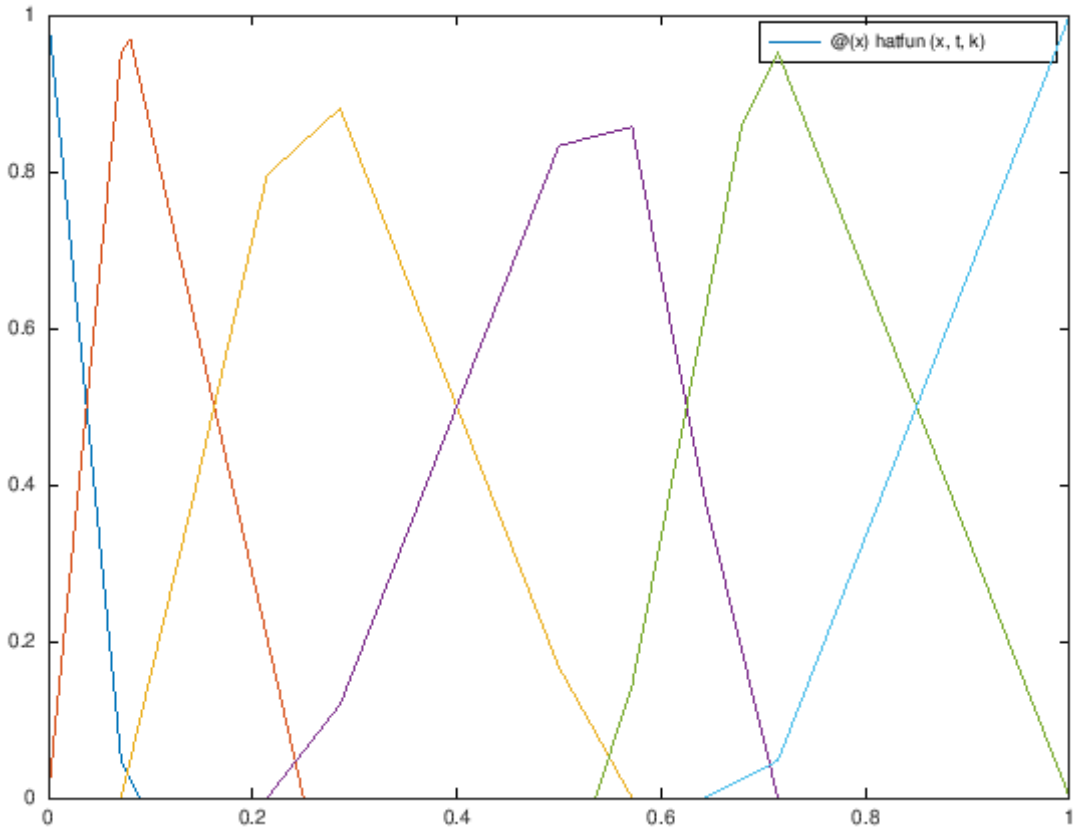
```
H=max(0,H); # this equivalent to checking the intervals as explained in the
 ↪tutorial
end
```

[60]:
```
t=[0 0.075 0.25 0.55 0.7 1]';
```

One little thing to be aware aout is that Octave's fplot function is not good at interpolation of piecewise functions, which yields incorrect plots, one needs to be careful about this as it might be a source of confusion

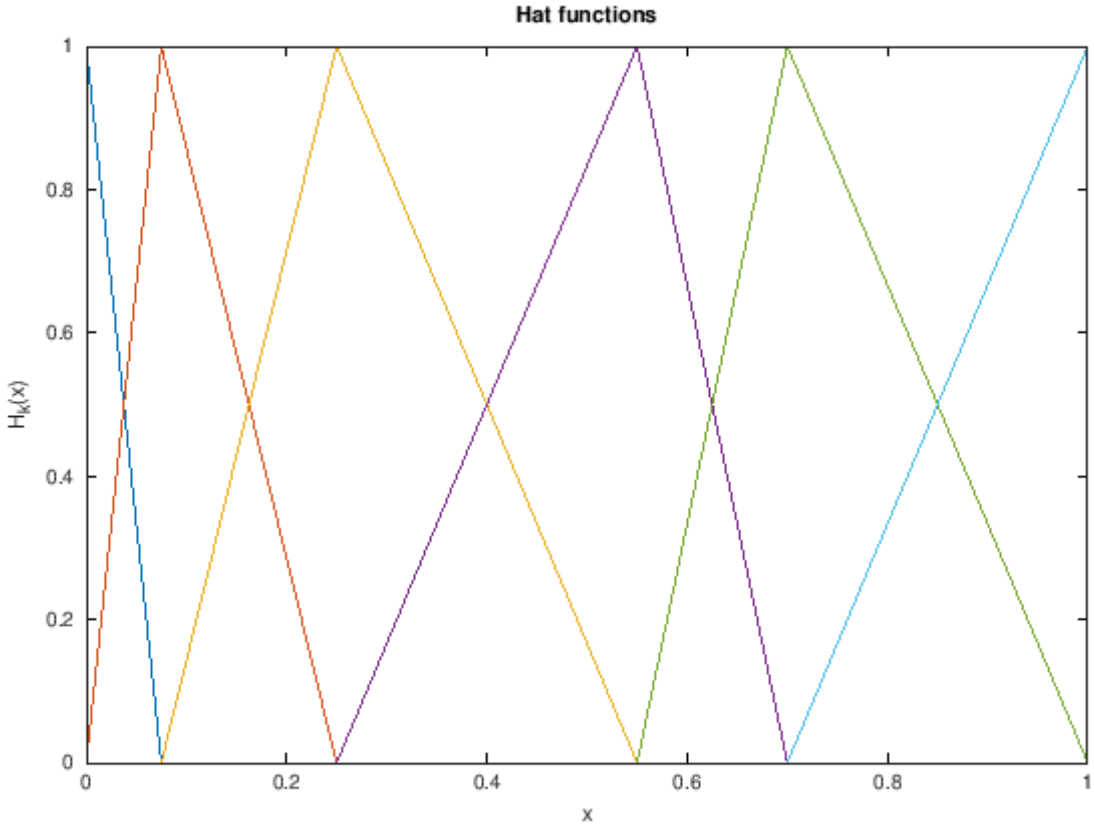
Advice: Avoid fplot in octave (it works nicely in matlab)

[61]:
```
for k=0:5
fplot(@(x) hatfun(x,t,k),[0,1])
hold on
end
```



A proper way of plotting would be making the dataset dense so that linear interpolation from plot works smoothly

[62]:
```
x=linspace(0,1,1000);
for k=0:5
    y=hatfun(x,t,k);
    plot(x, y)
    hold on
end
xlabel('x')
ylabel('H_{k}(x)')
title('Hat functions')
```



One remarkable thing about the hat functions is that they are cardinal functions for piecewise linear interpolation, meaning they satisfy

$$H_k(t_i) = \begin{cases} 1 & i = k \\ 0 & otherwise \end{cases} \tag{4}$$

That makes things easier because then we simply have

$$f(x) = \sum_{k=0}^{n} y_k H_k(x) \tag{5}$$

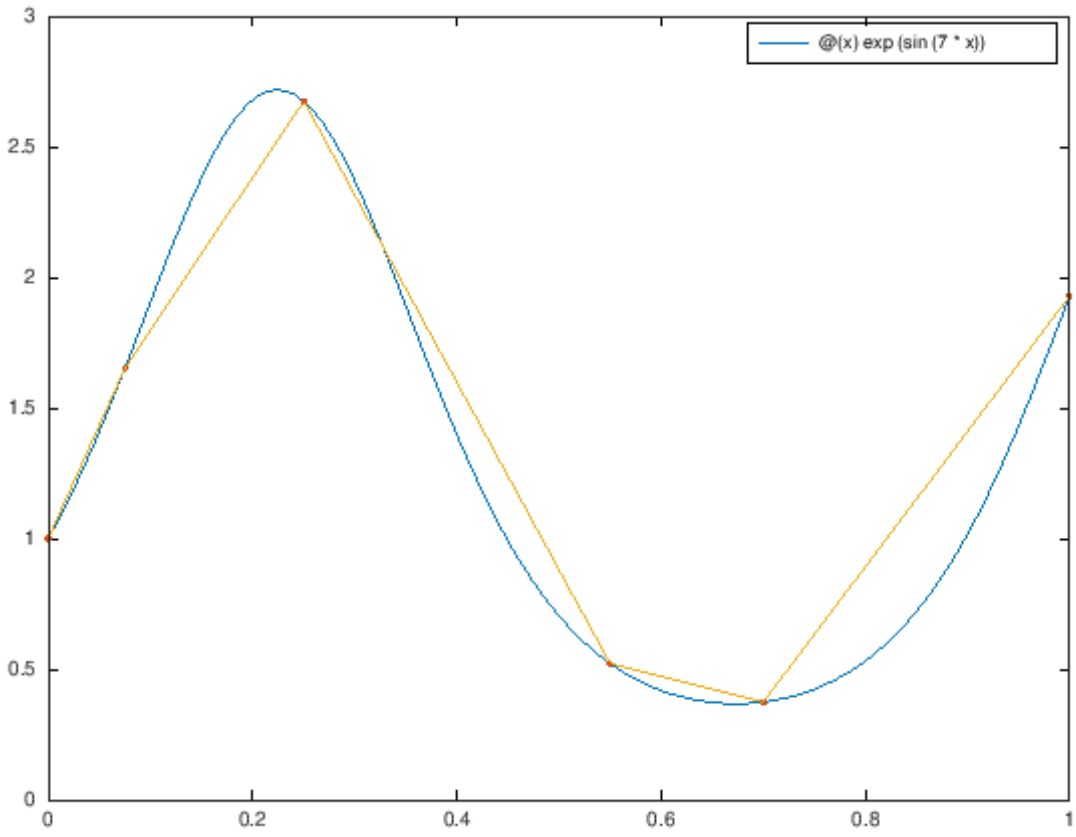
Let us see how linear piecewise interpolation works in action

Task: Create your own function that performs linear piecewise interpolation, test it on the next cell by substiting interp1 for your own function

You may use the template below and fill in the procedure we described above or simply find the lines that connect two points in a for loop

[72]:
```
% space for your function
function f=linearp(t,y,x)
    n=; % complete function
    f=0;
    for k=0:n
        f=; # Complete function
    end
end
```

[75]:
```
f=@(x) exp(sin(7*x));
t2=linspace(0,1,200);
x=linspace(0,1,200);
fplot(f,[0,1])
y=f(t);
hold on, plot(t,y,'.')
hold on,plot(x,interp1(t,y,x))
```



Finally let us convince ourselves that piecewise linear interpolation is perfectly good when the number of points is high, to do that let us simply plot the error (true value of the curve minus value of the interpolation at that value), and see that it goes down really fast, making this a perfectly good simple interpolation for applications such as curve plotting when the number of points can be increased for better resolution

[29]:
```
f=@(x) exp(sin(7*x));
x=linspace(0,1,10000);
n_=2.^(3:10)';
err_=0*n_;
for i=1:length(n_)
    n=n_(i);
    t=linspace(0,1,n+1)';
    p=@(x)interp1(t,f(t),x);
    err=max(abs(f(x)-p(x)));
    err_(i)=err;
end
```

[30]:
```
loglog(n_,err_,'.-')
xlabel('Points')
ylabel('Error')
```