Polynomial and interpolation functions

Polynomials.

In MATLAB polynomials are represented as a vector of coefficients with the first element corresponding to highest power of the polynomial. Example: \(x^4+3x^3-3x^2-11x-6\) corresponds to vector \([1, 3, -3, -11, -6]\).

Several functions can be used to deal with polynomials:

**roots**  
Find polynomial roots. Usage `roots(C)` where `C` is vector of polynomial coeffs. Example above (Note: all roots are real):

```matlab
r = roots([1, 3, -3, -11, -6])
r =
  2.0000
 -3.0000
 -1.0000 + 0.0000i
 -1.0000 - 0.0000i
```

**poly**  
For vector input construct polynomial with specified roots. Example: `poly([2, -3, -1, -1])` returns matrix of coefficients that are specified above. For square matrix input (N x N) returns row vector with (N+1) elements (coefficients of the characteristic polynomial `DET(lambda*EYE(A) - A)`). Example for random matrix:

```matlab
A = rand(3)
A =
  0.2190  0.6793  0.5194
  0.0470  0.9347  0.8310
  0.6789  0.3835  0.0346
poly(A)
ans =
  1.0000  -1.1882  -0.4587   0.0008
```

**polyval**  
Evaluate polynomial. Usage `polyval(C,x)`. Calculates value of the polynomial `C` at each point of `x`.

**polyfit**  
Fit polynomial to data. Usage `polyfit(x, y, n)` where `x` and `y` are data and `n` is the power of the polynomial. Returns vector of coefficients that can be used to calculate values of polynomial in specified points using `polyval`.
Example using \textit{polyval} and \textit{polyfit}:
\begin{verbatim}
x=-4:0.05:3;
y0=polyval(C,x);
y1=polyval(C,x)+10*rand(size(x))-5;
C1=polyfit(x,y1,4);
C2=polyfit(x,y1,3);
y2=polyval(C1,x); y3=polyval(C2,x);
plot(x,y0,'r',x,y1,'k.',x,y2,'b',x,y3,'g')
\end{verbatim}
% Original polynomial - red, fitted 4th power - blue; 3rd power - green.

\[ [p,S] = \text{polyfit}(x,y,n) \] returns the polynomial coefficients \( p \) and a structure \( S \) for use with \textit{polyval} to obtain error estimates or predictions. If the errors in the data \( y \) are independent normal with constant variance, \textit{polyval} produces error bounds that contain at least 50\% of the predictions. \[ [y,\delta y] = \text{polyval}(p,x,S); \]
Example:
\begin{verbatim}
>> x=linspace(0,10,100);
>> y=erfc(x/5)+randn(size(x))/50;
>> [cf,s] = polyfit(x,y,1);
>> [y1,delta1] = polyval(cf,x,s);
>> [cf,s] = polyfit(x,y,3);
>> [y2,delta2] = polyval(cf,x,s);
>> plot(x,y,'r',x,y1,'b',x,y1-delta1,'m',x,y1+delta1,'m',x,y2,'g',x,y2-delta2,'c',x,y2+delta2,'c')
\end{verbatim}
polyder      Differentiate polynomial. polyder(C) returns the derivative of the polynomial whose coefficients are the elements of vector C.

polyint     Integrates polynomial analytically. polyint(p,k) returns a polynomial representing the integral of polynomial p, using a scalar constant of integration k. polyint(p) assumes a constant of integration k=0.

conv         Multiply polynomials. conv(A, B) convolves vectors A and B. If A and B are vectors of polynomial coefficients, convolving them is equivalent to multiplying the two polynomials.

deconv      Divide polynomials. See help.
Data interpolation.

**interp1**  
1-D interpolation (1-D table lookup). $YI = \text{interp1}(X,Y,XI)$ returns vector $YI$ containing elements corresponding to the elements of $XI$ and determined by interpolation within vectors $X$ and $Y$. Example:

```matlab
>> x=0:0.5:5;
>> y=x.^0.3;
>> x1=linspace(0,5,200);
>> y1=interp1(x,y,x1,'linear');
>> y2=interp1(x,y,x1,'cubic');
>> y3=interp1(x,y,x1,'spline');
>> yy=x1.^0.3;
>> plot(x,y,'bd',x1,y1,'m',x1,y2,'r',x1,y3,'g',x1,yy,'k')
```

For more information see help.

**interp2**  
2-D interpolation (2-D table lookup). $ZI = \text{interp2}(X,Y,Z,XI,YI)$ returns matrix $ZI$ containing elements corresponding to the elements of $XI$ and $YI$ and determined by interpolation within the 2-D function described by matrices $X$, $Y$, and $Z$. See help.
interpft  1-D interpolation using FFT method. See help.

griddata  Data gridding. ZI = griddata(X,Y,Z,XI,YI) returns matrix ZI containing elements corresponding to the elements of matrices XI and YI and determined by interpolation within the 2-D function described by the (usually) non-uniformly-spaced vectors (X,Y,Z). XI can be a row vector, in which case it specifies a matrix with constant columns. Similarly, YI can be a column vector and it specifies a matrix with constant rows. See help.

detrend  Remove linear trends. Syntax: y = detrend(x); y = detrend(x,'constant'); y = detrend(x,'linear',bp). “bp” is the index to breakpoints between linear segments. Example:
    >> x=linspace(0,10,100);
    >> y=zeros(size(x));
    >> y(1:50)=polyval([-0.3,2],x(1:50));
    >> y(51:100)=polyval([0.4,-1],x(51:100));
    >> y(51:100)=y(51:100)-y(51)+y(50)+mean(diff(y(51:100)));
    >> y=y+randn(size(x))/10;
    >> y1=detrend(y,'linear',50);
    >> plot(x,y,'.-',x,y1,'r.-')
## Fourier transform functions.

**fft**  
Discrete Fourier transform. Usage `coeff=fft(Y)`. For input real or complex vector Y calculates coefficients of Fourier transform. If number of points in Y is power of two using fast algorithm. If Y is matrix fft performs transform on every row. Example:

```matlab
» x=-10:0.1:10;
» y=abs(x)<=2;
» coeff=fft(y);
» a=real(coeff);
» b=imag(coeff);
» p=abs(coeff);
» subplot(2,1,2), plot(1:length(p),p)
» subplot(2,1,1), plot(1:length(a),a,1:length(b),b)
```

Note that ‘0’ frequency coefficients are returned as first and last. To move to the center of the region use `fftshift`.

**fft2**  
Two-dimensional discrete Fourier transform. Can be used to analyze matrixes (images). Same usage as fft. See help.
The `ifft` function in MATLAB performs the inverse discrete Fourier transform. It is defined as `X = ifft(Y)`. This function takes complex numbers as input arguments and generally produces a complex result. Here's an example:

```matlab
x = -10:0.1:10;
y = sin(3*pi*exp(-x.^2/10));
y1 = y + 2*(rand(size(x)) - 0.5);
coeff = fft(y1);
coeff1 = coeff;
lim = 12;
coeff1((lim+1):(length(coeff)-lim)) = zeros(size((lim+1):(length(coeff)-lim)));
subplot(1,1,1), plot(1:length(coeff), abs(coeff), 1:length(coeff), abs(coeff1))
```
» y2=real(ifft(coeff1));
» subplot(1,1,1), plot(x,y,x,y2,'b')

ifft2 Two-dimensional inverse discrete Fourier transform.
abs    Magnitude of complex number or absolute value of real. abs(X)
angle  Phase angle for complex number. angle(X)
unwrap Remove phase angle jumps across 360 degree boundaries. unwrap(X)
fftshift Move zero\textsuperscript{th} lag to center of spectrum. In the example above if » coeff=fftshift(coeff) is used then plot of magnitude is
Function functions - nonlinear numerical methods

ode23  Solve differential equations, low order method. Usage: [T,Y]=ODE23('yprime', T0, Tfinal, Y0) integrates the system of ordinary differential equations described by the M-file YPRIME.M, over the interval T0 to Tfinal, with initial conditions Y0. OUTPUT: T - Returned integration time points (column-vector). Y - Returned solution, one solution column-vector per tout-value. Example: create M-file fder.m:

function f=fder(x,y)
  f=y*sin(x*y);
and use it to solve \( y' = y \times \sin(x \times y) \).
\[
[x, y] = \text{ode23}('fder', -2*\pi, 2*\pi, .52683339,1e-11);
[x1, y1] = \text{ode23}('fder', -2*\pi, 2*\pi, .5268334,1e-11);
\]
plot(x,y,x1,y1,'b')

Solution of \( y'=y\sin(x\times y) \)

Note that relative start amplitude change of 2e-8 at \( x= -2*\pi \) causes \( \sim2^8 \) amplitude change around \( x=0 \).

ode45, ode23, ode113, ode15s, ode23s, ode23t, ode23tb  Solve differential equations, different order methods.
Usage is the same as for ode23. Can be used as \([T,Y] = \text{solver}(odefun,tspan,y0,options,p1,p2...).\) See help for more information
quad  Numerically evaluate integral, low order method. Usage: Q=quad('f',a,b) approximates the integral of f(x) from a to b to within a relative error of 1e-3. 'f' is a string containing the name of the function. Function f must return a vector of output values if given a vector of input values. Q=quad(F,a,b,tol) integrates to a relative error of tol. Example: for function f

```matlab
function f=int_f(x)
f=sin(10 *x);
```

```matlab
» quad('int_f',0,pi,1e-5)
Gives result
ans =
-5.5511e-017
```

Though for a function f=sin(10 ./x) it gives message: Recursion level limit reached in quad. Singularity likely.

quad8  Numerically evaluate integral, high order method. Usage is the same as for quad but it easily finds integrals of oscillating functions like one above: f=sin(10 ./x). Obsolete function, now replaced by quad8.

```matlab
» quad8('int_f',1e-2,5,1e-5)
ans = 0.3198
```

fzero  Find zero of function of one variable. Usage: fzero(F,X) finds a zero of f(x). F is a string containing the name of a real-valued function of a single real variable or a handle to such function. X is a starting guess. The value returned is near a point where F changes sign. An optional third argument sets the options structure that contains fields: Display - Level of display. 'off' displays no output; 'iter' displays output at each iteration; 'final' displays just the final output; 'notify' (default) displays output only if the function does not converge and field TolX - Termination tolerance on x. Additional output arguments are passed to function F. Example: M-file tmp.m containing function f=exp(x)-1 ./x; Zero of f(x)=0 is

```matlab
» fzero('tmp',1,1e-5) -> ans = 0.56714329933851
```
fminbnd

Minimize a function of one variable on a fixed interval. \( x = fminbnd(fun,x1,x2) \) returns a value \( x \) that is a local minimizer of the function that is described in \( fun \) in the interval \( x_1 \leq x \leq x_2 \).

\( x = fminbnd(fun,x1,x2,options) \) minimizes with the optimization parameters specified in the structure \( options \).

You can define these parameters using the \texttt{optimset} function. \texttt{fminbnd} uses these options structure fields:

- **Display** Level of display. 'off' displays no output; 'iter' displays output at each iteration; 'final' displays just the final output; 'notify' (default) displays output only if the function does not converge.
- **MaxFunEvals** Maximum number of function evaluations allowed.
- **MaxIter** Maximum number of iterations allowed.
- **TolX** Termination tolerance on \( x \).

\( x = fminbnd(fun,x1,x2,options,P1,P2,...) \) provides for additional arguments, \( P1, P2, \ldots \), which are passed to the objective function, \( fun(x,P1,P2,...) \). Use \( options=[] \) as a placeholder if no options are set.

See help for more output options. Example:

```matlab
» fminbnd('cos',3,4)
ans =
  3.1416
```

Can be used for data fitting for function with only one variable parameter. Consider example: \( f(t, \tau) = \exp(-t/\tau) \).

“Produce” original data set using \( \tau = 1 \) and calculate value of the function with random noise added:

```matlab
» t=0:0.05:5;
» y=exp(-t)+(rand(size(t))-0.5)/4;
```

Next step is to choose (means write M-file) error function which shows how far model function is from real data. In this particular case we can use following function \texttt{exp_err.m}:

```matlab
function f=exp_err(a,x,y)
f=mean(abs(y-exp(-x/a)));
```

First argument is parameter which we are fitting, second and third \( X \) and \( Y \) of the data. We choose interval where to find \( \tau \): from \( 10^{-4} \) to 10, and do fitting with default options.

```matlab
tau_min=1e-4;
tau_max=10;
» tau=fminbnd('exp_err',tau_min,tau_max,[],t,y)
tau =
  0.9430
» plot(t,y,'b.',t,exp(-t/tau),'r'), grid on
```
Fit error is about 6%. If in this example amplitude of the added noise is decreased twice the fit error is decreased to 1-2%.

**fminsearch**  Minimize function of several variables. Usage: \texttt{fminsearch('F',X0)} attempts to return a vector \texttt{x} which is a local minimizer of \( F(x) \) near the starting vector \( X0 \). 'F' is a string containing the name of the objective function to be minimized or a function handle. \( F(x) \) should be a scalar valued function of a vector variable. Similar to \texttt{fminbnd}, can accept additional parameters: \texttt{fminsearch(F,X0,OPTIONS,P1,P2,...)}. Similar to \texttt{fminbnd} function, \texttt{fminsearch} can be used for model parameters fitting to the data. Steps are almost the same:

♦ Once data is obtained, choose the model for the functional dependence and express it as an equation (or two or as many as necessary).

♦ Create error function similar to one we created for function \texttt{fminbnd}, it has to return scalar of the cumulative error of the fit; most likely with at least three input variables: first one is the vector of parameters to be fitted, following by couple parameters of experimental data.

♦ Create vector of initial values of parameters. Usually it has to be not far from the global minimum since there is often possibility to encounter the local minimum.

♦ Choose values for \texttt{OPTIONS} parameter (\texttt{optimset} function) and run \( X=fminsearch('F',X0,OPTIONS,X,Y) \) to find \( X \) parameters.

♦ Examine the parameters obtained and overall fit error for meaningless and start over if necessary.