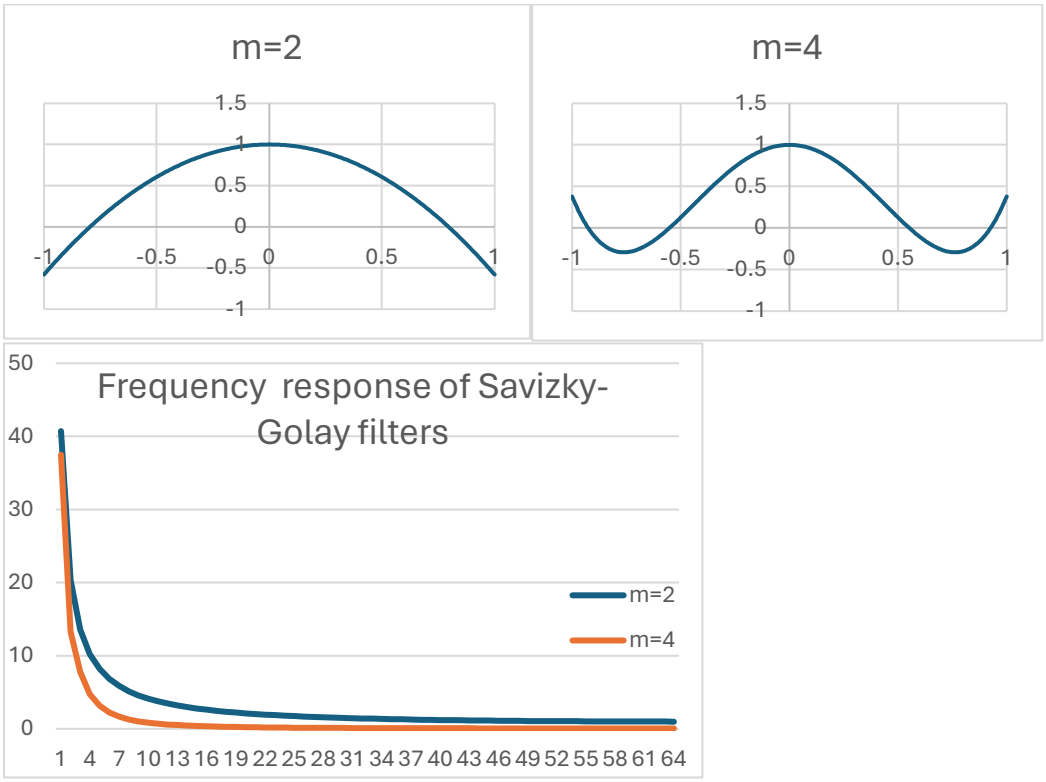
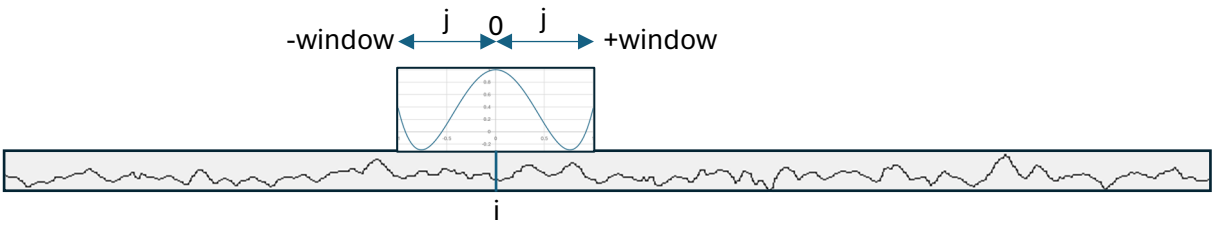
Savitzky-Golay

Define window size in ms. Full window is 2*window size+1

Apply the quadratic (m=2) function or quartic (m=4) as a sliding window along the time series to be smoothed. The wider the window the poorer thew time resolution



Convolve window with time series.



```
procedure TdlgData3DFilter.FilterDataSavGol(var ProcessedArray: TProcessedPosRecArray);  //Savitsky-Golay generalised moving average filter that convolves a polynomial with the signal to smooth (low pass filter)
var n,a: single;
   m4: boolean;                                  // The window size defines the cutoff frequency of the  low-pass filter
  procedure AddSample(j,t,dt: integer);          // The total length of the filter window must be a positive odd integer
                                                 // less than the length of the TimeSeries
  var w: single;                                 //m2 = quadratic polynomial m4 = quartic  polynomial
  begin                                          //all times measured in ticks (100,000 ticks per second)
   w:=(ProcessedArray[j].time - t)/dt;           //  calculate w (window time) as a fraction of the window size
                                                 //(i.e. to lie in the range 0 to 1)
   if m4 then
    w:=-4.45*sqr(w)+3.83*sqr(sqr(w))+1          //calculate the quartic window weight
   else
    w:=-1.58*sqr(w)+1;                          // or calculate the quadratic window weight
   a:=a+ProcessedArray[j].a*w;                  //apply the weight
   n:=n+w;
  end;
var i,j,t,dt: integer;
   ProcessedArray2: TProcessedPosRecArray;
begin
  ProcessedArray2:=copy(ProcessedArray);
  dt:=seWidth.Value*TicksPerSec div 1000;       //sewidth is window size in ms
                                                // sewidth of 1ms =  1*100,000/1000 = 100ticks
  m4:=rbM4.Checked;
  for i:=low(ProcessedArray) to high(ProcessedArray) do //ProcessedArray contains signal times series to be filtered
  begin
   n:=1;
   a:=ProcessedArray[i].a;                      //signal amplitude
   t:=ProcessedArray[i].time;                   //frame timestamp in ticks (in AAA samples are not forced to be
                                                //evenly spaced in time although they mostly are).
   j:=i-1;
   while (j >= low(ProcessedArray)) and (ProcessedArray[j].time >= t-dt) do   //Filters forward in time. Does not filter samples that are less than the specified window width (in ms) from the sample being smoothed
   begin                                        //i.e. applies the left side of the symmetric window function
    AddSample(j,t,dt);
    dec(j);
   end;

   j:=i+1;
   while (j <= high(ProcessedArray)) and (ProcessedArray[j].time <= t+dt) do  //Filters backward in time. Cannot filter samples that are more than the specified window width (in ms) from the sample being smoothed
   begin                                        //i.e. applies right side of the symmetric window function
    AddSample(j,t,dt);
    inc(j);
   end;

   ProcessedArray2[i].a:=a/n;                   //divide by cumulative weight total so as not to change the amplitude.
  end;

  move(ProcessedArray2[0],ProcessedArray[0],sizeof(ProcessedArray[0])*length(ProcessedArray));

  PosVelAcc(ProcessedArray);
end;
```