



```
function y=der1

%

% This function computes the derivative of
% a noisy function by mollification and GCV.
% The only input is the noise in the data
% for purposes of simulation.
%
% 1993-Copyright Diego A. Murio
%
% The program is completely automatic.
%
% The sample step size is 1/128 and the
```

```
% data is obtained in the closed interval [0,1].

% The derivative is computed everywhere in the

% interval, including the BOUNDARIES.

%

% -----

% This program callss the following M-files:

% MOLGCV1, MOL1, MOL,

% func1, func2,func3,func4,func5,func6,

% deriv1,deriv2,deriv3,deriv4,deriv5 and deriv6.

%

% OPTION:

% The final mollification of the

% finite difference approximation for the

% derivative can be eliminated. Activate the

% line %molda=derivfunc and comment the line

% [molda,delta]=molgcv1(derivfunc,n);.

% -----

%

ex=input('Choose one example from 1 to 7: ')

if ex~=1 & ex~=2 & ex~=3 & ex~=4 & ex~=5 & ex~=6 & ex~=7,
stop;end

epsil=input('Please enter amount of noise, epsilon, between
0 and 0.1: ')

if(epsil<0 | epsil>0.1), stop; end
```

```
clc

tic

n=128;

if (ex==1) [func]=func1(n); [der]=deriv1(n);else
if (ex==2) [func]=func2(n); [der]=deriv2(n);else
if (ex==3) [func]=func3(n); [der]=deriv3(n);else
if (ex==4) [func]=func4(n); [der]=deriv4(n);else
if (ex==5) [func]=func5(n); [der]=deriv5(n);else
if (ex==6) [func]=func6(n); [der]=deriv6(n);else
if (ex==7) [func]=func7(n); [der]=deriv7(n);end
end
end
end
end
end
end

iflag=ex;

%

fprintf('DERIVATIVE EXAMPLE No. %4.0f\n',iflag)

fprintf('Number of points %4.0f\n',n+1)

fprintf('Maximum data noise %12.5f\n',epsil)

fprintf('-----\n')

%
```

```

c=clock;

k=c(2)*c(4)*c(5)*c(6);

rand('seed',k)

for k=1:n+1

noise(k)=(2*rand(1)-1)*epsil;

end

da=func+noise;

%-----

%

% FIRST AND ONLY CALL

%

%-----

[molda,delta]=molgcv1(da,n);

%

l2f=sqrt(((func-molda)*(func-molda)')/n);

l2ff=sqrt((func*func')/n);

if(l2ff<1)

fprintf('l2-error mollified data function %12.5f\n', l2f)

fprintf('Radius of mollification %12.5f\n', delta)

fprintf('-----\n')

else

fprintf('l2-error mollified data function %12.5f\n', l2f)

```

```
fprintf('Relative error %12.5f\n', l2f/l2ff)

fprintf('Radius of mollification %12.5f\n', delta)

fprintf('-----\n')

end

x=0:1/n:1;

figure

plot(x,da,'--',x,func)

xlabel('Time');ylabel('Noisy and Exact Input Functions')

figure

plot(x,da,'--',x,molda)

xlabel('Time');ylabel('Noisy and Mollified Input
Functions')

figure

plot(x,da-molda)

xlabel('Time');ylabel('Input Error, Noisy-Mollified')

figure

plot(x,func-molda)

xlabel('Time');ylabel('Input Error, Exact-Mollified')

figure

plot(x,da-func)

xlabel('Time');ylabel('Input Error, Noisy-Exact')

%

% Centered Differences
```

```

%

for i=1:n-1

vfunc1(i)=n*(molda(i+2)-molda(i))/2;

end

dfirst=2*vfunc1(1)-vfunc1(2);

dlast=2*vfunc1(n-1)-vfunc1(n-2);

derivfunc=[dfirst,vfunc1,dlast];

% -----

% OPTION

%molda=derivfunc;

[molda,delta]=molgcv1(derivfunc,n);

% -----

l2d=sqrt(((der-molda)*(der-molda)')/n);

l2dd=sqrt((der*der')/n);

if(l2dd<1)

fprintf('l2-error mollified derivative function %12.5f\n',
l2d)

fprintf('Radius of mollification %12.5f\n', delta)

fprintf('-----\n')

else

fprintf('l2-error mollified derivative function %12.5f\n',
l2d)

fprintf('Relative error %12.5f\n', l2d/l2dd)

fprintf('Radius of mollification %12.5f\n', delta)

```

```
fprintf('-----\n')

end

figure

plot(x,der,'--',x,molda)

xlabel('Time');ylabel('Mollified and Exact Derivative Functions')

figure

plot(x,der-molda)

xlabel('Time');ylabel('Derivative Error')

toc
```

```
function[molda,delta]=molgcv1(da,n)

%

% This function is called by der1.m

% and it calls mol.m and moll.m

%

% INPUT:

% Data vector da of dimension n+1.

% Number of points minus one, n.

%

% OUTPUT:

% Mollified data vector, molda.

% Radius of mollification, delta.

%
```

```
% OPTION:

% The values of the GCV functional
% and/or the number of iterations
% can be activated or not. Simply
% comment (or not) the last two lines.
%

tol=0.001;

deltamin=0.001;

deltamax=0.10;

delta=0.04;

maxniter=30;

gr=0.5*(sqrt(5.0)-1);

cc=1-gr;

x0=deltamin;

x3=deltamax;

if(abs(deltamax-delta)>abs(delta-deltamin))

x1=delta;

x2=delta+cc*(deltamax-delta);

else

x2=delta;

x1=delta-cc*(delta-deltamin);

end

%-----
```



```
% CALLS ARE HERE

%

[ iwtmax,wt ]=mol (x1,n) ;

%

[molda,gcv]=mol1
( iwtmax,n,wt,x1,da ) ;

%

f1=gcv;

[ iwtmax,wt ]=mol (x2,n) ;

[molda,gcv]=mol1( iwtmax,n,wt,x2,da ) ;

f2=gcv;

%-----

counter=0;

test1=abs(x3-x0);

test2=tol*(abs(x1)+abs(x2));

while((test1>test2)&(counter<maxniter))

counter=counter+1;

if(f2<f1)

x0=x1;

x1=x2;

x2=gr*x1+cc*x3;

f1=f2;
```

```
[iwtmax,wt]=mol(x2,n);  
[molda,gcv]=mol1(iwtmax,n,wt,x2,da);  
f2=gcv;  
else  
x3=x2;  
x2=x1;  
x1=gr*x2+cc*x0;  
f2=f1;  
[iwtmax,wt]=mol(x1,n);  
[molda,gcv]=mol1(iwtmax,n,wt,x1,da);  
f1=gcv;  
end  
test1=abs(x3-x0);  
test2=tol*(abs(x1)+abs(x2));  
end  
if(f1<f2)  
golden=f1;  
minx=x1;  
else  
golden=f2;  
minx=x2;  
end  
delta=minx;
```

```

[iwtmax,wt]=mol(delta,n);

[molda,gcv]=mol1(iwtmax,n,wt,delta,da);

% -----

% OPTION

fprintf('Number of iterations: %4.0f\n',counter)

fprintf('GCV functional: %12.5e\n',gcv)

% -----



---



function [iwtmax,wt]=mol(delta,n)

%

% This function is called by molgcv1.m

%

% INPUT:

% Radius of mollification, delta.

% Number of points minus one, n.

%

% OUTPUT:

% Length (in step size units) of numerical

% "support" of Gaussian kernel, iwtmax.

% Gaussian kernel, wt.

%

a=1/(n*delta*sqrt(pi));

m=round(3*delta*n)+3;

```

```
iwtmax=2*m;

x=-m/n+1/n:1/n:m/n;

wt=a*exp(-x.^2/(delta*delta));

%plot(x,wt)



---



function[molda,gcv]=moll(iwtmax,n,wt,delta,da)

%

% This function is called by molgcv1.m.

%

% INPUT:

% Length (in step size units) of numerical

% support of Gaussian kernel, iwtmax.

% Number of points minus one, n.

% Gaussian kernel, wt.

% Radius of mollification, delta.

% Data vector to be mollified, da.

%

% OUTPUT:

% Generalized Cross Validation functional value, gcv.

% Mollified data vector, molda.

%

% Calculation of the constants ca and caa.

sn1=0;
```

```
sn2=0;

sd=0;

itmin=iwtmax/2;

for i=1:iwtmax-1

dr(i)=da(2+n-i);

end

for i=1:itmin

sk=0;

for k=itmin+i:iwtmax

sk=sk+wt(k);

end

sj1=0;

sj2=0;

for j=1:itmin+i-1

sj1=sj1+wt(j)*da(itmin+i-j);

sj2=sj2+wt(j)*dr(itmin+i-j);

end

sn1=sn1+(da(i)-sj1)*sk;

sn2=sn2+(dr(i)-sj2)*sk;

sd=sd+sk*sk;

end

ca=sn1/sd;

caa=sn2/sd;
```

```
for i=1:itmin-1
one(i)=ca;
three(i)=caa;
end

big=[one,da,three];
wtt=wt(1:(iwtmax-1));
c=conv(big,wtt);

for i=1:n+1
molda(i)=c(iwtmax-2+i);
end

gcv=2*((da-molda)*(da-molda)')/iwtmax;
```
