Automatic Differentiation using MATLAB OOP

Richard D. Neidinger
Davidson College

MOSAIC Kick-Off Workshop, IMA, MN
Summer 2010

Reconsider basic questions in Calculus and in Programming:

- How are derivatives calculated?
- What is object vs. procedure oriented programming?
- What is a function to a computer?

Alternatives for computing derivatives:

- Slope for numerical approximation.
- Rules for symbolic expression.
- Rules for numerical values!

If $h(x) = u(x) * v(x)$, then

symbols $h'(x) = u'(x) * v(x) + u(x) * v'(x)$

values $h'(a) = u'(a) * v(a) + u(a) * v'(a)$

Programming the product rule:

$$\text{Combine } [u(a), u'(a)] \text{ and } [v(a), v'(a)]$$

$$\text{and a "times operation" to make}$$

$$[h(a), h'(a)] = [u(a) * v(a), u'(a) * v(a) + u(a) * v'(a)].$$

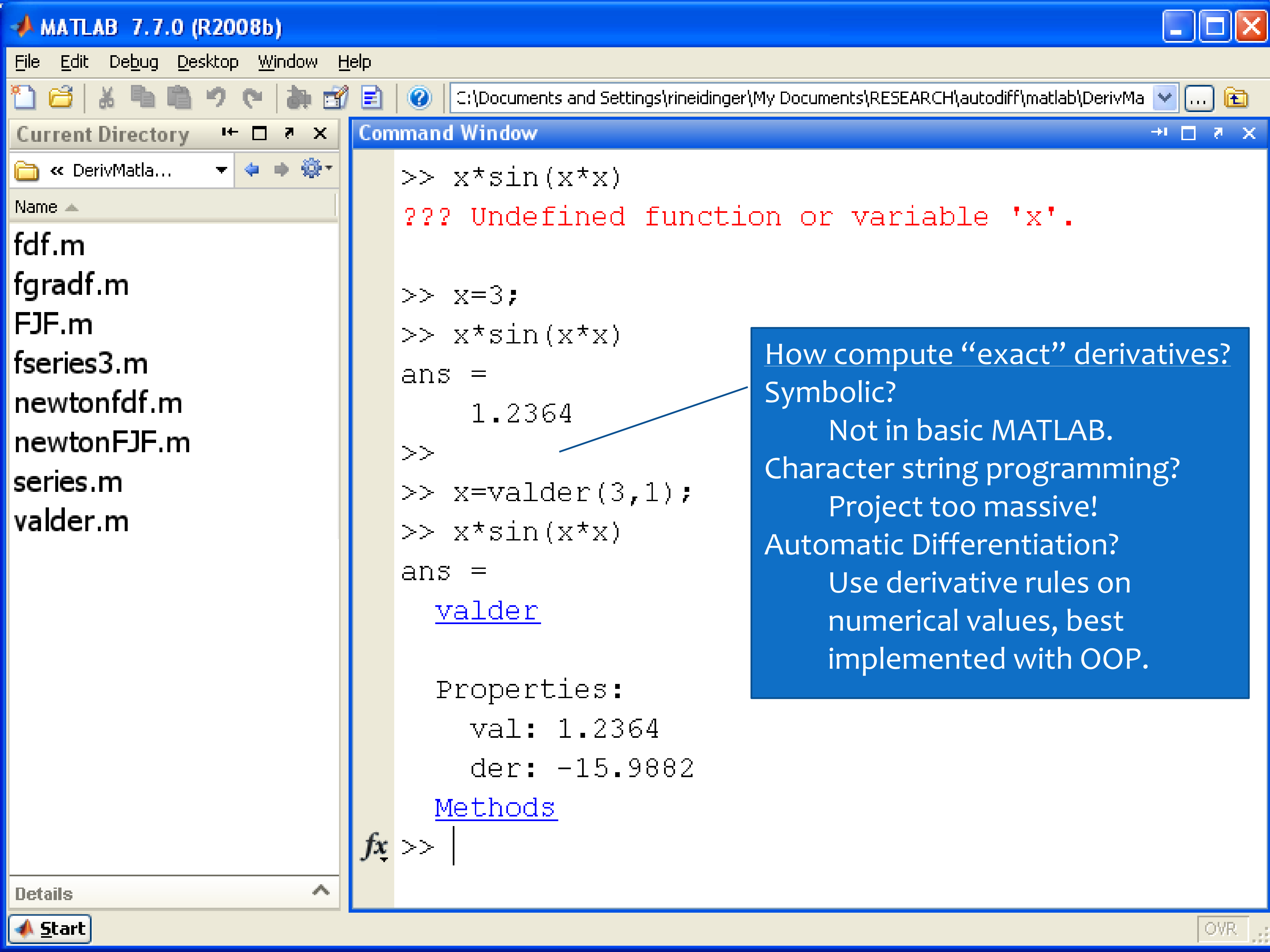
- <u>Procedural approach</u>: make up procedure adtimes on native data type (array of two doubles).
- <u>OOP approach</u>: make up an object class and overload native operations as methods on such objects.

Suggest AD in any course with computing environment:

- Designed for numerical computation.
- Allows operator overloading, preferably in OOP.
- Computing derivatives is desirable.

MATLAB* is ideal!
AD topic in my Numerical Analysis.

*Release 2008a or later



MATLAB 7.7.0 (R2008b)
File Edit Debug Desktop Window Help
C:\Documents and Settings\rineidinger\My Documents\RESEARCH\autodiff\matlab\DerivMa
Current Directory
« DerivMatla...
Name
fdf.m
fgradf.m
FJF.m
fseries3.m
newtonfdf.m
newtonFJF.m
series.m
valder.m
Details
Command Window
>> x*sin(x*x)
??? Undefined function or variable 'x'.
>> x=3;
>> x*sin(x*x)
ans =
1.2364
>>
>> x=valder(3,1);
>> x*sin(x*x)
ans =
valder
Properties:
val: 1.2364
der: -15.9882
Methods
>>
How compute "exact" derivatives?
Symbolic?
Not in basic MATLAB.
Character string programming?
Project too massive!
Automatic Differentiation?
Use derivative rules on
numerical values, best
implemented with OOP.
Start
OVR

C:\Documents and Settings\rineidinger\My Documents\RESEARCH\autodiff\matlab\Deriv&MatlabMAASE\valder.m*

File Edit Text Go Cell Tools Debug Desktop Window Help

Stack: Base

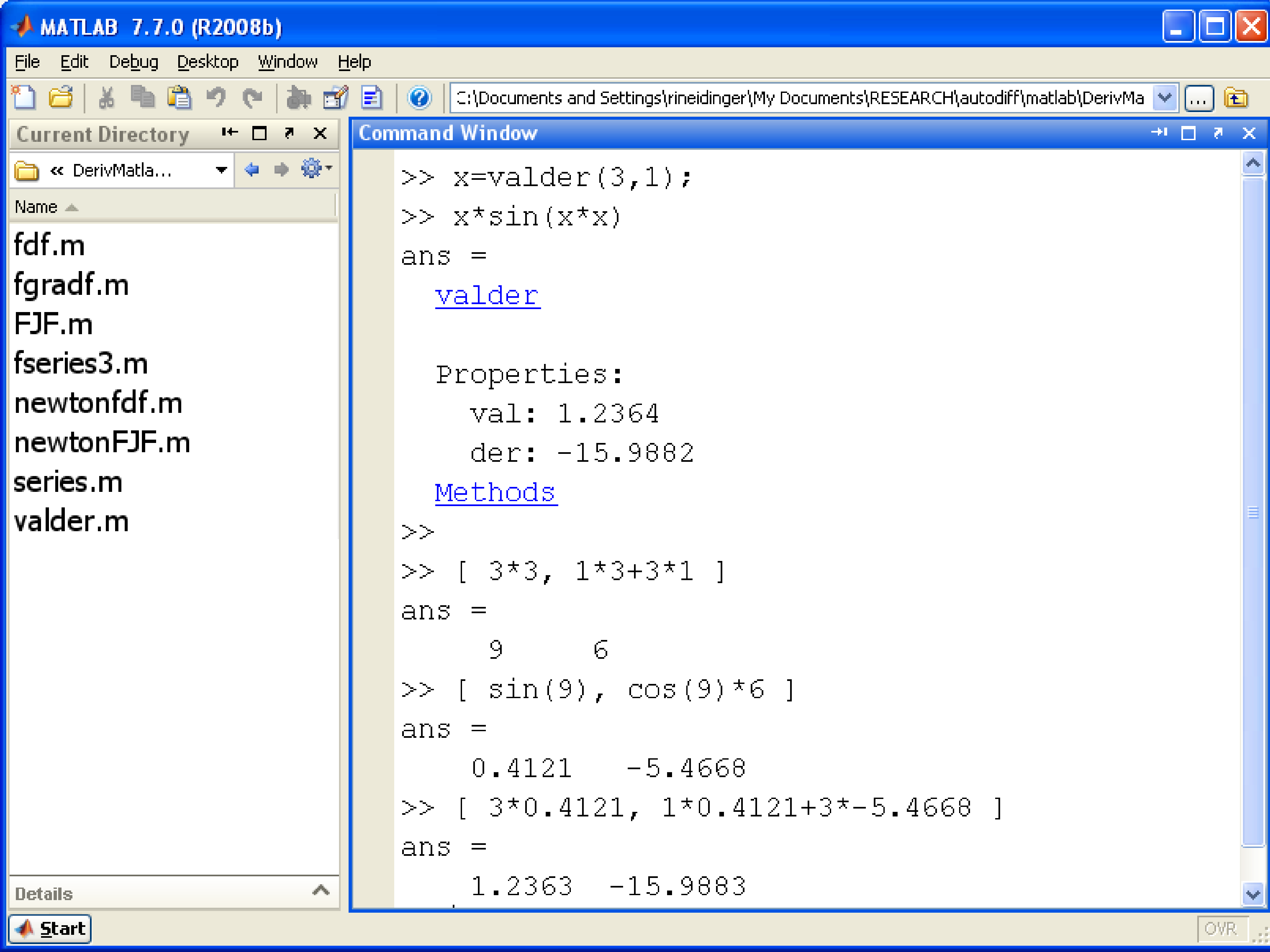
```
classdef valder
    % VALDER class implements Automatic Differentiation by operator overloading
    properties
        val  %function value
        der  %derivative value or gradient vector
    end
    methods
        function obj = valder(a,b)...
        function vec = double(obj)...
        function h = plus(u,v)...
        function h = uminus(u)...
        function h = minus(u,v)...
        function h = mtimes(u,v)...
        function h = mrdivide(u,v)...
        function h = mpower(u,v)...
        function h = exp(u)...
        function h = log(u)...
        function h = sqrt(u)...
        function h = sin(u)...
        function h = cos(u)...
        function h = tan(u)...
        function h = asin(u)...
        function h = atan(u)...
    end
end
```

```
function obj = valder(a,b)
   %VALDER class constructor; only the bottom case is needed.
   if nargin == 0 %never intended for use.
      obj.val = [];
      obj.der = [];
   elseif nargin == 1 %c=valder(a) for constant w/ derivative 0.
      obj.val = a;
      obj.der = 0;
   else
      obj.val = a; %given function value
      obj.der = b; %given derivative value or gradient vector
   end
end
```

```
function vec = double(obj)
   %VALDER/DOUBLE Convert valder object to vector of doubles.
   vec = [ obj.val, obj.der ];
end
```

```
function h = sin(u)
    %VALDER/SIN overloads sine with a valder object argument
    h = valder(sin(u.val), cos(u.val)*u.der);
end
```

```
function h = mtimes(u,v)
   %VALDER/MTIMES overloads * for at least one valder object argument
   if ~isa(u,'valder') %u is a scalar
      h = valder(u*v.val, u*v.der);
   elseif ~isa(v,'valder') %v is a scalar
      h = valder(v*u.val, v*u.der);
   else
      h = valder(u.val*v.val, u.der*v.val + u.val*v.der);
   end
end
```

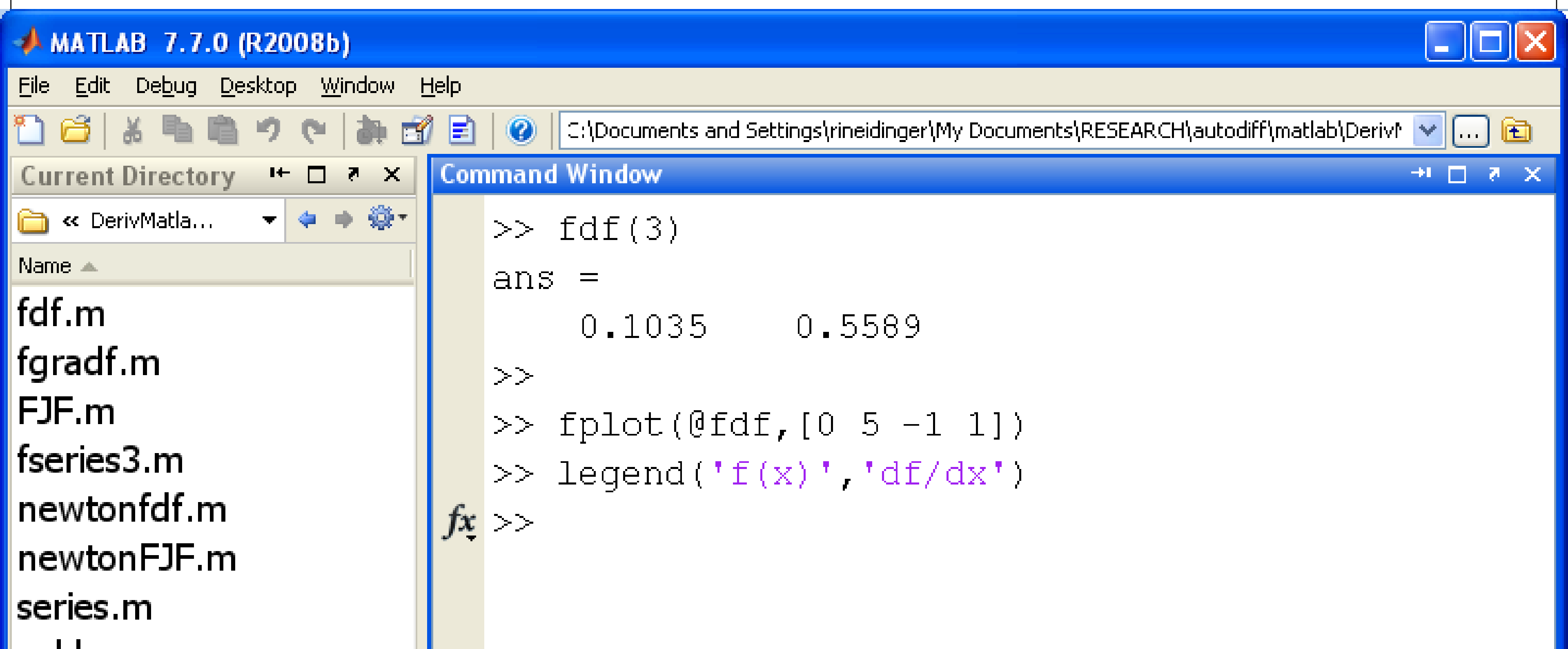

MATLAB 7.7.0 (R2008b)
File Edit Debug Desktop Window Help
C:\Documents and Settings\rineidinger\My Documents\RESEARCH\autodiff\matlab\DerivMa
Current Directory
« DerivMatla...
Name
fdf.m
fgradf.m
FJF.m
fseries3.m
newtonfdf.m
newtonFJF.m
series.m
valder.m
Details
Command Window
>> x=valder(3,1);
>> x*sin(x*x)
ans =
valder
Properties:
val: 1.2364
der: -15.9882
Methods
>>
>> [3*3, 1*3+3*1]
ans =
9 6
>> [sin(9), cos(9)*6]
ans =
0.4121 -5.4668
>> [3*0.4121, 1*0.4121+3*-5.4668]
ans =
1.2363 -15.9883
Start
OVR

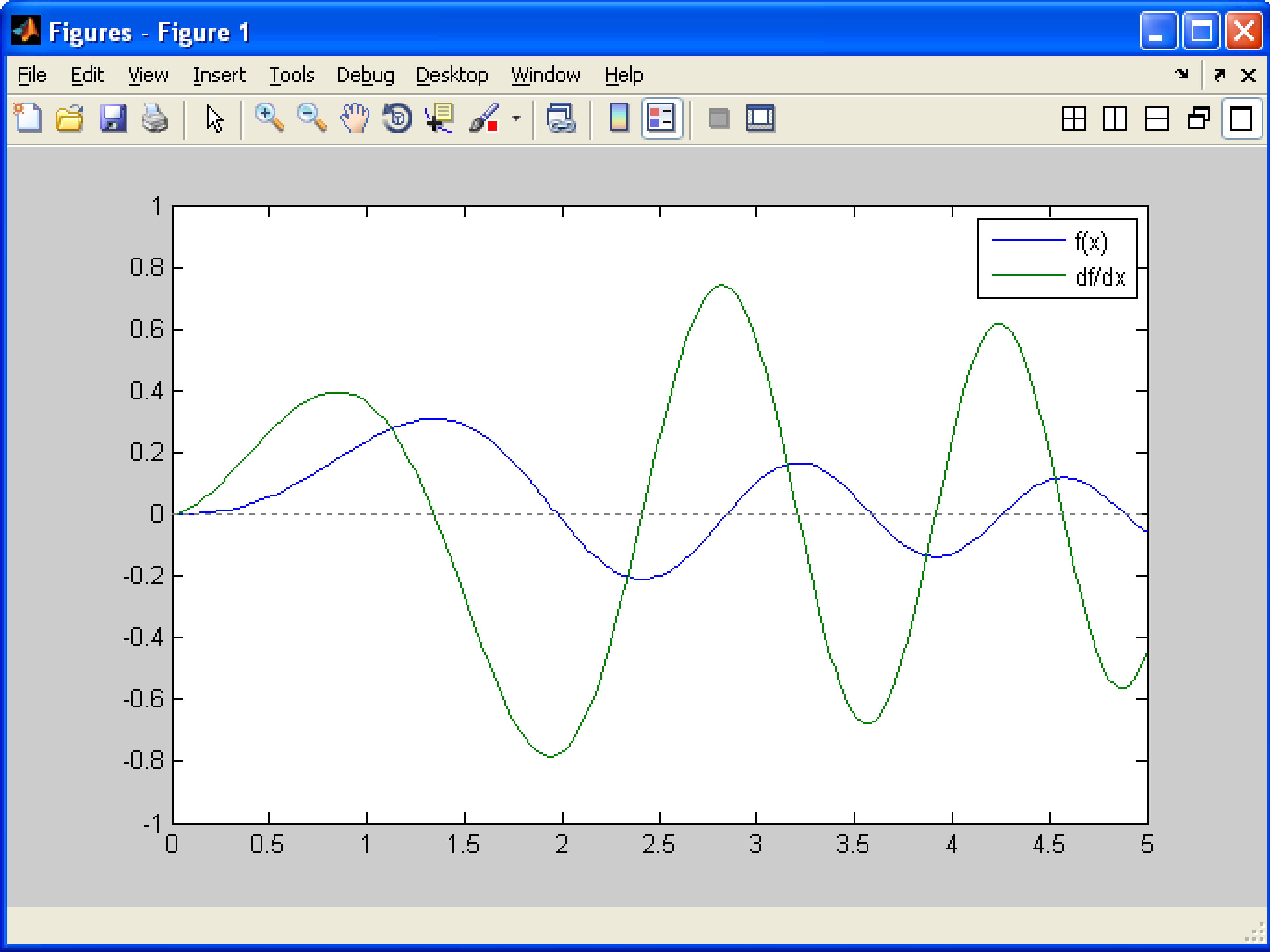
Application to One-variable Functions and Newton's Method

- Just returns the derivative value at one point?
- But that is exactly what a function does!

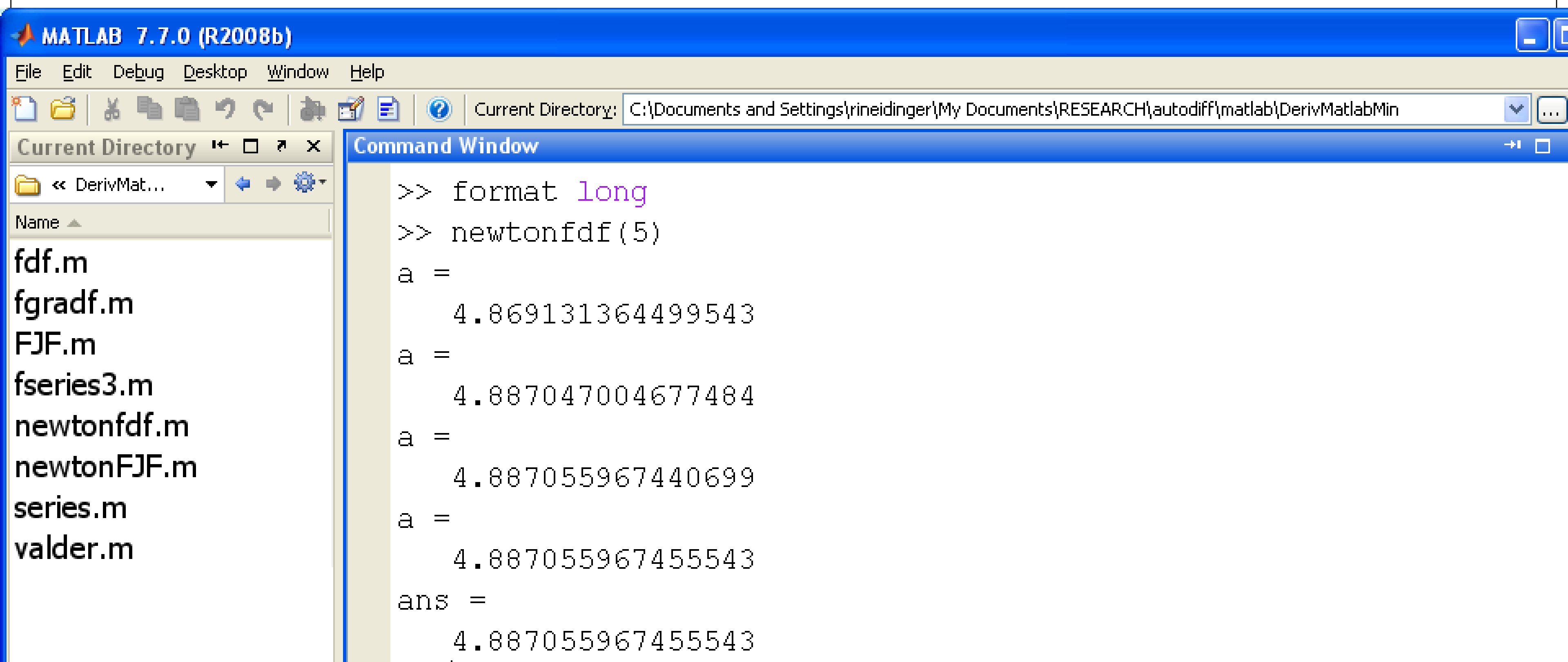
$$f(x) = e^{-\sqrt{x}} \sin(x \ln(1+x^2))$$

```
function vec = fdf(a)
%FDF takes a scalar and returns the double vector [ f(a), f'(a) ]
%   where f is defined in normal syntax below.
x = valder(a,1);
y = exp(-sqrt(x))*sin(x*log(1+x^2));
vec = double(y);
```




Figures - Figure 1
File Edit View Insert Tools Debug Desktop Window Help
f(x)
df/dx
1
0.8
0.6
0.4
0.2
0
-0.2
-0.4
-0.6
-0.8
-1
0 0.5 1 1.5 2 2.5 3 3.5 4 4.5 5

```
function root = newtonfdf(a)
%NEWTON seeks a zero of the function defined in fdf using the initial a
% root estimate and Newton's method (with no exception protections).
% fdf uses @valder to return a vector of function and derivative values.
delta = 1;
while abs(delta) > .000001
    fvec = fdf(a);
    delta = fvec(1)/fvec(2); %value/derivative
    a = a - delta
end
root = a;
```

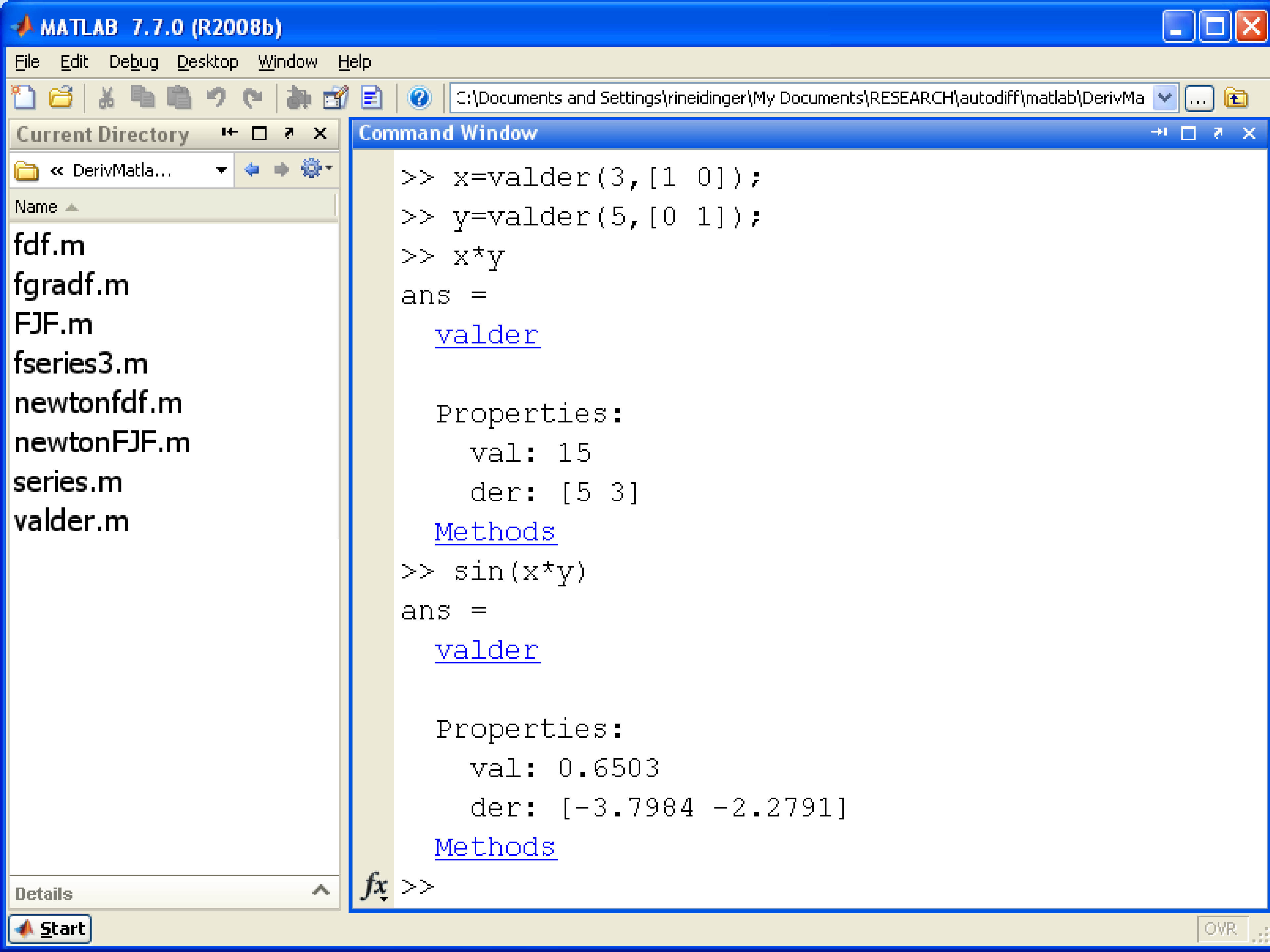


Application to Multivariable Gradients

- All derivative rules generalize with gradient in place of derivative!
- If $h(x,y,z) = \sin(u(x,y,z))$

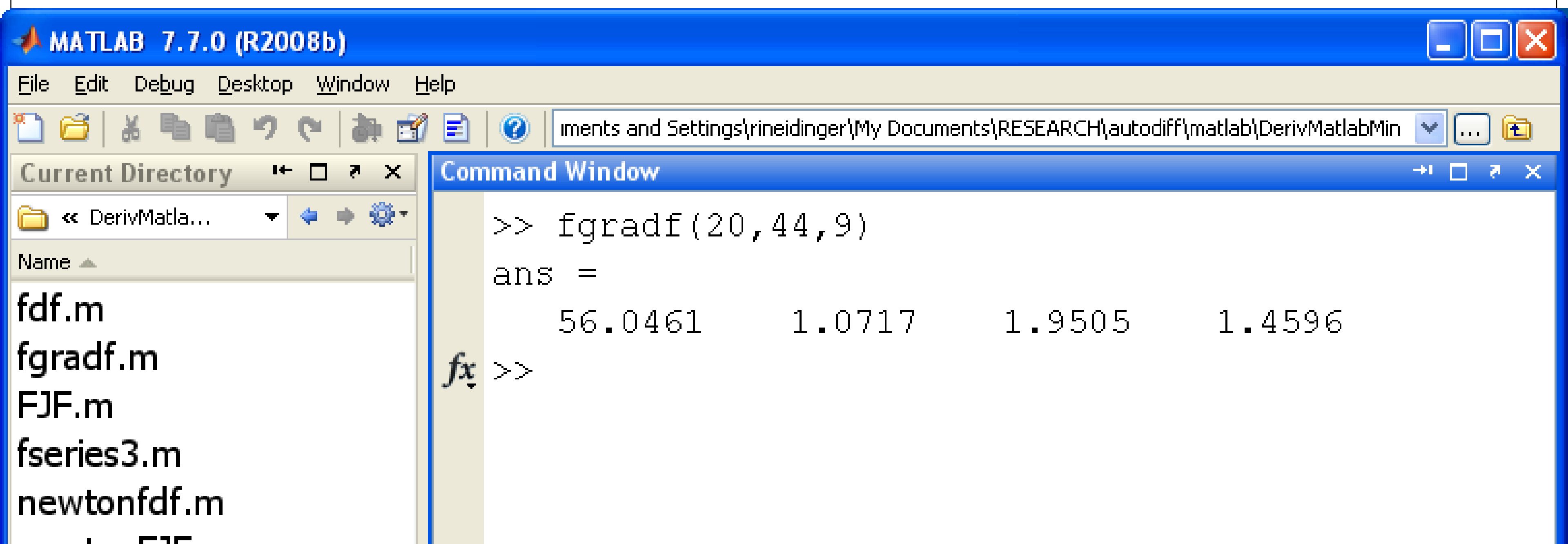
  then $[h_x , h_y , h_z] = \cos(u(x,y,z))*[u_x , u_y , u_z]$
- If $h(x,y) = u(x,y)*v(x,y)$

  then $[h_x , h_y] = [u_x , u_y]*v + u* [v_x , v_y]$
- Thus, the valder class works with gradients as the der property!

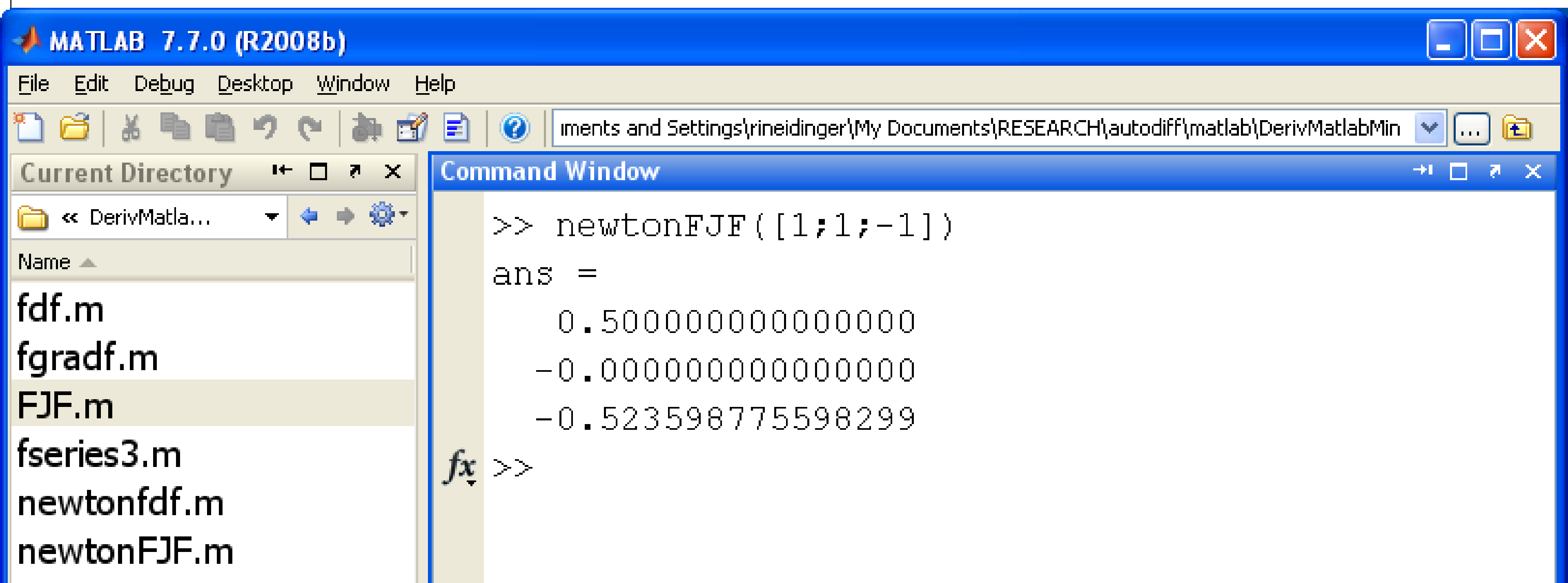


MATLAB 7.7.0 (R2008b)
File Edit Debug Desktop Window Help
C:\Documents and Settings\rineidinger\My Documents\RESEARCH\autodiff\matlab\DerivMa
Current Directory
« DerivMatla...
Name
fdf.m
fgradf.m
FJF.m
fseries3.m
newtonfdf.m
newtonFJF.m
series.m
valder.m
Details
Command Window
>> x=valder(3,[1 0]);
>> y=valder(5,[0 1]);
>> x*y
ans =
valder
Properties:
val: 15
der: [5 3]
Methods
>> sin(x*y)
ans =
valder
Properties:
val: 0.6503
der: [-3.7984 -2.2791]
Methods
>>
Start
OVR

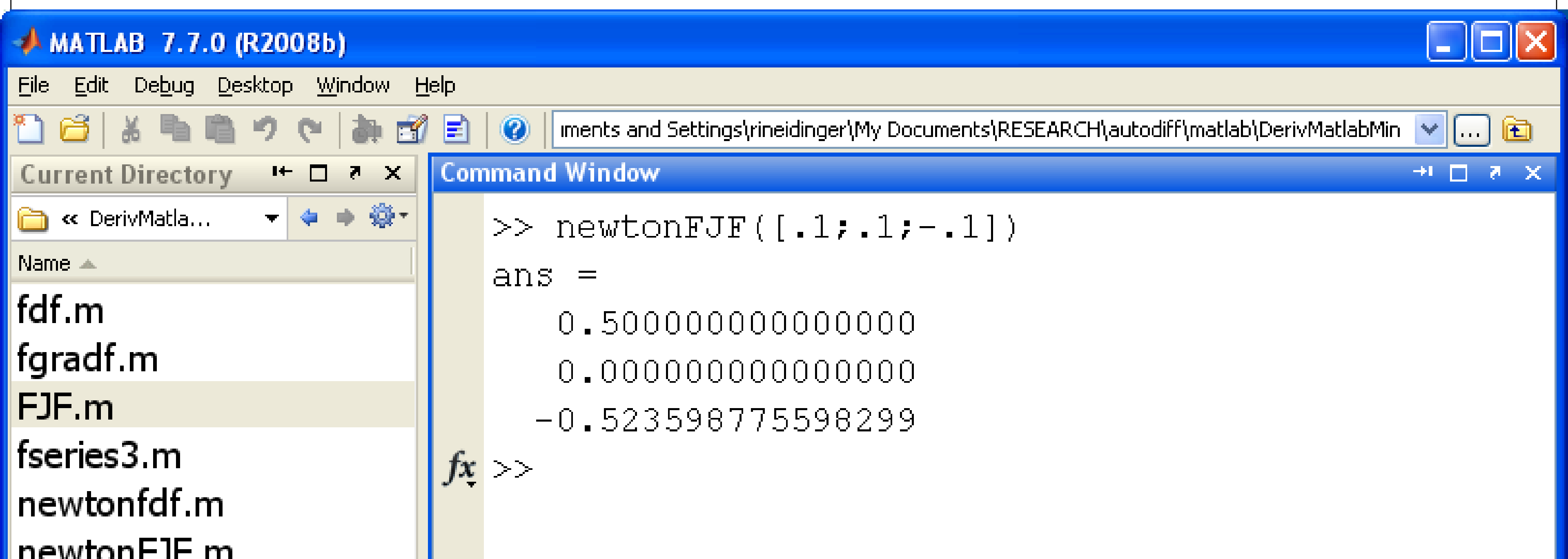
```
function vec = fgradf(a0,v0,h0)
%FGRADF computes tennis serve range and sensitivites to parameters
%   input is angle a, velocity v, and height h of serve
%   output is horiztonal range f, df/da, df/dv, df/dh
a = valder(a0,[1 0 0]); %angle in degrees
v = valder(v0,[0 1 0]); %velocity in ft/sec
h = valder(h0,[0 0 1]); %height in ft
rad = a*pi/180;
tana = tan(rad);
vhor = (v*cos(rad))^2;
f = (vhor/32)*(tana + sqrt(tana^2+64*h/vhor)); %horizontal range
vec = double(f);
```



```
function root = newtonFJF(A)
%NEWTONFJF seeks a zero of the function defined in FJF using the initial A
% root estimate and Newton's method (with no exception protections).
% FJF returns the value and Jacobian of a function F:R^n->R^n where
% A is a nx1 matrix input, F is nx1 matrix output and J is nxn Jacobian.
delta = 1;
while max(abs(delta)) > .000001
    [F,J] = FJF(A);
    delta = J\F;   % solves the linear system JX=F for X
    A = A - delta;
end
root = A;
```



```matlab
function [F, J] = FJF(A)
%FJF returns the value and Jacobian of a function F:R^3->R^3.
% A is a 3x1 matrix input, F is 3x1 matrix output and J is 3x3 Jacobian.
x = valder(A(1),[1 0 0]);
y = valder(A(2),[0 1 0]);
z = valder(A(3),[0 0 1]);
f1 = 3*x-cos(y*z)-1/2;
f2 = x^2 -81*(y+0.1)^2+sin(z)+1.06;
f3 = exp(-x*y)+20*z+(10*pi-3)/3;
values = [double(f1); double(f2); double(f3)];
F = values(:,1);
J = values(:,2:4);
```



Higher-Order AD and Taylor Series

- Instead of carrying just the first derivative, carry a vector of Taylor series coefficients.
- Algorithms for operations on series are the methods for "derivative rules."
- For $u(x) = u_0 + u_1(x-a) + u_2(x-a)^2 + \cdots$

  series object u will have properties:

  val: u_0

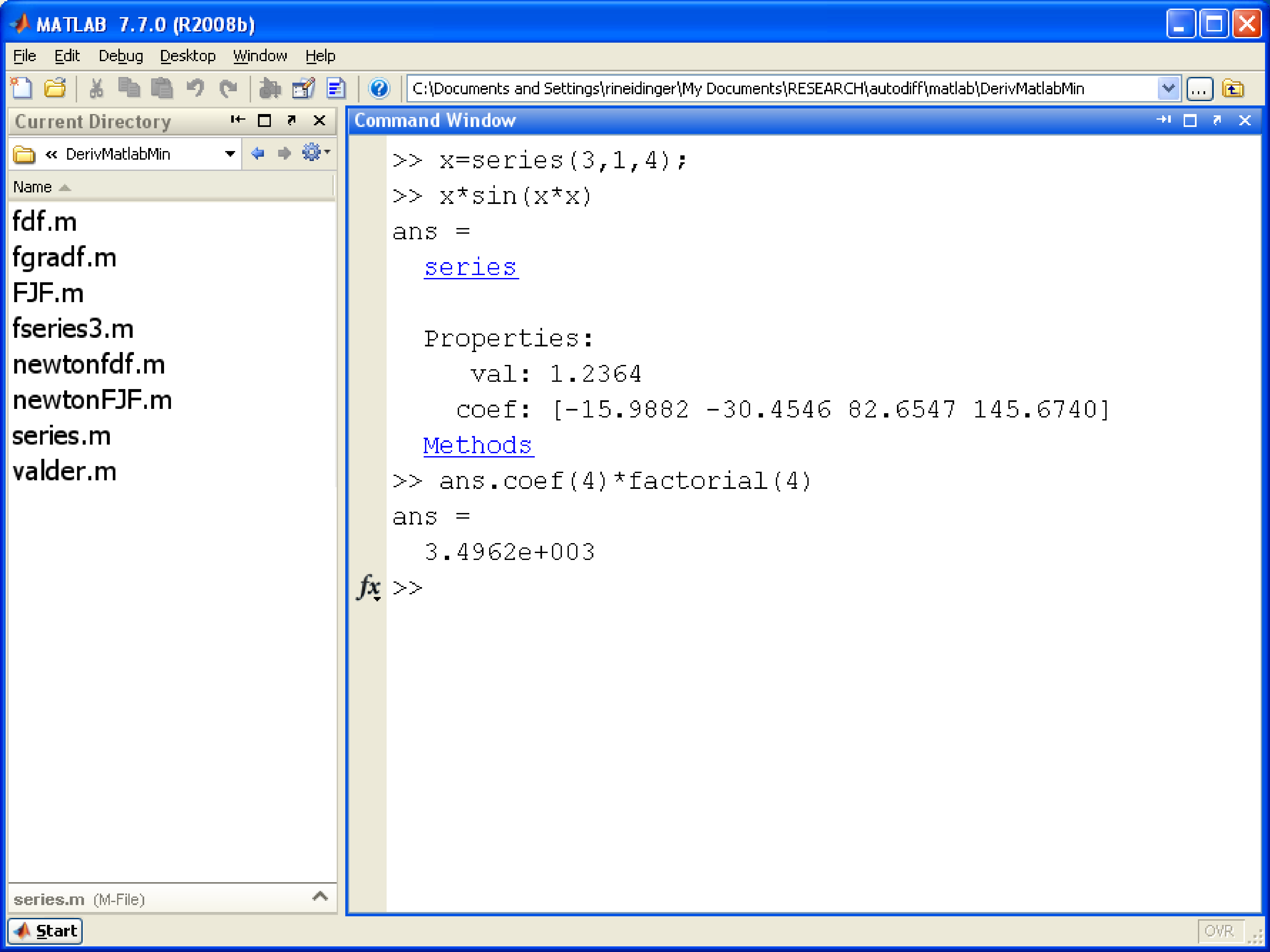
  coef: $[u_1, u_2, \ldots, u_n]$

C:\Documents and Settings\rineidinger\My Documents\RESEARCH\autodiff\matlab\DerivMatlabMin\series.m

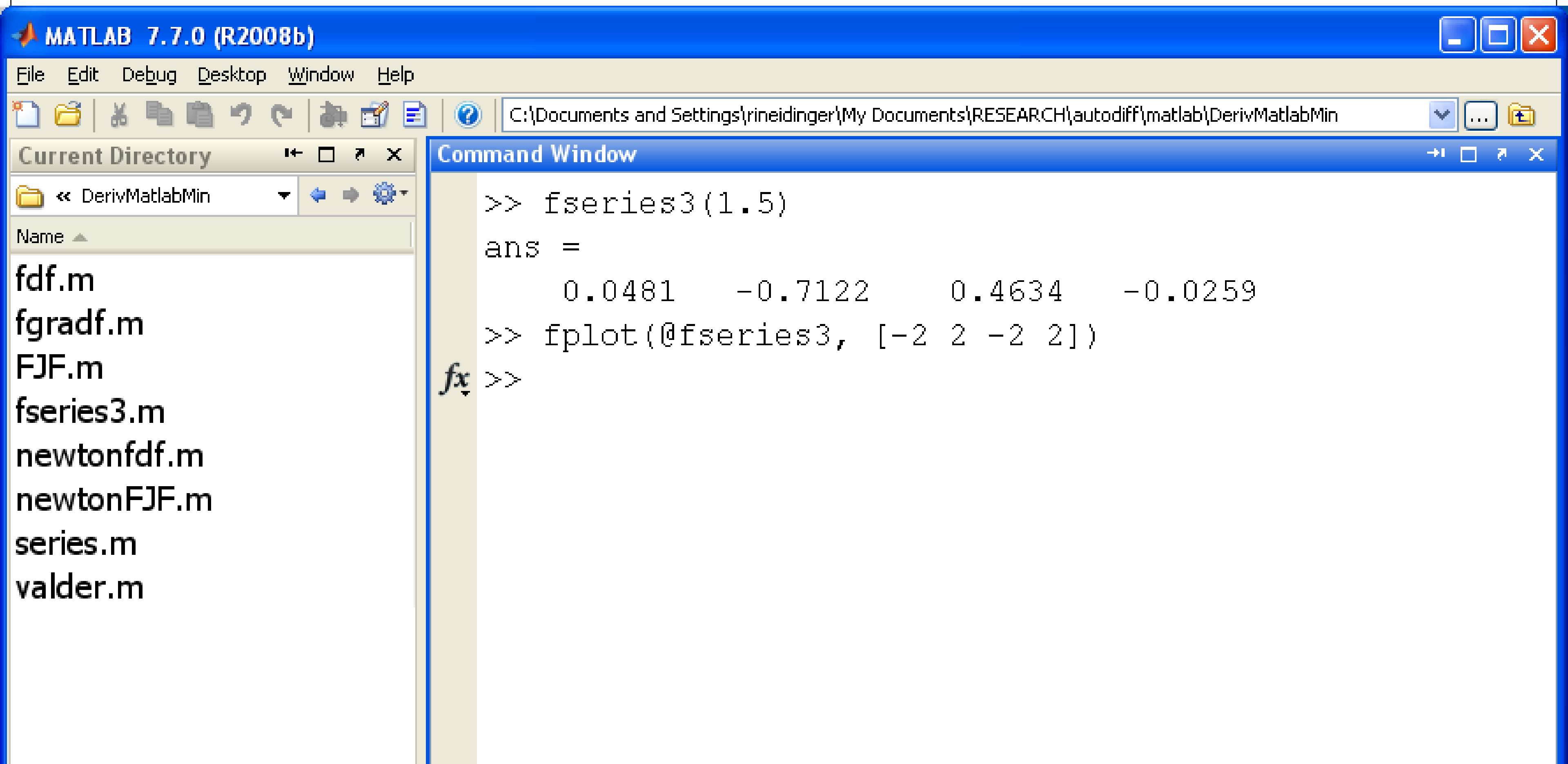
ile Edit Text Go Cell Tools Debug Desktop Window Help

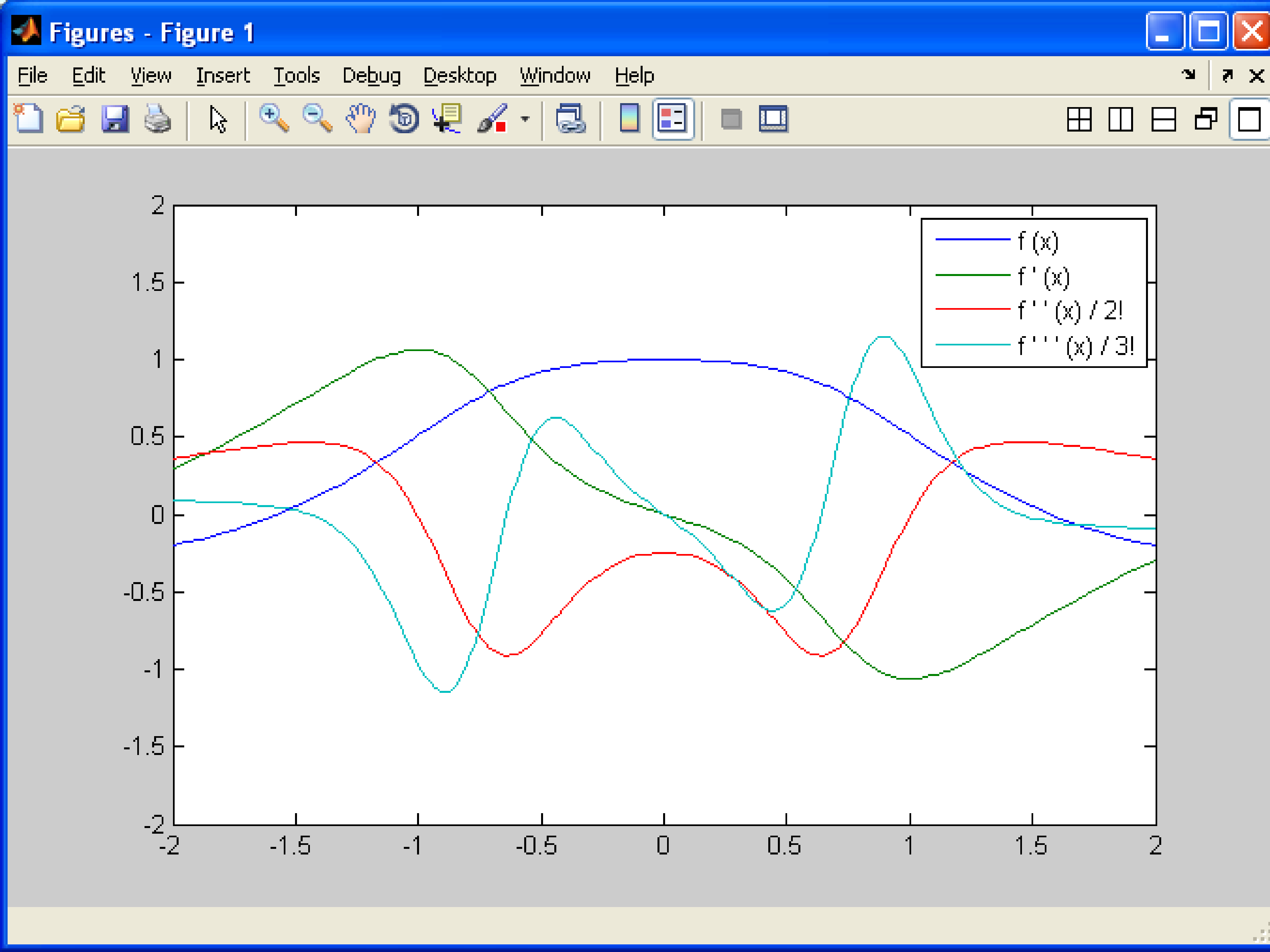
Stack: Base fx

```
classdef series
    % SERIES class implementing AD to compute series coefficients.%...%
    properties
        val  %function value (constant term)
        coef %vector of Taylor coefficients, linear to highest term
    end
    methods
        function obj = series(a,der,order)...
        function vector = double(obj)...
        function h = plus(u,v)...
        function h = uminus(u)...
        function h = minus(u,v)...
        function h = mtimes(u,v)...
        function h = mrdivide(u,v)...
        function h = sqrt(u)...
        function h = exp(u)...
        function h = log(u)...
        function h = mpower(u,r)...
        function [s, c] = sincos(u)...
        function h = sin(u)...
        function g = cos(u)...
        function h = tan(u)...
        function h = asin(u)...
        function h = atan(u)...
    end
end
```


MATLAB 7.7.0 (R2008b)
File Edit Debug Desktop Window Help
C:\Documents and Settings\rineidinger\My Documents\RESEARCH\autodiff\matlab\DerivMatlabMin
Current Directory
« DerivMatlabMin
Name
fdf.m
fgradf.m
FJF.m
fseries3.m
newtonfdf.m
newtonFJF.m
series.m
valder.m
series.m (M-File)
Command Window
>> x=series(3,1,4);
>> x*sin(x*x)
ans =
series
Properties:
val: 1.2364
coef: [-15.9882 -30.4546 82.6547 145.6740]
Methods
>> ans.coef(4)*factorial(4)
ans =
3.4962e+003
>>
Start
OVR

```
function vec = fseries3(a)
%FSERIES returns a vector of Taylor coefficients about a to order 3.
%   f is defined in normal syntax below.
x = series(a,1,3);
y = cos(x)*sqrt(exp(-x*atan(x/2)+log(1+x^2)/(1+x^4)));
vec = double(y);
```




Figures - Figure 1
File Edit View Insert Tools Debug Desktop Window Help
f (x)
f ' (x)
f ' ' (x) / 2!
f ' ' ' (x) / 3!
2
1.5
1
0.5
0
-0.5
-1
-1.5
-2
-2 -1.5 -1 -0.5 0 0.5 1 1.5 2

Materials available:

- "Introduction to Automatic Differentiation and MATLAB Object-Oriented Programming" to appear in *SIAM Review*, 52(3), Sept. 2010.
- MATLAB M-files and preprint available at: www.davidson.edu/math/neidinger/publicat.html

Overloading Other Operations

MATLAB Operators and Associated Functions

The following table lists the function names for common MATLAB operators.

Operation	Method to Define	Description
`a + b`	`plus(a,b)`	Binary addition
`a - b`	`minus(a,b)`	Binary subtraction
`-a`	`uminus(a)`	Unary minus
`+a`	`uplus(a)`	Unary plus
`a.*b`	`times(a,b)`	Element-wise multiplication
`a*b`	`mtimes(a,b)`	Matrix multiplication
`a./b`	`rdivide(a,b)`	Right element-wise division
`a.\b`	`ldivide(a,b)`	Left element-wise division
`a/b`	`mrdivide(a,b)`	Matrix right division
`a\b`	`mldivide(a,b)`	Matrix left division
`a.^b`	`power(a,b)`	Element-wise power
`a^b`	`mpower(a,b)`	Matrix power

C:\Documents and Settings\rineidinger\My Documents\RESEARCH\autodiff\matlab\Deriv&MatlabMAASE\valder.m*

File Edit Text Go Cell Tools Debug Desktop Window Help

Stack: Base

```
classdef valder
    % VALDER class implements Automatic Differentiation by operator overloading
    properties
        val  %function value
        der  %derivative value or gradient vector
    end
    methods
        function obj = valder(a,b)...
        function vec = double(obj)...
        function h = plus(u,v)...
        function h = uminus(u)...
        function h = minus(u,v)...
        function h = mtimes(u,v)...
        function h = mrdivide(u,v)...
        function h = mpower(u,v)...
        function h = exp(u)...
        function h = log(u)...
        function h = sqrt(u)...
        function h = sin(u)...
        function h = cos(u)...
        function h = tan(u)...
        function h = asin(u)...
        function h = atan(u)...
    end
end
```

```
function h = exp(u)
   %VALDER/EXP overloads exp of a valder object argument
   h = valder(exp(u.val), exp(u.val)*u.der);
end
function h = log(u)
   %VALDER/LOG overloads natural logarithm of a valder object argument
   h = valder(log(u.val), (1/u.val)*u.der);
end
```

```
function h = mpower(u,v)
   %VALDER/MPOWER overloads ^ with at least one valder object argument
   if ~isa(u,'valder') %u is a scalar
      h = valder(u^v.val, u^v.val*log(u)*v.der);
   elseif ~isa(v,'valder') %v is a scalar
      h = valder(u.val^v, v*u.val^(v-1)*u.der);
   else
      h = exp(v*log(u)); %call overloaded log, * and exp
   end
end
```