Lecture 2

Operations with arrays

Arrays can be created as row or column vectors

```
r=[1 2 3]
```

```
r = 1×3
     1     2     3
```

```
c=[1;2;3]
```

```
c = 3×1
     1
     2
     3
```

Multiplying a row with a column vector yields a scalar product $p = \sum r_i \cdot c_i$

```
r*c
```

```
ans = 14
```

```

```

To instead multiply corresponding elements of two vectors use the period "." symbol in combination with the mulitplication operator "*":

```
[1 2 3].*[2 3 4]
```

```
ans = 1×3
     2     6    12
```

Further examples of the period "." way of indicating element-wise operation

```
[2 6 12]./[1 2 3]
```

```
ans = 1×3
     2     3     4
```

```
linspace(2,2,5).^(0:1:4)
```

```
ans = 1×5
     1     2     4     8    16
```

The above is the same as

```
[2 2 2 2 2].^[0 1 2 3 4]
```

```
ans = 1×5
     1     2     4     8    16
```

Matlab uses round brackets for array indexing. The first element has index 1, the last index can be access with the keyword end

```
a=[100 200 300 400];
a(1)+a(2)+a(3)+a(end)
```

```
ans = 1000
```

Indices itself can be arrays with multiple values

```
i=[2 4];
a(i)
```

```
ans = 1×2
   200   400
```

Generating arrays of random numbers

Uniform between 0 and 1

```
rand(1,5)
```

```
ans = 1×5
    0.6944    0.9249    0.3706    0.6388    0.8897
```

Normal with zero mean and unit standard derivation

```
randn(1,5)
```

```
ans = 1×5
   -0.0728    0.3306   -0.1103   -0.9248   -0.8009
```

Example: smooth a noisy signal

Generate a periodic signal with overlayed normal noise

```
n = 101;
t = linspace(0,1,n);
y=sin(2*pi*2*t)+.15*randn(1,n);
```

Take a moving average between three neighbors

```
ysm = [.5*y(1)+.5*y(2) (y(1:end-2)+y(2:end-1)+y(3:end))/3 .5*y(end-1)+.5*y(end)]
```

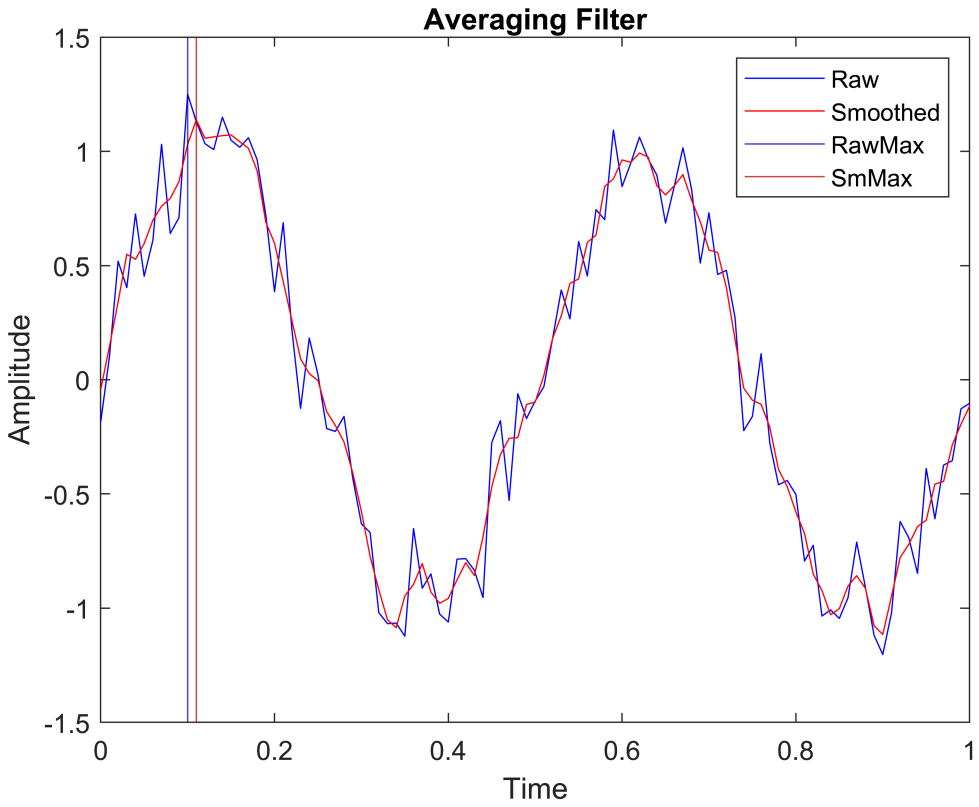
```
ysm = 1×101
   -0.0422    0.1449    0.3408    0.5494    0.5274    0.5962    0.6978    0.7603 ···
```

Locate amplitude, index and time of maximum of raw and smoothed signals

```
[ymax,idx_max] = max(y); [ysm_max, idx_sm_max] = max(ysm);
[ymax t(idx_max) ysm_max, t(idx_sm_max)]
```

```
ans = 1×4
    1.2505    0.1000    1.1379    0.1100
```

```
plot(t,y,'b-');hold on;plot(t,ysm,'r-');title('Averaging Filter');
xlabel('Time');ylabel('Amplitude');
xline(t(idx_max),'b-');xline(t(idx_sm_max),'r-');
legend('Raw','Smoothed','RawMax', 'SmMax');hold off;
```



Functions and Programming

Matlab supports used defined function definitions and common program paradigms. The function defintion syntax is

function return_value(s) **=** function_name**(** argument(s) **)**

end

Matlab requires functions to be defined in a seperate script file, or at the very end of a document. To see the implementation of the following function see the last slides

```
filter3points([]);filter3points([5]);filter3points([2 4]);
```

```
ret =

     []
ret = 5
ret = 1×2
     3     3
```

```
filter3points([1 2 3 4])
```

```
ret = 1×4
    1.5000    2.0000    3.0000    3.5000
ans = 1×4
    1.5000    2.0000    3.0000    3.5000
```

Numerical derivatives with finite differencing

The Matlab function **diff**(a) takes an array a as input and returns an array da of differences between elements

$$\Delta a_i = a_{i+1} - a_i$$

The resulting array da is by one element shorter than the original array.

```
a=[1 2 4 8];
diff(a)
```

```
ans = 1×3
     1     2     4
```

Given arrays of x and function values y=f(x), the derivative df(x)/dx can be approximated by the ratios of **diff**(y)./**diff**(x)

Central differences

An alternative way of calculating a numerical derivative is by taking central differences. Here the derivative at each point is approximated by the difference between both neighbors devided by two units.

$$\Delta\, a_i = \frac{(a_{i+1} - a_{i-1})}{2}$$

The scheme leaves the boundrary given by the first and last points unspecified. A linear extrapolation gives

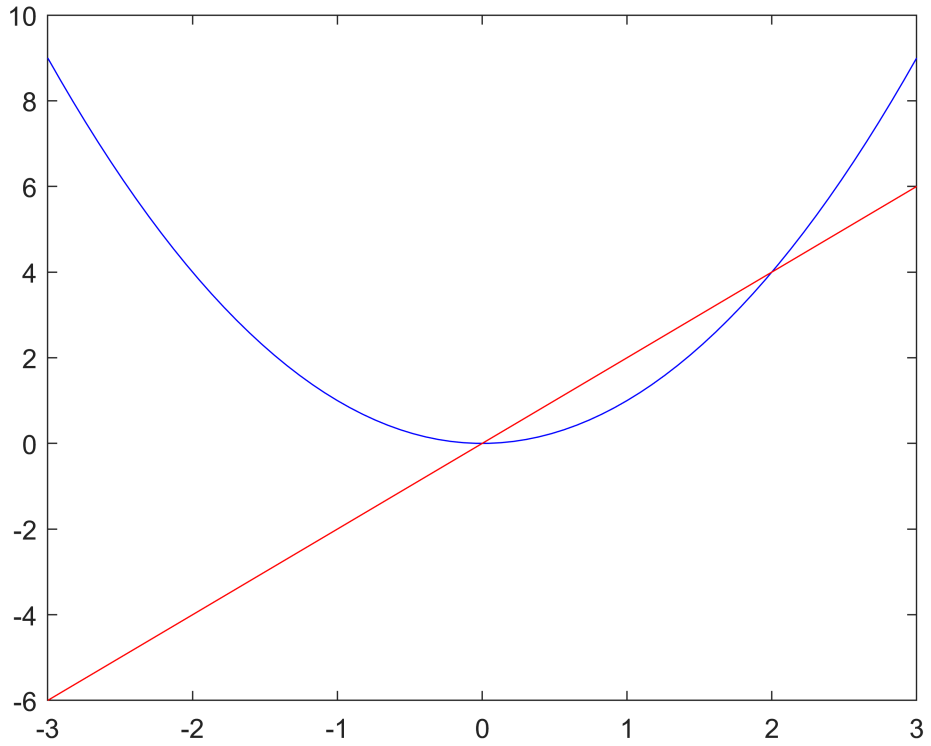
$$\Delta a_1 = \Delta a_2 - (\Delta a_3 - \Delta a_2) = 2 \cdot \Delta a_2 - \Delta a_3$$

$$\Delta a_N = \Delta a_{N-1} + (\Delta a_{N-1} - \Delta a_{N-2}) = 2 \cdot \Delta a_{N-1} - \Delta a_{N-2}$$

Taking differences to approximate derivatives is use in numerically simulating differential equations. The type of choice of the differencing, for example forward (upwind) vs central is important to ensure stability of the simulation. This will be important when looking at solving ODE or PDEs.

Example: central derivative for the function y = x^2

```
x=linspace(-3,3,70);
y=x.^2;
dy=(y(3:end)-y(1:end-2))/2;
dx=(x(3:end)-x(1:end-2))/2;
dydx=dy./dx;
dydx = [2*dydx(1)-dydx(2) dydx 2*dydx(end)-dydx(end-1)];
plot(x,y,'b-'); hold on; plot(x, dydx, 'r-'); hold off;
```



Parallel RCL Circuit

Consider a circuit with a resistor R, a inductor L and capacitor C all in parallel. The currents are additive

$$I = I_R + I_L + I_C$$

Subsituting the respective impedences $Z = \frac{V}{I}$ as the ratio of voltage to current accross the respective elements

$\frac{V}{Z} = \frac{V}{Z_R} + \frac{V}{Z_L} + \frac{V}{Z_C}$ with $Z_R = R,\ Z_L = j\omega\, L,\ Z_C = \frac{1}{j\omega C}$

yields the formula for the total impedance as

$$Z = \frac{1}{\frac{1}{Z_R} + \frac{1}{Z_L} + \frac{1}{Z_C}} = \frac{Z_R Z_L Z_C}{Z_R Z_L + Z_R Z_C + Z_L Z_C}$$

Paralell RLC Circuit Example

```
V=50;
w=2*pi*50;
R=50;Zr = R;
L=0.3;Zl= j*w*L;
C=15E-6;Zc= 1/(j*w*C);
Zt=Zr*Zl*Zc/(Zr*Zl+Zr*Zc+Zl*Zc)
```

```
Zt = 45.9997 + 13.5652i
```

Calculating phase shift and total impedance

```
angle(Zt)*360/(2*pi)
```

```
ans = 16.4306
```

```
abs(Zt)
```

```
ans = 47.9581
```

Tasks

1. Create a function that computes the numerical second derivative of an array y be taking central differences

$$\mathrm{f2} = \frac{y_{i+1} - 2y_i + y_{i-1}}{(\Delta\, x^2)}$$

where we assume a constant spacing $\Delta\, x$ throughout. Use your function to compute the second derivative of the function y=x^3 on the interval between -3 and 3 for 70 points just as we did in the lecture above and graph both the function y, as well as its first and second derivatives.

2. Create a function that computes the total impedance of the parallel RLC circuit above and plot the absolute value of the impedance vs. angular frequency w. Compare your results to a plot for the corresponding RLC circuit that has all elements in series from last week.

```
function ret = filter3points(y)
   if size(y)== 0
       ret = []
   elseif size(y)==1
       ret = y
   elseif size(y)==2
       ret = (0.5*y(1)+0.5*y(2)).*[1 1]
   else
      ret = [.5*y(1)+.5*y(2) (y(1:end-2)+y(2:end-1)+y(3:end))/3 .5*y(end-1)+.5*y(end)]
   end
end
```