Commands

clc	clear command window
clear	delete all variables
clear var1 var2 var3 …	delete selected variables
grid on	adds a grid to the current graph
grid off	removes grid from current graph
help command	provide help on command (e.g. help clear)
help function	provide help on function (e.g. help sqrt)
format compact	prevent blank lines in output
lookfor keyword	looks for functions having *keyword* in their initial comment
who	list all variables
whos	list all variables (with variable sizes)
name = @(*args*) *expression*	define an "anonymous" function

Miscellaneous

comments   % this is a comment

long comments   %{

Everything in here is a comment.
The start and end markers must be on lines by themselves.

%}

long lines   Lines that are uncomfortably long may be broken across **...**
multiple lines by ending all but the last line with a **...**
series of three dots.

creating vectors   linspace (s, e, n)   *n* evenly spaced values from *s* to *e*

s **:** e   from *s* to *e* in steps of 1

s **:** inc **:** e   from *s* in steps of *inc*, last value not to exceed *e*

[v1 v2 v3 v3 ...]   vector containing specified values

ones (1, n)   row vector of n ones

zeros (1, n)   row vector of n zeroes

left division   x\y can be thought of as being equivalent to $x^{-1} * y$

Function m-files

function [*outputs*] = *functionName* (*inputList*)
% *functionName* comment describing what function does
% comments describing how to use function (description of inputs, etc.)
...
end

% can have subfunctions (accessible only from within same file)

<u>Functions</u>

abs(x) absolute value of *x*
cond(A) condition number of matrix
det(A) determinant of matrix
diff(v) returns a vector contains the differences between successive elements of vector *v*. length of result is length(*v*) - 1
error('message') aborts m-file execution with specified error message
exp(x) *e* to the power of *x*
eye(n) creates an *n* by *n* identity matrix
figure (num) switch to working with specified figure
fminbnd(f, l, h) locates minimum of function *f* between *l* and *h*
fplot(f, range) plot function *f* over interval *range*. *range* is a 2 element vector.
example: fplot (f, [0, 10]) see *plot* for line options.
fprintf('format', v1, v2, v3...) outputs values using format defined by '*format'*
format value fields: %f (fixed point), %e, %E (scientific), %g (general), %d (integer)
width and precision control: %12.3f = use 3 decimal places in a field 12 wide
format controls: \n (newline) \t (tab)
fzero(f, loc) returns a root of function *f*
if *loc* is a 2 element vector, looks for a root between *loc*(1) and *loc*(2): more efficient
otherwise looks for a root in the vicinity of *loc*
dy = gradient(y) returns a vector containing first order differences for vector *v*
dy(1) is a forward difference, *dy*(*n*) is a baclwards difference
other elements of *dy* are central differences.
for *x* values one apart the result is the numerical estimate of *dy*/*dx*
for other spacings the result must be divided by the step size
input('prompt') outputs *prompt*, returns a value obtained from the user
integral (f,a,b) integrates function *f* between *a* and *b* (Simpson quadrature)
interp1(x, y, xin, opt) evaluates the spline defined by the data points at *xin*
opt can be 'linear' (linear spline), 'spline' (regular cubic spline), 'cubic' (cubic Hermite spline), or 'pchip' (same thing)
interp1(x, y, opt, 'pp') generates the piecewise polynomial for the spline defined by the data points. *opt* is as above (see also **ppval**)
inv(A) returns the inverse of square matrix *A*
length(x) returns the length of row or column vector *x*
linspace(begin, end, n) returns a vector containing *n* equally spaced values running from *begin* to *end*. if *n* is omitted 100 values are generated
log(x) natural logarithm (log base *e*) of *x*
log10(x) common logarithm (log base 10) of *x*
[L,U] = **lu**(A) LU decomposition of *A* (*L* "psychologically lower triangular")
[L,U,P] = **lu**(A) LU decomposition of *A*
max(*s*1, *s*2) returns the maximum of *s*1 and *s*2 (*s*1, and *s*2 scalars)
max(*v*)returns the maximum value in vector *v*

use [*maxVal*, *maxPos*] = **max** (*v*) to get both max value and its position

min(...) analogous to **max**

[t, y] = ode45(f, xspan, y0) solves the differential equation defined by slope function *f* over the interval *xspan* using initial conditions *y0*. *t* and *y* are column vectors containing the solution function *f* is given values for *t* and *y* and returns *dy*/*dt*

ones(r, c) creates a *r* by *c* matrix of ones

ones(size) creates a matrix of ones with the dimensions contained in 2 element vector *size*

plot(x, y, opt) plot *y* values against *x* values. *x* are *y* are vectors of the same length
opt is optional: 'r' = red, 'b' = blue, 'k' = black, 'g' = green,
'x' = crosses, 'o' = circles, '-' = solid line, ':' = dotted, '--' = dashed, '-.' = dash dot

polyfit(x, y, n)uses data points to generate an *n*th order polynomial

polyval(p, x) evaluates the polynomial defined by vector *p* for value *x*.
if *x* is a matrix the result will be a matrix of the same size.

polyder(p) produces a polynomial that is the derivative of the polynomial defined by vector *p*

polyint(p) produces a polynomial that is the integral of the polynomial defined by vector *p* (with the constant of integration assumed to be zero)

ppval(pp, xin) evaluates piecewise polynomial pp for the value(s) in *xin*

[Q0,R0]= **qr**(Z,0) QR decomposition of Z. The 0 selects "economy" version of outputs.

roots(p) returns a column vector containing all of the roots of the polynomial defined by vector *p*

size(x) returns a 2 element row vector containing the dimensions of *x* (# rows, # cols)
use [rows cols] = sign(x) to get dimensions into separate variables

sqrt(x) returns the square root of *x*

sum(v) sums up all of the elements of vector *v*

spline(x, y, xin) like **interp1** with *opt* = 'spline' (but allows for clamped end condition)
if *y* is exactly two elements longer than x the first and last elements are the first derivatives at the beginning and end of the spline (clamped end condition)

spline(x, y) as above but produces a piecewise polynomial rather than *y* values

title('title') adds a title to the current figure

trapz(x, y) applies trapezoidal integration to data points

trapz(y) as above but *x* values assumed evenly spaced and one apart

xlabel('label') adds a label to the *x* axis of the current figure

ylabel('label') adds a label to the *y* axis of the current figure

zeros(r,c) creates a *r* by *c* matrix of zeroes

zeros(size) creates a matrix of zeros with the dimensions contained in 2 element vector *size*

Trigonometric functions (input in radians): sin(a), cos(a), tan(a), cot(a), sec(a), csc(a)

Inverses (output in radians): asin(x), acos(x), atan(x), atan2(x, y), acot(x), asec(x), acsc(x)

Root Finding:

$$x_R = x_U - \frac{f(x_U)(x_U - x_L)}{f(x_U) - f(x_L)} \quad x_{i+1} = x_i - \frac{f(x_i)(x_{i-1} - x_i)}{f(x_{i-1}) - f(x_i)} \quad x_{i+1} = x_i - \frac{f(x_i)}{f'(x_i)}$$

$$E_{MAX} = \frac{\Delta x^0}{2^N} \quad N = \log_2 \frac{\Delta x^0}{E_{MAX}} = \frac{\log(\Delta x^0 / E_{MAX})}{\log 2}$$

N = estimate number (first is estimate 1), estimate N involves N-1 function evaluations/wall movements

Golden Section:

$$X_2 = X_U - d, \quad X_1 = X_L + d, \quad d = (\phi - 1)(X_U - X_L)$$

$$\phi = \frac{1+\sqrt{5}}{2} = 1.6180339887 \quad E_{MAX} = \frac{\Delta x^0 (\phi - 1)^{N-1}}{2}$$

max error assumes midpoint of interval chosen as answer; as above estimate N involves N-1wall movements

Simultaneous Equations:

$$row = row - (element\_below\_pivot / pivot) * pivot\_row$$

$$Ax = b \Rightarrow LUx = b \Rightarrow Ld = b \text{ and } Ux = d \qquad d = L \backslash b,\ x = U \backslash d$$
$$Ax = b \Rightarrow P^{-1}LU = b \Rightarrow LUx = Pb \Rightarrow Ld = Pb \text{ and } Ux = d \qquad d = L \backslash (P * b),\ x = U \backslash d$$

Iterative methods: $x_{k+1} = Cx_k + d \quad c_{ii} = 0 \quad c_{ij} = -a_{ij} / a_{ii} \quad d_i = b_i / a_{ii}$

Lagrange Polynomial:

$$p(x) = L_1(x)y_1 + L_2(x)y_2 + L_3(x)y_3 + \dots L_N(x)y_N$$

$$L_k(x) = \frac{(x - x_1)(x - x_2)\dots(x - x_{k-1})(x - x_{k+1})\dots(x - x_N)}{(x_k - x_1)(x_k - x_2)\dots(x_k - x_{k-1})(x_k - x_{k+1})\dots(x_k - x_N)}$$

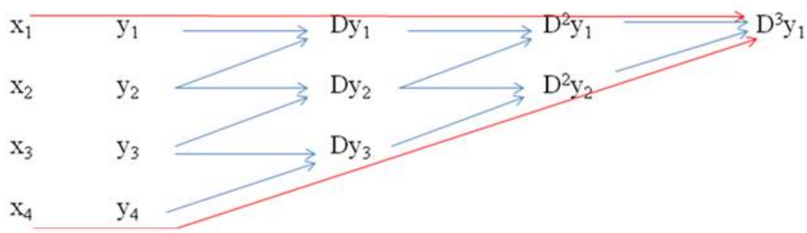
Numerator of $L_k(x)$ is product of all $(x - x_i)$ except for $(x - x_k)$
Denominator of $L_k(x)$ is product of all $(x_k - x_i)$ except for $(x_k - x_k)$

Newton's Polynomial:

$$p(x) = a_1 + a_2(x - x_1) + a_3(x - x_1)(x - x_2) + \dots + a_N(x - x_1)\dots(x - x_{N-1})$$

$$a_1 = y_1 \quad a_2 = Dy_1 \quad a_3 = D^2 y_1 \quad \dots \quad a_N = D^{N-1} y_1$$

$$Dy_i = \frac{y_{i+1} - y_i}{x_{i+1} - x_i} \quad D^2 y_i = \frac{Dy_{i+1} - Dy_i}{x_{i+2} - x_i} \quad D^k y_i = \frac{D^{k-1} y_{i+1} - D^{k-1} y_i}{x_{i+k} - x_i}$$



Regression:

Straight line fit: $y = mx + b \quad m = \dfrac{n\sum x_i y_i - \sum x_i \sum y_i}{n\sum x_i^{\,2} - \left(\sum x_i\right)^2} \quad b = \dfrac{\sum x_i^{\,2} \sum y_i - \sum x_i \sum x_i y_i}{n\sum x_i^{\,2} - \left(\sum x_i\right)^2} = \bar{y} - m\bar{x}$

$$r^2 = \frac{S_t - S_r}{S_t} \quad S_t = \sum (y_i - \bar{y})^2 \quad S_r = \sum (y_i - f(x_i))^2$$

For a straight line fit only: $r = \dfrac{n\sum x_i y_i - \sum x_i \sum y_i}{\sqrt{n\sum x_i^{\,2} - \left(\sum x_i\right)^2}\sqrt{n\sum y_i^{\,2} - \left(\sum y_i\right)^2}}$

For $y = \alpha e^{\beta x} \quad x' = x \quad y' = \ln(y) \quad \alpha = e^b, \beta = m$

For $y = \alpha x^{\beta} \quad x' = \log(x) \quad y' = \log(y) \quad \alpha = 10^b, \beta = m$

For $y = \alpha \dfrac{x}{\beta + x} \quad x' = \dfrac{1}{x} \quad y' = \dfrac{1}{y} \quad \alpha = 1/b, \beta = m/b$

General least squares : $y = a_1 z_1(x) + a_2 z_2(x) + a_3 z_3(x) + \cdots a_m z_m(x)$

Basic solution : $z_{ij} = z_j(x_i) \quad Z^T Z a = Z^T y$ QR decomposition : $Z = QR \quad Ra = Q^T y$

Integration:

$$I = \frac{h}{2}\left(f(x_0) + f(x_1)\right) \;\Rightarrow\; I = \frac{h}{2}\left(f(x_0) + 2f(x_1) + 2f(x_2) + 2f(x_3) + \ldots + 2f(x_{N-1}) + f(x_N)\right)$$

$$I = \frac{h}{3}\left(f(x_0) + 4f(x_1) + f(x_2)\right) \;\Rightarrow\; I = \frac{h}{3}\left(f(x_0) + 4f(x_1) + 2f(x_2) + 4f(x_3) + 2f(x_4) + \ldots + 4f(x_{N-1}) + f(x_N)\right)$$

$$I = \frac{3h}{8}\left(f(x_0) + 3f(x_1) + 3f(x_2) + f(x_3)\right)$$

$I_{j,1}$ = estimate using $h_j \quad h_j = \dfrac{h_1}{2^{j-1}} \quad I_{j,k} = \dfrac{4^{k-1} I_{j+1,k-1} - I_{j,k-1}}{4^{k-1} - 1} \quad |\varepsilon| = \left|\dfrac{I_{1,k} - I_{2,k-1}}{I_{1,k}}\right|$

n	c_0	x_0	c_1	x_1	c_2	x_2	c_3	x_3
2	1	-0.57735	1	0.57735				
3	5/9	-0.77459	8/9	0	5/9	0.77459		
4	0.347855	-0.861136	0.652145	-0.339981	0.652145	0.339981	0.347855	0.861136

Directly usable only for -1 to 1. For *a* to *b* use $\int_a^b f(x)\,dx \approx \left(\frac{b-a}{2}\right)\sum_{i=0}^{n-1}\left(c_i f\left(\frac{b+a}{2}+\frac{b-a}{2}x_i\right)\right)$

Differentiation:

First Derivative	Formula	Error
Forward (2 point)	$\left[f(x_{i+1})-f(x_i)\right]/h$	*O(h)*
Forward (3 point)	$\left[-f(x_{i+2})+4f(x_{i+1})-3f(x_i)\right]/2h$	$O(h^2)$
Backwards (2 point)	$\left[f(x_i)-f(x_{i-1})\right]/h$	*O(h)*
Backwards (3 point)	$\left[3f(x_i)-4f(x_{i-1})+f(x_{i-2})\right]/2h$	$O(h^2)$
Central (2 point)	$\left[f(x_{i+1})-f(x_{i-1})\right]/2h$	$O(h^2)$
Central (4 point)	$\left[-f(x_{i+2})+8f(x_{i+1})-8f(x_{i-1})+f(x_{i-2})\right]/12h$	$O(h^4)$

$D_{j,1} = \text{estimate using } h_j \quad h_j = \frac{h_1}{2^{j-1}} \quad D_{j,k} = \frac{4^{k-1}D_{j+1,k-1} - D_{j,k-1}}{4^{k-1}-1} \quad |\varepsilon| = \left|\frac{D_{1,k} - D_{2,k-1}}{D_{1,k}}\right|$

ODE's:

$$\frac{dy}{dt} = f(t,y) \quad y_{i+1} = y_i + \phi h \quad \text{Euler}: \phi = f(t_i, y_i) \quad \text{Midpoint}: \phi = f(t_{i+1/2}, y_{i+1/2})$$

$$\text{Heun}: y^0{}_{i+1} = y_i + f(t_i, y_i)h \quad y^j{}_{i+1} = y_i + \frac{f(t_i, y_i) + f(t_{i+1}, y^{j-1}{}_{i+1})}{2}h$$

$$\text{no iteration } y_{i+1} = y^1{}_{i+1} \quad \text{with iteration } y_{i+1} = y^m{}_{i+1} \quad \left|\frac{y^m{}_{i+1} - y^{m-1}{}_{i+1}}{y^m{}_{i+1}}\right| < \varepsilon$$

$$A\frac{d^2y}{dt^2} + B\frac{dy}{dt} + Cy = D \quad \Rightarrow \quad \frac{dy}{dt} = y', \quad \frac{dy'}{dt} = \frac{D - By' - Cy}{A}$$