Appendix A

List of MATLAB Routines with Descriptions

Table A.1: Description of MATLAB Programs and Selected Functions

<table>
<thead>
<tr>
<th>Routine</th>
<th>Chapter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>finance</td>
<td>1</td>
<td>Financial analysis program illustrating programming methods.</td>
</tr>
<tr>
<td>inputv</td>
<td>1</td>
<td>Function to read several data items on one line.</td>
</tr>
<tr>
<td>polyplot</td>
<td>2</td>
<td>Program comparing polynomial and spline Interpolation.</td>
</tr>
<tr>
<td>squarrun</td>
<td>2</td>
<td>Program illustrating conformal mapping of a square.</td>
</tr>
<tr>
<td>squarmap</td>
<td>2</td>
<td>Function for Schwarz-Christoffel mapping of a circular disk inside a square.</td>
</tr>
<tr>
<td>cubrange</td>
<td>2</td>
<td>Function to compute data range limits for 2D or 3D data.</td>
</tr>
<tr>
<td>pendulum</td>
<td>2</td>
<td>Program showing animated large oscillations of a pendulum.</td>
</tr>
<tr>
<td>animpen</td>
<td>2</td>
<td>Function showing pendulum animation.</td>
</tr>
<tr>
<td>smdplot</td>
<td>2</td>
<td>Program to animate forced motion of a spring-mass-damper system.</td>
</tr>
<tr>
<td>smsolve</td>
<td>2</td>
<td>Function to solve a constant coefficient linear second order differential equation with a harmonic forcing function.</td>
</tr>
<tr>
<td>strngrun</td>
<td>2</td>
<td>Program animating wave motion in a string with given initial deflection.</td>
</tr>
<tr>
<td>strngwav</td>
<td>2</td>
<td>Function to compute deflections of a vibrating string.</td>
</tr>
<tr>
<td>animate</td>
<td>2</td>
<td>Function to show animation of a vibrating string.</td>
</tr>
</tbody>
</table>

continued on next page
<table>
<thead>
<tr>
<th>Routine</th>
<th>Chapter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>splinerr</td>
<td>2</td>
<td>Program showing differential geometry properties of a space curve.</td>
</tr>
<tr>
<td>curvprp</td>
<td>2</td>
<td>Function using spline interpolation to compute differential properties of a space curve.</td>
</tr>
<tr>
<td>splined</td>
<td>2</td>
<td>Function to compute first or second derivatives of a cubic spline.</td>
</tr>
<tr>
<td>srfex</td>
<td>2</td>
<td>Program illustrating combined plotting of several surfaces.</td>
</tr>
<tr>
<td>frus</td>
<td>2</td>
<td>Function to compute points on a frustum.</td>
</tr>
<tr>
<td>surfman</td>
<td>2</td>
<td>Function to plot several functions together without distortion.</td>
</tr>
<tr>
<td>rgdbodo</td>
<td>2</td>
<td>Program illustrating 3D rigid body rotation and translation.</td>
</tr>
<tr>
<td>rotatran</td>
<td>2</td>
<td>Function to perform coordinate rotation.</td>
</tr>
<tr>
<td>membran</td>
<td>3</td>
<td>Program illustrating static deflection of a membrane.</td>
</tr>
<tr>
<td>mbvprun</td>
<td>3</td>
<td>Program to solve a mixed boundary value problem for a circular disk.</td>
</tr>
<tr>
<td>makratsq</td>
<td>3</td>
<td>Program showing conformal mapping of a square using rational functions.</td>
</tr>
<tr>
<td>ratcof</td>
<td>3</td>
<td>Function to compute coefficients for rational function interpolation.</td>
</tr>
<tr>
<td>raterp</td>
<td>3</td>
<td>Function to evaluate a rational function using coefficients from function raterp.</td>
</tr>
<tr>
<td>strdyneq</td>
<td>3</td>
<td>Program to solve the structural dynamics equation using eigenvalue-eigenvector methods.</td>
</tr>
<tr>
<td>fhrmck</td>
<td>3</td>
<td>Function to solve a linear second order matrix differential equation having a harmonic forcing function.</td>
</tr>
<tr>
<td>recmemfr</td>
<td>3</td>
<td>Program illustrating use of functions null and eig to compute rectangular membrane frequencies.</td>
</tr>
</tbody>
</table>

*continued on next page*
<table>
<thead>
<tr>
<th>Routine</th>
<th>Chapter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>multimer</td>
<td>3</td>
<td>Program comparing execution of intrinsic MATLAB matrix multiplication and slow Fortran style using loops.</td>
</tr>
<tr>
<td>lintrp</td>
<td>4</td>
<td>Function for piecewise linear interpolation allowing finite jump discontinuities.</td>
</tr>
<tr>
<td>curvprop</td>
<td>4</td>
<td>Program to compute the length and area bounded by a curve defined by spline interpolation.</td>
</tr>
<tr>
<td>spcof</td>
<td>4</td>
<td>Function to compute spline interpolation coefficients used by function <code>spterp</code>.</td>
</tr>
<tr>
<td>spterp</td>
<td>4</td>
<td>Function to interpolate, differentiate, and integrate a cubic spline having general end conditions.</td>
</tr>
<tr>
<td>powermat</td>
<td>4</td>
<td>Function used by functions <code>spcof</code> and <code>spterp</code>.</td>
</tr>
<tr>
<td>splineq</td>
<td>4</td>
<td>Function to interpolate, integrate, and differentiate using the intrinsic function <code>spline</code>.</td>
</tr>
<tr>
<td>splincof</td>
<td>4</td>
<td>Function that computes coefficients used by <code>splineq</code> to handle general end conditions.</td>
</tr>
<tr>
<td>matlbdat</td>
<td>4</td>
<td>Program that draws the word MATLAB using a spline.</td>
</tr>
<tr>
<td>finitdif</td>
<td>4</td>
<td>Program to compute finite difference formulas.</td>
</tr>
<tr>
<td>findifco</td>
<td>4</td>
<td>Function to compute finite difference formulas for derivatives of arbitrary order.</td>
</tr>
<tr>
<td>simpson</td>
<td>5</td>
<td>Function using Simpson’s rule to integrate an exact function or one defined by spline interpolation.</td>
</tr>
<tr>
<td>gcquad</td>
<td>5</td>
<td>Function to perform composite Gauss integration of arbitrary order, and return the base points and weight factors.</td>
</tr>
<tr>
<td>quadtest</td>
<td>5</td>
<td>Program comparing the performance of <code>gcquad</code> and <code>quadl</code> for several test functions.</td>
</tr>
</tbody>
</table>

continued on next page
<table>
<thead>
<tr>
<th>Routine</th>
<th>Chapter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>areaprog</td>
<td>5</td>
<td>Program to compute area, centroidal coordinates and inertial properties of general areas bounded by spline curves.</td>
</tr>
<tr>
<td>apropl</td>
<td>5</td>
<td>Function to compute geometrical properties of general areas.</td>
</tr>
<tr>
<td>volrevol</td>
<td>5</td>
<td>Program to compute geometrical properties of partial volumes of revolution bounded by spline curves.</td>
</tr>
<tr>
<td>volrev</td>
<td>5</td>
<td>Function to compute geometrical properties of partial volumes of revolution.</td>
</tr>
<tr>
<td>rotasurf</td>
<td>5</td>
<td>Function to plot a partial surface of revolution.</td>
</tr>
<tr>
<td>ropesymu</td>
<td>5</td>
<td>Program using numerical and symbolic computation to evaluate geometrical properties of a rope shaped solid.</td>
</tr>
<tr>
<td>ropedraw</td>
<td>5</td>
<td>Function to draw a twisted rope shaped surface.</td>
</tr>
<tr>
<td>twistprop</td>
<td>5</td>
<td>Function using symbolic computation to obtain geometrical properties.</td>
</tr>
<tr>
<td>srfv</td>
<td>5</td>
<td>Function to compute geometrical properties of a solid specified by general surface coordinates.</td>
</tr>
<tr>
<td>polhdrun</td>
<td>5</td>
<td>Program to produce geometrical properties and a surface plot of an arbitrary polyhedron.</td>
</tr>
<tr>
<td>polhedron</td>
<td>5</td>
<td>Function for geometrical properties of a polyhedron.</td>
</tr>
<tr>
<td>polyxy</td>
<td>5</td>
<td>Function for geometrical properties of a polygon.</td>
</tr>
<tr>
<td>sqrtquadtest</td>
<td>5</td>
<td>Program using <code>quadl</code> and <code>gquad</code> to evaluate integrals having square root type singularities at the integration end points.</td>
</tr>
<tr>
<td>quadqsqrt</td>
<td>5</td>
<td>Function applying <code>gquad</code> to integrals having square root type singularities.</td>
</tr>
<tr>
<td>quadlsqrt</td>
<td>5</td>
<td>Function applying <code>quadl</code> to integrals having square root type singularities.</td>
</tr>
<tr>
<td>Routine</td>
<td>Chapter</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>---------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>triplint</td>
<td>5</td>
<td>Program applying Gauss quadrature to evaluate a triple integral with variable</td>
</tr>
<tr>
<td></td>
<td></td>
<td>integration limits.</td>
</tr>
<tr>
<td>plotjrun</td>
<td>6</td>
<td>Program to compute and plot integer order Bessel functions using the FFT.</td>
</tr>
<tr>
<td>runimpv</td>
<td>6</td>
<td>Program using the FFT to analyze earthquake data.</td>
</tr>
<tr>
<td>fouapprox</td>
<td>6</td>
<td>Function for Fourier series approximation of a general function.</td>
</tr>
<tr>
<td>fouseris</td>
<td>6</td>
<td>Program to plot truncated Fourier series expansions of general functions.</td>
</tr>
<tr>
<td>fousum</td>
<td>6</td>
<td>Function to sum a Fourier series and include coefficient smoothing.</td>
</tr>
<tr>
<td>cablinea</td>
<td>7</td>
<td>Program showing modal superposition analysis of a swinging cable.</td>
</tr>
<tr>
<td>udfrevid</td>
<td>7</td>
<td>Function computing undamped response of a second order matrix differential</td>
</tr>
<tr>
<td></td>
<td></td>
<td>equation with general initial conditions.</td>
</tr>
<tr>
<td>strdynrk</td>
<td>7</td>
<td>Function using ode45 to solve a second order matrix differential equation.</td>
</tr>
<tr>
<td>deislnr</td>
<td>7</td>
<td>Program comparing implicit second and fourth order integrators which use</td>
</tr>
<tr>
<td></td>
<td></td>
<td>fixed stepsize.</td>
</tr>
<tr>
<td>mckde2i</td>
<td>7</td>
<td>Function to solve a matrix ODE using a second order fixed stepsize integrator.</td>
</tr>
<tr>
<td>mckde4i</td>
<td>7</td>
<td>Function to solve a matrix ODE using a fourth order fixed stepsize integrator.</td>
</tr>
<tr>
<td>rkdestab</td>
<td>8</td>
<td>Program to plot stability zones for Runge-Kutta integrators.</td>
</tr>
<tr>
<td>prun</td>
<td>8</td>
<td>Program illustrating ode45 response calculation of an inverted pendulum.</td>
</tr>
<tr>
<td>toprun</td>
<td>8</td>
<td>Program for dynamic response of a spinning top.</td>
</tr>
<tr>
<td>traject</td>
<td>8</td>
<td>Program for a projectile trajectory.</td>
</tr>
<tr>
<td>cablenl</td>
<td>8</td>
<td>Program illustrating animated nonlinear dynamic response for a multi-link</td>
</tr>
<tr>
<td></td>
<td></td>
<td>cable of rigid links.</td>
</tr>
</tbody>
</table>

*continued on next page*
<table>
<thead>
<tr>
<th>Routine</th>
<th>Chapter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>plotmotn</td>
<td>8</td>
<td>Function to animate the dynamic response of a cable.</td>
</tr>
<tr>
<td>sprchan</td>
<td>8</td>
<td>Program for animated nonlinear dynamics of an elastic cable shaken at both ends.</td>
</tr>
<tr>
<td>laplarec</td>
<td>9</td>
<td>Program using Fourier series to solve the Laplace equation in a rectangle having general boundary conditions.</td>
</tr>
<tr>
<td>recseris</td>
<td>9</td>
<td>Function to compute a harmonic function and gradient components in a rectangular region.</td>
</tr>
<tr>
<td>stringft</td>
<td>9</td>
<td>Program for Fourier series solution and animated response for a string with given initial displacement.</td>
</tr>
<tr>
<td>forcmove</td>
<td>9</td>
<td>Program for response of a string subjected to a moving concentrated load.</td>
</tr>
<tr>
<td>membwave</td>
<td>9</td>
<td>Program animating the response of a rectangular or circular membrane subjected to an oscillating concentrated force.</td>
</tr>
<tr>
<td>besjroot</td>
<td>9</td>
<td>Function to compute a table of integer order Bessel function roots.</td>
</tr>
<tr>
<td>membananim</td>
<td>9</td>
<td>Function to show animated membrane response.</td>
</tr>
<tr>
<td>bemimpac</td>
<td>9</td>
<td>Program showing wave propagation in a simply supported beam subjected to an oscillating end moment.</td>
</tr>
<tr>
<td>beamanim</td>
<td>9</td>
<td>Function to animate the motion of a vibrating beam.</td>
</tr>
<tr>
<td>pilevibs</td>
<td>9</td>
<td>Program illustrating the response of a pile embedded in an oscillating elastic foundation.</td>
</tr>
<tr>
<td>slabheat</td>
<td>9</td>
<td>Program for heat conduction in a slab having sinusoidally varying end temperature.</td>
</tr>
</tbody>
</table>

*continued on next page*
<table>
<thead>
<tr>
<th>Routine</th>
<th>Chapter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>heatcyln</td>
<td>9</td>
<td>Program analyzing transient heat conduction in a circular cylinder.</td>
</tr>
<tr>
<td>tempstdy</td>
<td>9</td>
<td>Function for the steady-state temperature in a circular cylinder with general</td>
</tr>
<tr>
<td></td>
<td></td>
<td>boundary conditions.</td>
</tr>
<tr>
<td>foubesco</td>
<td>9</td>
<td>Function to compute coefficients in a Fourier-Bessel series.</td>
</tr>
<tr>
<td>besjtabl</td>
<td>9</td>
<td>Function giving a table of integer order Bessel function roots.</td>
</tr>
<tr>
<td>rector</td>
<td>9</td>
<td>Program to compute torsional stresses in a beam of rectangular cross section.</td>
</tr>
<tr>
<td>eigverr</td>
<td>10</td>
<td>Program comparing eigenvalues of a second order differential equation</td>
</tr>
<tr>
<td></td>
<td></td>
<td>computed using finite difference methods and using collocation with spline</td>
</tr>
<tr>
<td></td>
<td></td>
<td>interpolation.</td>
</tr>
<tr>
<td>prnstres</td>
<td>10</td>
<td>Function to compute principal stresses and principal directions for a symmet-</td>
</tr>
<tr>
<td></td>
<td></td>
<td>ric second order stress tensor.</td>
</tr>
<tr>
<td>trusvibs</td>
<td>10</td>
<td>Program to compute and show animation of the natural vibration modes of a</td>
</tr>
<tr>
<td></td>
<td></td>
<td>general pin connected truss.</td>
</tr>
<tr>
<td>drawtruss</td>
<td>10</td>
<td>Function to draw the deflection modes of a truss.</td>
</tr>
<tr>
<td>eigsym</td>
<td>10</td>
<td>Function solving the constrained eigenvalue problem associated with an</td>
</tr>
<tr>
<td></td>
<td></td>
<td>elastic structure fixed as selected points.</td>
</tr>
<tr>
<td>elmstf</td>
<td>10</td>
<td>Function to form mass and stiffness matrices of a pin connected truss.</td>
</tr>
<tr>
<td>colbuc</td>
<td>10</td>
<td>Program to compute buckling loads of a variable depth column with general</td>
</tr>
<tr>
<td></td>
<td></td>
<td>end conditions.</td>
</tr>
<tr>
<td>cbfreq</td>
<td>10</td>
<td>Program comparing cantilever beam natural frequencies computed by exact,</td>
</tr>
<tr>
<td></td>
<td></td>
<td>finite difference, and finite element methods.</td>
</tr>
</tbody>
</table>

*continued on next page*
<table>
<thead>
<tr>
<th>Routine</th>
<th>Chapter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>cbfrqnmw</td>
<td>10</td>
<td>Function to compute exact cantilever beam frequencies.</td>
</tr>
<tr>
<td>cbfrqfdm</td>
<td>10</td>
<td>Function to compute cantilever beam frequencies using finite difference methods.</td>
</tr>
<tr>
<td>cbfrqfem</td>
<td>10</td>
<td>Function to compute cantilever beam frequencies using the finite element method.</td>
</tr>
<tr>
<td>elipfreq</td>
<td>10</td>
<td>Program for natural frequencies and animation of the mode shapes of an elliptic membrane.</td>
</tr>
<tr>
<td>freqsimpl</td>
<td>10</td>
<td>Function to compute elliptic membrane natural frequencies and mode shapes.</td>
</tr>
<tr>
<td>eigenrec</td>
<td>10</td>
<td>Function to solve a rectangular eigenvalue problem of the form: $XA + BX = \lambda(XC + DX)$.</td>
</tr>
<tr>
<td>plotmode</td>
<td>10</td>
<td>Function to plot the mode shapes of the membrane.</td>
</tr>
<tr>
<td>vdb</td>
<td>11</td>
<td>Program to compute shear, moment, slope, and deflection in a variable depth multi-support beam with general external loading conditions.</td>
</tr>
<tr>
<td>extload</td>
<td>11</td>
<td>Function to compute load and deformation quantities for distributed and concentrated loading on a beam.</td>
</tr>
<tr>
<td>sngf</td>
<td>11</td>
<td>Singularity function used to describe beam loads.</td>
</tr>
<tr>
<td>trapsum</td>
<td>11</td>
<td>Trapezoidal rule function used to integrate beam functions.</td>
</tr>
<tr>
<td>sqrtsurf</td>
<td>12</td>
<td>Function used to illustrate branch cut discontinuities for an analytic function.</td>
</tr>
<tr>
<td>elipinrvr</td>
<td>12</td>
<td>Function to invert the function mapping the exterior of a circle onto the exterior of an ellipse.</td>
</tr>
<tr>
<td>elipdplt</td>
<td>12</td>
<td>Program showing grid lines for conformal mapping of a circular disk onto an elliptic disk.</td>
</tr>
</tbody>
</table>

*continued on next page*
## LIST OF MATLAB ROUTINES WITH DESCRIPTIONS

<table>
<thead>
<tr>
<th>Routine</th>
<th>Chapter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>elipdisk</td>
<td>12</td>
<td>Function mapping an elliptic disk onto a circular disk.</td>
</tr>
<tr>
<td>gridview</td>
<td>12</td>
<td>Function to plot a curvilinear coordinate grid.</td>
</tr>
<tr>
<td>linfrac</td>
<td>12</td>
<td>Function to perform linear fractional transformations.</td>
</tr>
<tr>
<td>crc2crc</td>
<td>12</td>
<td>Function analyzing mapping of circles and straight lines under a linear fractional transformation.</td>
</tr>
<tr>
<td>eccentric</td>
<td>12</td>
<td>Function to determine a concentric annulus which maps onto a given eccentric annulus.</td>
</tr>
<tr>
<td>swcsq10</td>
<td>12</td>
<td>Program illustrating both interior and exterior maps regarding a circle and a square.</td>
</tr>
<tr>
<td>squarat</td>
<td>12</td>
<td>Rational function map taking the inside of a circle onto the interior of a square or the exterior of a square onto the exterior of a square.</td>
</tr>
<tr>
<td>swcsqmap</td>
<td>12</td>
<td>Function using truncated series expansions in relation to circle to square maps.</td>
</tr>
<tr>
<td>lapcrcl</td>
<td>12</td>
<td>Program solving the Laplace equation in a circular disk for either Dirichlet or Neumann boundary conditions.</td>
</tr>
<tr>
<td>cauchtst</td>
<td>12</td>
<td>Program using a Cauchy integral to solve a mixed boundary value problem for a circular disk.</td>
</tr>
<tr>
<td>cauchint</td>
<td>12</td>
<td>Function to numerically evaluate a Cauchy integral.</td>
</tr>
<tr>
<td>elipcyl</td>
<td>12</td>
<td>Program illustrating inviscid fluid flow about an elliptic cylinder in an infinite stream.</td>
</tr>
<tr>
<td>runtors</td>
<td>12</td>
<td>Program using a Cauchy integral and conformal mapping to compute torsional stresses in a beam.</td>
</tr>
<tr>
<td>runplate</td>
<td>12</td>
<td>Program using complex stress functions to compute stresses in a plate with a circular hole.</td>
</tr>
</tbody>
</table>

*continued on next page*
<table>
<thead>
<tr>
<th>Routine</th>
<th>Chapter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>platecrc</td>
<td>12</td>
<td>Function computing series coefficients for complex stress functions pertaining to a plate with a circular hole.</td>
</tr>
<tr>
<td>strfun</td>
<td>12</td>
<td>Function to evaluate stress functions phi and psi.</td>
</tr>
<tr>
<td>cartstrs</td>
<td>12</td>
<td>Function using complex stress functions to evaluate Cartesian stress components.</td>
</tr>
<tr>
<td>rec2polr</td>
<td>12</td>
<td>Function transforming stress components from Cartesian to polar coordinates.</td>
</tr>
<tr>
<td>elipmaxst</td>
<td>12</td>
<td>Program using conformal mapping and complex stress functions to compute stress in a plate with an elliptic hole.</td>
</tr>
<tr>
<td>runtraj</td>
<td>13</td>
<td>Program using one-dimensional search to optimize a projectile trajectory.</td>
</tr>
<tr>
<td>vibfit</td>
<td>13</td>
<td>Program using multi-dimensional search to fit a nonlinear equation to vibration response data.</td>
</tr>
<tr>
<td>cablsolv</td>
<td>13</td>
<td>Program to compute large deflection static equilibrium of a loaded cable.</td>
</tr>
<tr>
<td>brachist</td>
<td>13</td>
<td>Program to determine a minimum time descent curve (brachistochrone).</td>
</tr>
<tr>
<td>cylclose</td>
<td>13</td>
<td>Program using multi-dimensional search to find the closest points on two adjacent circular cylinders.</td>
</tr>
<tr>
<td>surf2surf</td>
<td>13</td>
<td>Function using exhaustive search to find the closest points on two surfaces.</td>
</tr>
<tr>
<td>nelmed</td>
<td>13</td>
<td>Function similar to fminsearch which implements the Nelder and Mead algorithm for multi-dimensional search.</td>
</tr>
</tbody>
</table>
Appendix B

Selected Utility and Application Functions

Function animate

function animate(x,y,titl,tim,trace)
% animate(x,y,titl,tim,trace)
% ~~~~~~~~~~~~~~~~~~~~~~~~~~~
% This function performs animation of a 2D curve
% x,y - arrays with columns containing curve positions
% for successive times. x can also be a single
% vector if x values do not change. The animation
% is done by plotting (x(:,j),y(:,j)) for
% j=1:size(y,2).
% titl- title for the graph
% tim - the time in seconds between successive plots

if nargin<5, trace=0; else, trace=1; end;
if nargin<4, tim=.05; end
if nargin<3, trac=''; end; [np,nt]=size(y);
if min(size(x))==1, j=ones(1,nt); x=x(:);
else, j=1:nt; end; ax=newplot;
if trace, XOR='none'; else, XOR='xor'; end
r=[min(x(:)),max(x(:)),min(y(:)),max(y(:))];
axis('equal') % Needed for an undistorted plot
axis(r), % axis('off')
curve = line('color','k','linestyle','-',... 'erase',XOR, 'xdata',[],'ydata',[]);
xlabel('x axis'), ylabel('y axis'), title(titl)
for k = 1:nt
  set(curve,'xdata',x(:,j(k)),'ydata',y(:,k))
  if tim>0, pause(tim), end, drawnow, shg
end
Function aprop

function [p,zplot]=aprop(xd,yd,kn)
%
% [p,zplot]=aprop(xd,yd,kn)
% ~~~~~~~~~~~~~~~~~~~~~~~~~
% This function determines geometrical properties
% of a general plane area bounded by a spline
% curve
% xd,yd - data points for spline interpolation
% with the boundary traversed in counter-
% clockwise direction. The first and last
% points must match for boundary closure.
% kn - vector of indices of points where the
% slope is discontinuous to handle corners
% like those needed for shapes such as a
% rectangle.
% p - the vector [a,xcg,ycg,axx,axy,ayy]
% containing the area, centroid coordinates,
% moment of inertia about the y-axis,
% product of inertia, and moment of inertia
% about the x-axis.
% zplot - complex vector of boundary points for
% plotting the spline interpolated geometry.
% The points include the numerical quadrature
% points interspersed with data values.
% User functions called: gcquad, curve2d
if nargin==0
    td=linspace(0,2*pi,13); kn=[1,13];
    xd=cos(td)+1; yd=sin(td)+1;
end
nd=length(xd); nseg=nd-1;
[dum,bp,wf]=gcquad([],1,nd,6,nseg);
[z,zplot,zp]=curve2d(xd,yd,kn,bp);
w=[ones(size(z)), z, z.*conj(z), z.^2].*...
    repmat(imag(conj(z).*zp),1,4);
v=(wf'*w)./ [2,3,8,8]; vr=real(v); vi=imag(v);
p=[vr(1:2),vi(2),vr(3)+vr(4),vi(4),vr(3)-vr(4)];
p(2)=p(2)/p(1); p(3)=p(3)/p(1);
Function besjroot

```matlab
function rts=besjroot(norder,nrts,tol)

% This function computes an array of positive roots
% of the integer order Bessel functions besselj of
% the first kind for various orders. A chosen number
% of roots is computed for each order
% norder - a vector of function orders for which
% roots are to be computed. Taking 3:5
% for norder would use orders 3, 4, and 5.
% nrts - the number of positive roots computed for
% each order. Roots at x=0 are ignored.
% rts - an array of roots having length(norder)
% rows and nrts columns. The element in
% column k and row i is the k'th root of
% the function besselj(norder(i),x).
% tol - error tolerance for root computation.

if nargin<3, tol=1e-5; end
jn=inline('besselj(n,x)','x','n');
N=length(norder); rts=ones(N,nrts)*nan;
opt=optimset('TolFun',tol,'TolX',tol);
for k=1:N
  n=norder(k); xmax=1.25*pi*(nrts-1/4+n/2);
xsrch=.1:pi/4:xmax; fb=besselj(n,xsrch);
f==length(fb); K=find(fb(1:f-1).*fb(2:f)<=0);
  if length(K)<nrts
    disp('Search error in function besjroot');
    rts=nan; return
  else
    K=K(1:nrts);
    for i=1:nrts
      interval=xsrch(K(i):K(i)+1);
      rts(k,i)=fzero(jn,interval,opt,n);
    end
  end
end
```
Function cubrange

function range=cubrange(xyz,ovrsiz)
% range=cubrange(xyz,ovrsiz)
% ~~~~~~~~~~~~~~~~~~~~~~~~~~
% This function determines limits for a square
% or cube shaped region for plotting data values
% in the columns of array xyz to an undistorted
% scale
% xyz - a matrix of the form [x,y] or [x,y,z]
% where x,y,z are vectors of coordinate
% points
% ovrsiz - a scale factor for increasing the
% window size. This parameter is set to
% one if only one input is given.
% range - a vector used by function axis to set
% window limits to plot x,y,z points
% undistorted. This vector has the form
% [xmin,xmax,ymin,ymax] when xyz has
% only two columns or the form
% [xmin,xmax,ymin,ymax,zmin,zmax]
% when xyz has three columns.
% User m functions called: none
%----------------------------------------------

if nargin==1, ovrsiz=1; end
pmin=min(xyz); pmax=max(xyz); pm=(pmin+pmax)/2;
pd=max(ovrsiz/2*(pmax-pmin));
if length(pmin)==2
  range=pm([1,1,2,2])+pd*[-1,1,-1,1];
else
  range=pm([1 1 2 2 3 3])+pd*[-1,1,-1,1,-1,1];
end

Function curve2d

function [z,zplot,zp]=curve2d(xd,yd,kn,t)
%
% [z,zplot,zp]=curve2d(xd,yd,kn,t)
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% This function generates a spline curve through
% given data points with corners (slope dis-
% continuities) allowed as selected points.
% xd,yd - real data vectors of length nd
% defining the curve traversed in
% counterclockwise order.
% kn  - vectors of point indices, between one
% and nd, where slope discontinuities
% occur
% t   - a vector of parameter values at which
% points on the spline curve are
% computed. The components of t normally
% range from one to nd, except when t is
% a negative integer,-m. Then t is
% replaced by a vector of equally spaced
% values using m steps between each
% successive pair of points.
% z   - vector of points on the spline curve
% corresponding to the vector t
% zplot - a complex vector of points suitable
% for plotting the geometry
% zp   - first derivative of z with respect to
% t for the same values of t as is used
% to compute z
% User m functions called: splined
%---------------------------------------------------
%------------------------------------------------------------------------------------------------------------------

% nd=length(xd); zd=xd(:)+i*yd(:); td=(1:nd)';
if isempty(kn), kn=[1;nd]; end
kn=sort(kn(:)); if kn(1)==1, kn=[1;kn]; end
if kn(end)==nd, kn=[kn;nd]; end
N=length(kn)-1; m=round(abs(t(1)));
if -t(1)==m, t=linspace(1,nd,1+N*m)'; end
z=[]; zp=[]; zplot=[];
for j=1:N
k1=kn(j); k2=kn(j+1); K=k1:k2;
k=find(k1<=t & t<k2);
if j==N, k=find(k1<=t & t<=k2); end
if isempty(k)
zk=spline(K,zd(K),t(k)); z=[z;zk];
zplot=[zplot;zd(k1);zk];
if nargout==3
© 2003 by CRC Press LLC
zp=[zp;splined(K,zd(K),t(k))];
end
end
end

Function eigenrec

function [eigs,vecs,Amat,Bmat]=eigenrec(A,B,C,D)
% [eigs,vecs,Amat,Bmat]=eigenrec(A,B,C,D)
% Solve a rectangular eigenvalue problem of the form: X*A+B*X=lambda*(X*C+D*X)
% n=size(B,1); m=size(A,2); s=[n,m]; N=n*m;
% Amat=zeros(N,N); Bmat=Amat; kn=1:n; km=1:m;
for i=1:n
    IK=sub2ind(s,i*ones(1,m),km);
    Bikn=B(i,kn); Dikn=D(i,kn);
    for j=1:m
        I=sub2ind(s,i,j);
        Amat(I,IK)=A(km,j)'; Bmat(I,IK)=C(km,j)';
        KJ=sub2ind(s,km,j*ones(1,n));
        Amat(I,KJ)=Amat(I,KJ)+Bikn;
        Bmat(I,KJ)=Bmat(I,KJ)+Dikn;
    end
end
[vecs,eigs]=eig(Bmat\Amat);
[eigs,k]=sort(diag(eigs));
vecs=reshape(vecs(:,k),n,m,N);

Function eigsym

function [evecs,eigvals]=eigsym(k,m,c)
% [evecs,eigvals]=eigsym(k,m,c)
% This function solves the eigenvalue of the constrained eigenvalue problem
% k*x=(lambda)*m*x, with c*x=0.
% Matrix k must be real symmetric and matrix
% m must be symmetric and positive definite;
% otherwise, computed results will be wrong.

© 2003 by CRC Press LLC
function [t,y,lambda]=fhrmck(m,c,k,f1,f2,w,tlim,nt,y0,v0)
% [t,y,lambda]=fhrmck(m,c,k,f1,f2,w,tlim,nt,y0,v0)
% This function uses eigenfunction analysis to solve
% the matrix differential equation
% m*y''(t)+c*y'(t)+k*y(t)=f1*cos(w*t)+f2*sin(w*t)
% with initial conditions of y(0)=y0, y'(0)=v0
% The solution is general unless 1) a zero or repeated
% eigenvalue occurs or 2) the system is undamped and
% the forcing function matches a natural frequency.
% If either error condition occurs, program execution
% terminates with t and y set to nan.
% m,c,k - mass, damping, and stiffness matrices
% f1,f2 - amplitude vectors for the sine and cosine forcing function components
% w - frequency of the forcing function
% tlim - a vector containing the minimum and maximum time limits for evaluation of the solution
% nt - the number of times at which the solution is evaluated within the chosen limits
% y0,v0 - initial position and velocity vectors
% t - vector of time values for the solution
% y - matrix of solution values where y(i,j) is the value of component j at time t(i)
% lam - the complex natural frequencies arranged in order of increasing absolute value

if nargin==0 % Generate default data using 2 masses
    m=eye(2,2); k=[2,-1;-1,1]; c=.3*k;
    f1=[0;1]; f2=[0;0]; w=0.6; tlim=[0,100]; nt=400;
end
n=size(m,1); t=linspace(tlim(1),tlim(2),nt);
if nargin<10, y0=zeros(n,1); v0=y0; end

% Determine eigenvalues and eigenvectors for the homogeneous solution
A=[zeros(n,n), eye(n,n); -m\[k, c]];
[U,lam]=eig(A); [lam,j]=sort(diag(lam)); U=U(:,j);

% Check for zero or repeated eigenvalues and for undamped resonance
wmin=abs(lam(1)); tol=wmin/1e6;
[dif,J]=min(abs(lam-i*w)); lj=num2str(lam(J));
if wmin==0, disp(''); return
    disp(['The homogeneous equation has a zero eigenvalue which is not allowed.'])
    disp('Execution is terminated'), disp('')
    t=nan; y=nan; return
elseif any(abs(diff(lam))<tol)
    disp('A repeated eigenvalue occurred.')
    disp('Execution is terminated'), disp('')
    t=nan; y=nan; return
elseif dif<tol & sum(abs(c(:)))==0

© 2003 by CRC Press LLC
59: disp('The system is undamped and the forcing')
60: disp(['function resonates with ', ...
61: 'eigenvalue ', lj])
62: disp('Execution is terminated.')
63: disp(' '), t=nan; y=nan; return
64: else
65: % Determine the particular solution
66: a=(-w^2*m+k+i*w*c)\(f1-i*f2);
67: yp=real(a*exp(i*w*t));
68: yp0=real(a); vp0=real(i*w*a);
69: end
70:
71: % Scale the homogeneous solution to satisfy the
72: % initial conditions
73: U=U*diag(U\[y0-yp0; v0-vp0]);
74: yh=real(U(1:n,:)*exp(lam*t));
75:
76: % Combine results to obtain the total solution
77: t=t(:); y=[yp+yh];
78:
79: % Show data graphically only for default case
80: if nargin==0
81: waterfall(t,(1:n),y'), xlabel('time axis')
82: ylabel('mass index'), zlabel('Displacements')
83: title([['DISPLACEMENT HISTORY FOR A ',...
84: int2str(n),'-MASS SYSTEM']])
85: colormap([1,0,0]), shg
86: end

Function findifco

1: function [c,e,m,crat]=findifco(k,a)
2: %
3: % [c,e,m,crat]=findifco(k,a)
4: % -----------------------------
5: % This function approximates the k'th derivative
6: % of a function using function values at n
7: % interpolation points. Let f(x) be a general
8: % function having its k'th derivative denoted
9: % by F(x,k). The finite difference approximation
10: % for the k'th derivative employing a stepsize h
11: % is given by:
12: % F(x,k)=Sum(c(j)*f(x+a(j)*h), j=1:n)/h^k +
% TruncationError
% with m=n-k being the order of truncation
% error which decreases like h^m and
% TruncationError=(h^m)\*[(e(1)\*F(x,n)+...
% e(2)\*F(x,n+1)*h+e(3)*F(x,n+2)*h^2+O(h^3))
%
% a - a vector of length n defining the
% interpolation points x+a(j)*h where
% x is an arbitrary parameter point
% k - order of derivative evaluated at x
% c - the weighting coefficients in the
% difference formula above. c(j) is
% the multiplier for value f(x+a(j)*h)
% e - error component vector in the above
% difference formula
% m - order of truncation order in the
% formula. The relation m=n-k applies.
% crat - a matrix of integers such that c is
% approximated by crat(1,:)/crat(2,:)

a=a(:); n=length(a); m=n-k; mat=ones(n,n+4);
for j=2:n+4; mat(:,j)=a/(j-1).*mat(:,j-1); end
A=pinv(mat(:,1:n)); e=-ec(k+1,:); %
% [ctop,cbot]=rat(c,1e-8); crat=[ctop(:)';cbot(:)'];

Function gcquad

function [val,bp,wf]=gcquad(func,xlow,...
    xhigh,nquad,mparts,varargin)

% [val,bp,wf]=gcquad(func,xlow,...
%     xhigh,nquad,mparts,varargin)

% -------------------------------
% This function integrates a general function using
% a composite Gauss formula of arbitrary order. The
% integral value is returned along with base points
% and weight factors obtained by an eigenvalue based
% method. The integration interval is divided into
% mparts subintervals of equal length and integration
% over each part is performed with a Gauss formula

© 2003 by CRC Press LLC
% making nquad function evaluations. Results are
% exact for polynomials of degree up to 2*nquad-1.
% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% func - name of a function to be integrated
% having an argument list of the form
% func(x,p1,p2,...) where any auxiliary
% parameters p1,p2,.. are passed through
% variable varargin. Use [ ] for the
% function name if only the base points
% and weight factors are needed.
% xlow,xhigh - integration limits
% nquad - order of Gauss formula chosen
% mparts - number of subintervals selected in
% the composite integration
% varargin - variable length parameter used to
% pass additional arguments needed in
% the integrand func
% val - numerical value of the integral
% bp,wf - vectors containing base points and
% weight factors in the composite
% integral formula

% A typical calculation such as:
% Fun=inline('(sin(w*t).^2).*exp(c*t)','t','w','c');
% A=0; B=12; nquad=21; mparts=10; w=10; c=8;
% [value,pcterr]=integrate(Fun,A,B,nquad,mparts,w,c);
% gives value = 1.935685556078172e+040 which is
% accurate within an error of 1.9e-13 percent.
% User m functions called: the function name passed
% in the argument list

% Compute base points and weight factors
% for the single interval [-1,1]. (Ref:
% 'Methods of Numerical Integration' by
% P. Davis and P. Rabinowitz, page 93)
% u=(1:nquad-1)./sqrt((2*(1:nquad-1)).^2-1);
% [vc,bp]=eig(diag(u,-1)+diag(u,1));
% [bp,k]=sort(diag(bp)); wf=2*vc(1,k)' .^2;
% Modify the base points and weight factors
to apply for a composite interval
d=(xhigh-xlow)/mparts;  d1=d/2;
dbp=d1*bp(:);  dwf=d1*wf(:);  dr=d*(1:mparts);
cbp=dbp(:,ones(1,mparts))+ ...
dr(ones(nquad,1),:)+(xlow-d1);
cwf=dwf(:,ones(1,mparts));  wf=cwf(:);  bp=cbp(:);

% Compute the integral
if isempty(func)
   val=[];
else
   f=feval(func,bp,varargin{:});  val=wf'*f(:);
end

Function gridview

function gridview(x,y,xlabl,ylabl,titl)

% gridview(x,y,xlabl,ylabl,titl)
% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% This function views a surface from the top
% to show the coordinate lines of the surface.
% It is useful for illustrating how coordinate
% lines distort in a conformal transformation.
% Calling gridview with no arguments depicts the
% mapping of a polar coordinate grid map under
% a transformation of the form
% z=R*(zeta+m/zeta).
%
% x,y - real matrices defining a
%   curvilinear coordinate system
% xlabl,ylabl - labels for x and y axes
% titl - title for the graph
%
% User m functions called: cubrange
%-----------------------------

if nargin<5
   xlabl='real axis';  ylabl='imaginary axis';

© 2003 by CRC Press LLC
26:    titl='';
27:    end
28:
29:    % Default example using z=R*(zeta+m/zeta)
30:    if nargin==0
31:        zeta=linspace(1,3,10)'; ...
32:        exp(i*linspace(0,2*pi,81));
33:        a=2; b=1; R=(a+b)/2; m=(a-b)/(a+b);
34:        z=R*(zeta+m./zeta); x=real(z); y=imag(z);
35:        titl=['Circular Annulus Mapped onto an ', ...
36:              'Elliptical Annulus'];
37:    end
38:
39:    range=cubrange([x(:),y(:)],1.1);
40:
41:    % The data define a curve
42:    if size(x,1)==1 | size(x,2)==1
43:        plot(x,y,'-k'); xlabel(xlabl); ylabel(ylabl);
44:        title(titl); axis('equal'); axis(range);
45:        grid on; figure(gcf);
46:        if nargin==0
47:            print -deps gridviewl;
48:        end
49:    % The data define a surface
50:    else
51:        plot(x,y,'k-',x',y','k-')
52:        xlabel(xlabl); ylabel(ylabl); title(titl);
53:        axis('equal'); axis(range); grid on;
54:        figure(gcf);
55:        if nargin==0
56:            print -deps gridview;
57:        end
58:    end
59:
60:    %==============================================
61:    function range=cubrange(xyz,ovrsiz)
62:    %
63:    % range=cubrange(xyz,ovrsiz)
64:    %
65:    % This function determines limits for a square
66:    % or cube shaped region for plotting data values
67:    % in the columns of array xyz to an undistorted
68:    % scale
69:    %

© 2003 by CRC Press LLC
% xyz - a matrix of the form [x,y] or [x,y,z] where x,y,z are vectors of coordinate points
% ovrsiz - a scale factor for increasing the window size. This parameter is set to one if only one input is given.
% range - a vector used by function axis to set undistorted. This vector has the form [xmin,xmax,ymin,ymax] when xyz has only two columns or the form [xmin,xmax,ymin,ymax,zmin,zmax] when xyz has three columns.
% User m functions called: none

%----------------------------------------------
if nargin==1, ovrsiz=1; end
pmin=min(xyz); pmax=max(xyz); pm=(pmin+pmax)/2;
pd=max(ovrsiz/2*(pmax-pmin));
if length(pmin)==2
  range=pm([1,1,2,2])+pd*[-1,1,-1,1];
else
  range=pm([1 1 2 2 3 3])+pd*[-1,1,-1,1,-1,1];
end

Function inputv

function varargout=inputv(prompt)
% [a1,a2,...,a_nargout]=inputv(prompt)
%~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% This function reads several values on one line. The items should be separated by commas or blanks.
% prompt - A string preceding the data entry. It is set to ' ? ' if no value of prompt is given.
% a1,a2,...,a_nargout - The output variables

© 2003 by CRC Press LLC
that are created. If
% not enough data values
% are given following the
% prompt, the remaining
% undefined values are
% set equal to NaN

% A typical function call is:
% [A,B,C,D]=inputv('Enter values of A,B,C,D: ')
%
%----------------------------------------------------------------------

if nargin==0, prompt=' ? '; end
u=input(prompt,'s'); v=eval(['[',u,']']);
ni=length(v); no=nargout;
varargout=cell(1,no); k=min(ni,no);
for j=1:k, varargout{j}=v(j); end
if no>ni
for j=ni+1:no, varargout{j}=nan; end
end

Function lintrp

function y=lintrp(xd,yd,x)
% y=lintrp(xd,yd,x)
% This function performs piecewise linear
% interpolation through data values stored in
% xd, yd, where xd values are arranged in
% nondecreasing order. The function can handle
% discontinuous functions specified when some
% successive values in xd are equal. Then the
% repeated xd values are shifted by a small
% amount to remove the discontinuities.
% Interpolation for any points outside the range
% of xd is also performed by continuing the line
% segments through the outermost data pairs.
% xd,yd - vectors of interpolation data values
% x      - matrix of values where interpolated
% values are required

© 2003 by CRC Press LLC
y - matrix of interpolated values

k = find(diff(xd) == 0);
if length(k) ~ = 0
    xd(k+1) = xd(k+1) + (xd(end) - xd(1)) * 1e3 * eps;
end
y = interp1(xd, yd, x, 'linear', 'extrap');

Function manyrts

function roots = manyrts(func, a, b, nsteps, ...
    maxrts, tol, varargin)
    %
    % roots = manyrts(func, a, b, nsteps, maxrts, tol, ...
    %     varargin)
    % ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    % This function attempts to find multiple roots
    % of a function by searching an interval in steps
    % of equal length and finding a root in each
    % interval where a sign change occurs
    % func       - name of a function of the form
    %             func(x, p1, p2, ...) where additional
    %             parameters after the first are
    %             passed through varargin
    % a, b       - upper and lower limits of the
    %             search interval
    % nsteps     - number of intervals from a to b
    %             which are checked to detect a
    %             sign change
    % maxrts     - maximum number of roots sought
    %             within the search limits. The
    %             search terminates when the number
    %             of roots found equals maxrts.
    % tol        - the root tolerance passed to
    %             function fzero. A default value of
    %             1e-10 is used if no value is given
    % varargin   - the cell variable provided to pass
    %             multiple arguments to function func

    if nargin < 6, tol = 1e-10; end;
    if nargin < 5, maxrts = 100; end
    if isstruct(tol), options = tol;
    else
options=optimset('tolfun',tol,'tolx',tol);
end
x=linspace(a,b,nsteps); roots=[];
rtlast=-realmax;
for j=1:nsteps-1
  xj=x(j); xj1=x(j+1);
  fj=feval(func,xj,varargin{:});
  fji=feval(func,xj1,varargin{:});
  if fj.*fji<=0
    rt=fzero(func,[xj,xj1],
                options,varargin{:});
    if (rt-rtlast)>tol
      roots=[roots,rt]; rtlast=rt;
    end
  end
if length(roots)==maxrts, break, end
end

Function membanim

function membanim(u,x,y,t)
%
% function membanim(u,x,y,t)
% ---------------------------
% This function animates the motion of a
% vibrating membrane
%
% u array in which component u(i,j,k) is the
% displacement for y(i),x(j),t(k)
% x,y arrays of x and y coordinates
% t vector of time values

% Compute the plot range
if nargin==0;
  [u,x,y,t]=memrecwv(2,1,15.5,1.5,.5,5);
end
xmin=min(x(:)); xmax=max(x(:));
ymin=min(y(:)); ymax=max(y(:));
xmid=(xmin+xmax)/2; ymid=(ymin+ymax)/2;
d=max(xmax-xmin,ymax-ymin)/2; Nt=length(t);
range=[xmid-d,xmid+d,ymid-d,ymid+d,...
      3*min(u(:)),3*max(u(:))];
while 1 % Show the animation repeatedly
    disp(' '), disp('Press return for animation')
    dumy=input('or enter 0 to stop > ? ','s');
    if ~isempty(dumy)
        disp(' '), disp('All done'), break
    end

    % Plot positions for successive times
    for j=1:Nt
        surf(x,y,u(:,:,j)), axis(range)
        xlabel('x axis'), ylabel('y axis')
        zlabel('u axis'), titl=sprintf('MEMBRANE POSITION AT T=%5.2f',t(j));
        title(titl), colormap([1 1 1])
        colormap([127/255 1 212/255])
        axis off
        drawnow, shg, pause(.1)
    end
end

Function plotmotn

function plotmotn(x,y,titl,isave)
    % plotmotn(x,y,titl,isave)
    % ---------------------
    % This function plots the cable time
    % history described by coordinate values
    % stored in the rows of matrices x and y.
    % x,y - matrices having successive rows
    % which describe position
    % configurations for the cable
    % titl - a title shown on the plots
    % isave - parameter controlling the form
    % of output. When isave is not input,
    % only one position at a time is shown
    % in rapid succession to animate the
    % motion. If isave is given a value,
    % then successive are all shown at
    % once to illustrate a kinematic
    % trace of the motion history.
% User m functions called: none
%---------------------------------------------------------------------

% Set a square window to contain all
% possible positions
[nt,n]=size(x);
if nargin==4, save =1; else, save=0; end
xmin=min(x(:)); xmax=max(x(:));
ymin=min(y(:)); ymax=max(y(:));
w=max(xmax-xmin,ymax-ymin)/2;
hold off; clf; axis('normal'); axis('equal');
title(titl)
xlabel('x axis'); ylabel('y axis')
if save==0
  for j=1:nt
    xj=x(j,:); yj=y(j,:);
    plot(xj,yj,'-k',xj,yj,'ok');
    axis(range), axis off
    title(titl)
    figure(gcf), drawnow, pause(.1)
  end
  pause(2)
else
  hold off; close
  for j=1:nt
    xj=x(j,:); yj=y(j,:);
    plot(xj,yj,'-k',xj,yj,'ok');
    axis(range), axis off, hold on
  end
  title(titl)
  figure(gcf), drawnow, hold off, pause(2)
end

% Save plot history for subsequent printing
% print -deps plotmotn

Function polhedrn

function [v,rc,vrr,irr]=polhedrn(x,y,z,idface)
% % [v,rc,vrr,irr]=polhedrn(x,y,z,idface)
This function determines the volume, centroidal coordinates and inertial moments for an arbitrary polyhedron.

\[
x, y, z - \text{vectors containing the corner indices of the polyhedron}
\]
\[
idface - \text{a matrix in which row } j \text{ defines the corner indices of the } j^\text{th} \text{ face.}
\]
\[
\text{Each face is traversed in a counterclockwise sense relative to the outward normal. The column dimension equals the largest number of indices needed to define a face. Rows requiring fewer than the maximum number of corner indices are padded with zeros on the right.}
\]
\[
v - \text{the volume of the polyhedron}
\]
\[
rc - \text{the centroidal radius}
\]
\[
vrr - \text{the integral of } R \times R^\top \times d(vol)
\]
\[
irr - \text{the inertia tensor for a rigid body of unit mass obtained from } vrr \text{ as}
\]
\[
\text{eye}(3,3) \times \text{sum(diag(vrr))} - vrr
\]

User m functions called: pyramid

---

```matlab
r=[x(:,),y(:,),z(,:)]; nf=size(idface,1); v=0; vr=0; vrr=0;
for k=1:nf
    i=idface(k,:); i=find(i>0);
    [u,ur,urr]=pyramid(r(i,:));
    v=v+u; vr=vr+ur; vrr=vrr+urr;
end
rc=vr/v; irr=eye(3,3)*sum(diag(vrr))-vrr;
```

---

Function polyxy

```matlab
function [area,xbar,ybar,axx,axy,ayy]=polyxy(x,y)
```

© 2003 by CRC Press LLC
% This function computes the area, centroidal coordinates, and inertial moments of an arbitrary polygon.
% x, y - vectors containing the corner coordinates. The boundary is traversed in a counterclockwise direction.
% area - the polygon area
% xbar, ybar - the centroidal coordinates
% axx - integral of x^2*dx*dy
% axy - integral of xy*dy
% ayy - integral of y^2*dx*dy
% User m functions called: none

n=1:length(x); n1=n+1;
% x=[x(:);x(1)]; y=[y(:);y(1)];
a=(x(n).*y(n1)-y(n).*x(n1))';
area=sum(a)/2; a6=6*area;
xbar=a*(x(n)+x(n1))/a6; ybar=a*(y(n)+y(n1))/a6;
axy=a*(x(n).*y(n1)+x(n1).*y(n))/24;
avv=a*(x(n).*y(n)+y(n1).*x(n))/24;
axx=a*(x(n).*y(n1)+x(n1).*y(n))/24;

Function quadlsqrt

function v=quadlsqrt(fname,type,a,b,tol,trace, varargin)
% v=quadlsqrt(fname,type,a,b,tol,trace, varargin)
% This function uses the MATLAB integrator quadl to evaluate integrals having square root type singularities at one or both ends of the integration interval a < x < b. The integrand has the form: func(x)/sqrt(x-a) if type==1.
% func(x)/sqrt(b-x) if type==2.
% func(x)/sqrt((x-a)*(b-x)) if type==3.
%
% func    - the handle for a function continuous
%           from x=a to x=b
% type     - 1 if the integrand is singular at x=a
%           2 if the integrand is singular at x=b
%           3 if the integrand is singular at both
%           x=a and x=b.
% a,b      - integration limits with b > a

if nargin<6 | isempty(trace), trace=0; end  
if nargin<5 | isempty(tol), tol=1e-8; end    
if nargin<7  
    varargin{1}=type; varargin{2}=[a,b];  
    varargin{3}=fname;                      
else  
    n=length(varargin); c=[a,b]; varargin{n+1}=type;  
    varargin{n+2}=c; varargin{n+3}=fname;          
end

if type==1 | type==2  
v=2*quadl(@fshift,0,sqrt(b-a),...  
tol,trace,varargin{:});              
else  
v=quadl(@fshift,0,pi,tol,trace,varargin{:}); 
end

%=========================================  
function u=fshift(x,varargin)  
% u=fshift(x,varargin) 
% This function shifts arguments to produce 
% a nonsingular integrand called by quadl 
% N=length(varargin); fname=varargin{N};  
% c=varargin{N-1}; type=varargin{N-2};  
% a=c(1); b=c(2); c1=(b+a)/2; c2=(b-a)/2;  
switch type  
case 1, t=a+x.^2; case 2, t=b-x.^2;  
case 3, t=c1+c2*cos(x);  
end

if N>3, u=feval(fname,t,varargin{1:N-3});  
else, u=feval(fname,t); end

© 2003 by CRC Press LLC
Function ratcof

function [a,b]=ratcof(xdata,ydata,ntop,nbot)

% Determine a and b to approximate ydata as a rational function of the variable xdata.
% The function has the form:
%
% y(x) = sum(1=>ntop) ( a(j)*x^(j-1) ) / ( 1 + sum(1=>nbot) ( b(j)*x^(j)) )
%
% xdata,ydata - input data vectors (real or complex)
% ntop,nbot - number of series terms used in the numerator and the denominator.

ydata=ydata(:); xdata=xdata(:);
m=length(ydata);
if nargin==3, nbot=ntop; end;
x=ones(m,ntop+nbot); x(:,ntop+1)=-ydata.*xdata;
for i=2:ntop, x(:,i)=xdata.*x(:,i-1); end
for i=2:nbot
  x(:,i+ntop)=xdata.*x(:,i+ntop-1);
end
ab=pinv(x)*ydata; %ab=x\ydata;
a=ab(1:ntop); b=ab(ntop+1:ntop+nbot);

Function raterp

function y=raterp(a,b,x)

% This function interpolates using coefficients from function ratcof.

© 2003 by CRC Press LLC
% a,b - polynomial coefficients from function ratcof
% x - argument at which function is evaluated
% y - computed rational function values

%--------------------------------------------------------------

a=flipud(a(:)); b=flipud(b(:));
y=polyval(a,x)./(1+x.*polyval(b,x));

Function smdsolve

function [x,v]=smdsolve(m,c,k,f1,f2,w,x0,v0,t)
    % [x,v]=smdsolve(m,c,k,f1,f2,w,x0,v0,t)
    % ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
    % This function solves the differential equation
    % m*x''(t)+c*x'(t)+k*x(t)=f1*cos(w*t)+f2*sin(w*t)
    % with x(0)=x0 and x'(0)=v0
    %
    % m,c,k - mass, damping and stiffness coefficients
    % f1,f2 - magnitudes of cosine and sine terms in
    % the forcing function
    % w - frequency of the forcing function
    % t - vector of times to evaluate the solution
    % x,v - computed position and velocity vectors

    ccrit=2*sqrt(m*k); wn=sqrt(k/m);
    % If the system is undamped and resonance will
    % occur, add a little damping
    if c==0 & w==wn; c=ccrit/1e6; end;

    % If damping is critical, modify the damping
    % very slightly to avoid repeated roots
    if c==ccrit; c=c*(1+1e-6); end

    % Forced response solution
    a=(f1-i*f2)/(k-m*w^2+i*c*w);
    X0=real(a); VO=real(i*w*a);
    X=real(a*exp(i*w*t)); V=real(i*w*a*exp(i*w*t));

    % Homogeneous solution
r = sqrt(c^2 - 4*m*k);
s1 = (-c + r)/(2*m); s2 = (-c - r)/(2*m);
p = [1,1;s1,s2]\[x0-X0;v0-V0];

% Total solution satisfying the initial conditions
x = X + real(p(1)*exp(s1*t) + p(2)*exp(s2*t));
v = V + real(p(1)*s1*exp(s1*t) + p(2)*s2*exp(s2*t));

Function splined

function val = splined(xd, yd, x, if2)

% val = splined(xd, yd, x, if2)
% ---------------------------
% This function evaluates the first or second
derivative of the piecewise cubic
% interpolation curve defined by the intrinsic
% function spline provided in MATLAB. If fewer
% than four data points are input, then simple
% polynomial interpolation is employed

% xd, yd - data vectors determining the spline
dp curve produced by function spline
% x - vector of values where the first or
% the second derivative are desired
% if2 - a parameter which is input only if
% y''(x) is required. Otherwise, y'(x)
is returned.
% val - the first or second derivative values
% for the spline

% User m functions called: none

m = length(xd); [b,c] = unmkpp(spline(xd, yd));
if n>3 % Use a cubic spline
  if nargin==3, c = [3*c(:,1),2*c(:,2),c(:,3)];
  else, c = [6*c(:,1),2*c(:,2)]; end
  val = ppval(mkpp(b,c),x);
else % Use a simple polynomial
  c = polyder(polyfit(xd(:,1), yd(:,1), n-1));
  if nargin==3, c = polyder(c); end

© 2003 by CRC Press LLC
val=polyval(c,x);
end

Function splineg

function [val,b,c]=splineg(xd,yd,x,deriv,endc,b,c)
% [val,b,c]=splineg(xd,yd,x,deriv,endc,b,c)
% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% For a cubic spline curve through data points xd,yd, this function evaluates y(x), y'(x),
% y''(x), or integral(y(x)*dx, xd(1) to x(j) )
% for j=1:length(x). The coefficients needed to
% evaluate the spline are also computed.
%
% xd,yd - data vectors defining the cubic spline curve
% x - vector of points where curve properties are computed.
% deriv - denoting the spline curve as y(x),
% deriv=0 gives a vector for y(x)
% deriv=1 gives a vector for y'(x)
% deriv=2 gives a vector for y''(x)
% deriv=3 gives a vector of values for integral(y(z)*dz) from xd(1)
% to x(j) for j=1:length(x)
% endc - endc=1 makes y'''(x) continuous at xd(2) and xd(end-1).
% endc=[2,left_slope,right_slope] imposes slope values at both ends.
% endc=[3,left_slope] imposes the left end slope and makes the discontinuity of y''' at xd(end-1) small.
% endc=[4,right_slope] imposes the right end slope and makes the discontinuity of y''' at xd(2) small.
% b,c coefficients needed to perform the spline interpolation. If these are not
given, function unmkpp is called to generate them.
% val values y(x),y'(x),y''(x) or
% integral(y(z)dz, z=xd(1)..x) for
% deriv=0, 1, 2, or 3, respectively.
% User m files called: splincof
% -------------------------------------------
if nargin<5 | isempty(endc), endc=1; end
if nargin<7, [b,c]=splincof(xd,yd,endc); end
n=length(xd); [N,M]=size(c);

switch deriv

case 0 % Function value
    val=ppval(mkpp(b,c),x);

case 1 % First derivative
    C=[3*c(:,1),2*c(:,2),c(:,3)];
    val=ppval(mkpp(b,C),x);

case 2 % Second derivative
    C=[6*c(:,1),2*c(:,2)];
    val=ppval(mkpp(b,C),x);

case 3 % Integral values from xd(1) to x
    k=M:-1:1;
    C=[c./k(ones(N,1),:),zeros(N,1)];
    dx=xd(2:n)-xd(1:n-1); s=zeros(n-2,1);
    for j=1:n-2, s(j)=polyval(C(j,:),dx(j)); end
    C(:,5)=[0;cumsum(s)]; val=ppval(mkpp(b,C),x);
end

%=============================================

function [b,c]=splincof(xd,yd,endc)
% [b,c]=splincof(xd,yd,endc)
% This function determines coefficients for cubic spline interpolation allowing four different types of end conditions.
% xd,yd - data vectors for the interpolation
% endc - endc=1 makes y''''(x) continuous at xd(2) and xd(end-1).
% endc=[2,left_slope,right_slope] imposes slope values at both ends.
% endc=[3,left_slope] imposes the left...
% end slope and makes the discontinuity
% of $y'''$ at $x_d(\text{end}-1)$ small.
% endc=[4,right_slope] imposes the right
% end slope and makes the discontinuity
% of $y'''$ at $x_d(2)$ small.
% if nargin<3, endc=1; end;
type=endc(1); xd=xd(:); yd=yd(:);
switch type
  case 1
    % $y'''(x)$ continuous at the $x_d(2)$ and $x_d(\text{end}-1)$
    [b,c]=unmkpp(spline(xd,yd));
  case 2
    % Slope given at both ends
    [b,c]=unmkpp(spline(xd,[endc(2);yd;endc(3)]));
  case 3
    % Slope at left end given. Compute right end
    % slope.
    [b,c]=unmkpp(spline(xd,yd));
    c=[3*c(:,1),2*c(:,2),c(:,3)];
    sright=ppval(mkpp(b,c),xd(end));
    [b,c]=unmkpp(spline(xd,[endc(2);yd;sright]));
  case 4
    % Slope at right end known. Compute left end
    % slope.
    [b,c]=unmkpp(spline(xd,yd));
    c=[3*c(:,1),2*c(:,2),c(:,3)];
    sleft=ppval(mkpp(b,c),xd(1));
    [b,c]=unmkpp(spline(xd,[sleft;yd;endc(2)]));
end

Function spterp

function [v,c]=spterp(xd,yd,id,x,endv,c)
% [v,c]=spterp(xd,yd,id,x,endv,c)
  % This function performs cubic spline interpo-
% lation. Values of y(x), y'(x), y''(x) or the
% integral(y(t)*dt, xd(1)...x) are obtained.
% xd, yd - data vectors with xd arranged in
% ascending order.
% id - id equals 0,1,2,3 to compute y(x),
% y'(x), integral(y(t)*dt,t=xd(1)...x),
% respectively.
% v - values of the function, first deriva-
% tive, second derivative, or integral
% from xd(1) to x
% c - the coefficients defining the spline
% curve.
% endv - vector giving the end conditions in
% one of the following five forms:
% endv=1 or endv omitted makes
% c(2) and c(n-1) zero
% endv=[2,left_end_slope,...
% right_end_slope] to impose slope
% values at each end
% endv=[3,left_end_slope] imposes the
% left end slope value and makes
% c(n-1) zero
% endv=[4,right_end_slope] imposes the
% right end slope value and makes
% c(2) zero
% endv=5 defines a periodic spline by
% making y,y',y'' match at both ends

if nargin<5 | isempty(endv), endv=1; end
n=length(xd); sx=size(x); x=x(:); X=x-xd(1);

if nargin<6, c=spcof(xd,yd,endv); end
C=c(1:n); s1=c(n+1); m1=c(n+2); X=x-xd(1);

if id==0 % y(x)
  v=yd(1)+s1*X+m1/2*X.*X+...
  powermat(x,xd,3)*C/6;
elseif id==1 % y'(x)
  v=s1+m1*X+powermat(x,xd,2)*C/2;
elseif id==2 % y''(x)
  v=m1+powermat(x,xd,1)*C;
else % integral(y(t)*dt, t=xd(1)..x)
  v=yd(1)*X+s1/2*X.*X+m1/6*X.^3+...
  powermat(x,xd,4)*C/24;

© 2003 by CRC Press LLC
end
v=reshape(v,sx);

%==============================================

function c=spcof(x,y,endv)
% c=spcof(x,y,endv)
% This function determines spline interpolation
% coefficients consisting of the support
% reactions concatenated with y' and y'' at
% the left end.
% x,y - data vectors of interpolation points.
% Denote n as the length of x.
% endv - vector of data for end conditions
% described in function spterp.
% c - a vector [c(1);...;c(n+2)] where the
% first n components are support
% reactions and the last two are
% values of y'(x(1)) and y''(x(1)).
if nargin<3, endv=1; end
x=x(:); y=y(:); n=length(x); u=x(2:n)-x(1);
a=zeros(n+2,n+2); a(1,1:n)=1;
a(2:n,:)=[powermat(x(2:n),x,3)/6,u,u.*u/2];
b=zeros(n+2,1); b(2:n)=y(2:n)-y(1);
if endv(1)==1 % Force, force condition
  a(n+1,2)=1; a(n+2,n-1)=1;
elseif endv(1)==2 % Slope, slope condition
  b(n+1)=endv(2); a(n+1,n+1)=1;
  b(n+2)=endv(3); a(n+2,:)=...
[(x(n)-x').^2]/2,1,x(n)-x(1)];
elseif endv(1)==3 % Slope, force condition
  b(n+1)=endv(2); a(n+1,n+1)=1; a(n+2,n-1)=1;
elseif endv(1)==4 % Force, slope condition
  a(n+1,2)=1; b(n+2)=endv(2);
  a(n+2,:)=[(x(n)-x').^2]/2,1,x(n)-x(1)];
elseif endv(1)==5
  a(n+1,1:n)=x(n)-x'; b(n)=0;
  a(n+2,1:n)=1/2*(x(n)-x').^2;
  a(n+2,n+2)=x(n)-x(1);
else
  error('Invalid value of endv in function spcof')
end
if endv(1)==1 & n<4,  c=pinv(a)*b;
else,  c=a\b;  end

function a=powermat(x,X,p)
% a=powermat(x,X,p)
% This function evaluates various powers of a
% matrix used in cubic spline interpolation.
% x,X - arbitrary vectors of length n and N
% a - an n by M matrix of elements such that
% a(i,j)=(x(i)>X(j))*abs(x(i)-X(j))^p
x=x(:);  n=length(x);  X=X(:)';  N=length(X);
a=x(:,ones(1,N))-X(ones(n,1),:);  a=a.*(a>0);
switch p,  case 0,  a=sign(a);  case 1,  return;
case 2,  a=a.*a;  case 3;  a=a.*a.*a;
case 4,  a=a.*a;  a=a.*a;  otherwise,  a=a.^p;  end

Function srfv

function [v,rc,vrr]=srfv(x,y,z)
% [v,rc,vrr]=srfv(x,y,z)
% --------------------------
% This function computes the volume, centroidal
% coordinates, and inertial tensor for a volume
% covered by surface coordinates contained in
% arrays x,y,z
% x,y,z - matrices containing the coordinates
% of a grid of points covering the
% surface of the solid
% v - volume of the solid
% rc - centroidal coordinate vector of the
% solid
% vrr - inertial tensor for the solid with the
% mass density taken as unity
% User functions called: scatripl proptet
%-----------------------------------------------
% p=inline(...
% 'v*(eye(3)*(r(:)'','r(:)*r(:)''-r(:)*r(:)''','v','r');
%d=mean([x(:,),y(:,),z(:)])
%x=x-d(1); y=y-d(2); z=z-d(3);

[n,m]=size(x); i=1:n-1; I=i+1; j=1:m-1; J=j+1;
xij=x(i,j); yij=y(i,j); zij=z(i,j); xIj=x(I,j); yIj=y(I,j); zIj=z(I,j);
xIJ=x(I,J); yIJ=y(I,J); zIJ=z(I,J);
xIj=x(i,J); yiJ=y(i,J); ziJ=z(i,J);

% Tetrahedron volumes
v1=scatripl(xij,yij,zij,xIj,yIj,zIj,xIJ,yIJ,zIJ);
v2=scatripl(xij,yij,zij,xIJ,yIJ,zIJ,xIj,yiJ,ziJ);
v=sum(sum(v1+v2));

% First moments of volume
X1=xij+xIj+xIJ; X2=xij+xIJ+xiJ;
Y1=yij+yIj+yIJ; Y2=yij+yIJ+yIj;
Z1=zij+zIj+zIJ; Z2=zij+zIJ+ziJ;
vx=sum(sum(v1.*X1+v2.*X2));
vy=sum(sum(v1.*Y1+v2.*Y2));
vz=sum(sum(v1.*Z1+v2.*Z2));

% Second moments of volume
vrr=proptet(v1,xij,yij,zij,xIj,yIj,zIj,...
 xIJ,yIJ,zIJ,X1,Y1,Z1)+...
 proptet(v2,xij,yij,zij,xIJ,yIJ,zIJ,...
 xIj,yiJ,zij,X2,Y2,Z2);
rc=[vx,vy,vz]/v/4; vs=sign(v);
v=abs(v)/6; vrr=vs*vrr/120;
vrr=[vrr([1 4 5]), vrr([4 2 6]), vrr([5 6 3])'];

rc=eye(3,3)*sum(diag(vrr))-vrr;
%vrr=vrr-p(v,rc)+p(v,rc+d); rc=rc+d;

Function strdynrk

function [t,x,v]=strdynrk(t,x0,v0,m,c,k,functim)
% [t,x,v]=strdynrk(t,x0,v0,m,c,k,functim)
% This function uses ode45 to solve the matrix
% differential equation: M*X''+C*X'+K*X=F(t)
% t - vector of solution times
% x0,v0 - initial position and velocity vectors
% m,c,k - mass, damping and stiffness matrices
% functim - character name for the driving force
% x,v - arrays containing solution values for
% position and velocity

% A typical call to strdynrk function is:
% m=eye(3,3); k=[2,-1,0;-1,2,-1;0,-1,2];
% c=.05*k; x0=zeros(3,1); v0=zeros(3,1);
% t=linspace(0,10,101);
% [t,x,v]=strdynrk(t,x0,v0,m,c,k,’func’);

global Mi C K F n n1 n2
Mi=inv(m); C=c; K=k; F=functim;
n=size(m,1); n1=1:n; n2=n+1:2*n;
[t,z]=ode45(@sde,t,[x0(:);v0(:)]);
x=z(:,n1); v=z(:,n2);

function zp=sde(t,z)
% zp=sde(t,z)
% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
global Mi C K F n n1 n2
zp=[z(n2); Mi*(feval(F,t)-C*z(n2)-K*z(n1))];
%================================

function f=func(t)
% f=func(t)
f=[-1;0;2]*sin(1.413*t);
%================================

Function surf2surf

function [d,r,R]=surf2surf(x,y,z,X,Y,Z,n)
% [d,r,R]=surf2surf(x,y,z,X,Y,Z,n)
% ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
% This function determines the closest points on two
% surfaces and the distance between these points. It
% is similar to function srf2srf except that large
arrays can be processed.

% x,y,z - arrays of points on the first surface
% X,Y,Z - arrays of points on the second surface
% d - the minimum distance between the surfaces
% r,R - vectors containing the coordinates of the nearest points on the first and the second surface
% n - length of subvectors used to process the data arrays. Sending vectors of length n to srf2srf and taking the best of the subresults allows processing of large arrays of data points

% User m functions used: srf2srf

if nargin<7, n=500; end
N=prod(size(x)); M=prod(size(X)); d=realmax;
kN=max(1,floor(N/n)); kM=max(1,floor(M/n));
for i=1:kN
  i1=1+(i-1)*n; i2=min(i1+n,N); i12=i1:i2;
  xi=x(i12); yi=y(i12); zi=z(i12);
  for j=1:kM
    j1=1+(j-1)*n; j2=min(j1+n,M); j12=j1:j2;
    [dij,rij,Rij]=srf2srf(...
      xi,yi,zi,X(j12),Y(j12),Z(j12));
    if dij<d, d=dij; r=rij; R=Rij; end
  end
end

%=======================================================================

function [d,r,R]=srf2srf(x,y,z,X,Y,Z)
% [d,r,R]=srf2srf(x,y,z,X,Y,Z)
% This function determines the closest points on two surfaces and the distance between these points.
% x,y,z - arrays of points on the first surface
% X,Y,Z - arrays of points on the second surface
% d - the minimum distance between the surfaces
% r,R - vectors containing the coordinates of the nearest points on the first and the second surface
x=x(:); y=y(:); z=z(:); n=length(x); v=ones(n,1);
X=X(:)'; Y=Y(:)'; Z=Z(:)'; N=length(X); h=ones(1,N);
d2=(x(:,h)-X(v,:)).^2; d2=d2+(y(:,h)-Y(v,:)).^2;
d2=d2+(z(:,h)-Z(v,:)).^2;
[u,i]=min(d2); [d,j]=min(u); i=i(j); d=sqrt(d);
r=[x(i);y(i);z(i)]; R=[X(j);Y(j);Z(j)];

Function surfmany

function surfmany(varargin)

%function surfmany(x1,y1,z1,x2,y2,z2,...
% x3,y3,z3,...xn,yn,zn)
% This function plots any number of surfaces
% on the same set of axes without shape
% distortion. When no input is given, then a
% six-legged solid composed of spheres and
% cylinders is shown.
%
% User m functions called: none
%-----------------------------------------------

if nargin==0

% Default data for a six-legged solid
n=10; rs=.25; d=7; rs=2; rc=.75;
[xs,ys,zs]=sphere; [xc,yc,zc]=cylinder;
xs=rs*xs; ys=rs*ys; zs=rs*zs;
xc=rc*xc; yc=rc*yc; zc=2*d*zc-d;
x1=xs; y1=ys; z1=zs;
x2=zs+d; y2=ys; z2=xs;
x3=zs-d; y3=ys; z3=xs;
x4=xs; y4=zs-d; z4=ys;
x5=xs; y5=zs+d; z5=ys;
x6=xs; y6=ys; z6=zs+d;
x7=xs; y7=ys; z7=zs-d;
x8=xc; y8=yc; z8=zc;
x9=zc; y9=xc; z9=yc;
x10=yc; y10=zc; z10=xc;
varargin={x1,y1,z1,x2,y2,z2,x3,y3,z3,...
 x4,y4,z4,x5,y5,z5,x6,y6,z6,x7,y7,z7,...
 x8,y8,z8,x9,y9,z9,x10,y10,z10};
end

% Find the data range
n=length(varargin);
Function volrevol

% This function computes geometrical properties
% for a volume of revolution resulting when a
% closed curve in the (x,z) plane is rotated,
% through given angular limits, about the z axis.
% The cross section of the volume is defined by
% a spline curve passed through data points

function [v,rg,Irr,X,Y,Z,aprop,xd,zd,kn]=...
volrev(xd,zd,kn,th,nth,noplot)

% [v,rg,Irr,X,Y,Z,aprop,xd,zd,kn]=...
% volrev(xd,zd,kn,th,nth,noplot)
% -----------------------------
% (xd,zd) in the same manner as was done in
% function areaprop for plane areas.

% xd,zd - data vectors defining the spline
% interpolated boundary, which is
% traversed in a counterclockwise
% direction
% kn - indices of any points where slope
% discontinuity is allowed to turn
% sharp corners
% p - vector of volume properties containing
% [v, xcg, ycg, zcg, vxx, vyy, vzz,...
% vxy, vyz, vzx] where v is the volume,
% (xcg,ycg,zcg) are coordinates of the
% centroid, and the remaining properties
% are volume integrals of the following
% integrand:
% [x.^, y.^2, z.^2, xy, yz, zx]*dxdyxz
% X,Y,Z - data arrays containing points on the
% surface of revolution. Plotting these
% points shows the solid volume with
% the ends left open. Function fill3
% is used to plot the surface with ends
% closed
% aprop - a vector containing properties of the
% area in the (x,z) plane that was used
% to generate the volume. aprop=[area,...
% xcentroidal, ycentroidal, axx, axz, azz].

% User m functions called: rotasurf, gcquad,
% curve2d, anglefun, splined
%----------------------------------------------------------------------
if nargin==0
    t1=-pi:pi/6:0; t2=0:pi/6:pi;
    Zd=[0,exp(i*t1),1/2+i+exp(i*t2)/2,0,-1];
    xd=real(Zd)+4; zd=imag(Zd);
    kn=[1,2,8,9,15,16];
    th=[-pi/2,pi]; nth=31;
end

% Plot a surface of revolution based on the
% input data points
if nargin==6
    [X,Y,Z]=rotasurf(xd,zd,th,nth,1);
else

© 2003 by CRC Press LLC
% Obtain base points and weight factors for the composite Gauss formula of order seven used in the numerical integration

nd=length(xd); nseg=nd-1;
[dum,bp,wf]=gcquad([],1,nd,7,nseg);

% Evaluate complex points and derivative values on the spline curve which is rotated to form the volume of revolution
[u,uplot,up]=curve2d(xd,zd,kn,bp);
% plot(real(uplot),imag(uplot)), axis equal,shg
x=real(u); dx=real(up); z=imag(u);
dz=imag(up); da=x.*dz-z.*dx;

% Evaluate line integrals for area properties
p=[ones(n,1), x, z, x.^2, x.*z, z.^2, x.^3,...
   (x.^2).*z, x.*(z.^2)].*repmat(da,1,9);
p=(wf(:)'*p)./[2 3 3 4 4 4 5 5 5];

% Scale area properties by multipliers involving the rotation angle for the volume
f=anglefun(th(2))-anglefun(th(1));
v=f(1)*p(2); rg=f([2 3 1]).*p([4 4 5])/v;

vrr=[f([4 5 2]); f([5 6 3]); f([2 3 1])].*
    [p([7 7 8]); p([7 7 8]); p([8 8 9])];
Irr=eye(3)*sum(diag(vrr))-vrr;
aprop=[p(1),p(2:3)/p(1),p(4:6)];
References


portable linear algebra library for high performance computers. In *Super-

linear algebra subprograms. Technical report, Argonne National Laboratory,

and linear algebra operations. Technical Report CS-90-122, University of
Tennessee Computer Science Department, Knoxville, Tennessee, December
1990.

linear algebra subprograms. *ACM Transactions on Mathematical Software*,
December 1989.

[33] J. J. Dongarra, J. Du Croz, S. Hammarling, and R. Hanson. An extended set of
fortran basic linear algebra subprograms. *ACM Transactions on Mathematical


properties for complex geometries. *Computers in Mechanical Engineering*,


[38] G. Forsythe and C. B. Moler. *Computer Solution of Linear Algebraic Systems*.


tem Routines — EISPACK Guide Extension*, volume 51 of *Lecture Notes in


© 2003 by CRC Press LLC


© 2003 by CRC Press LLC


© 2003 by CRC Press LLC


