HYDRAN-XR
Command Reference

Hydrodynamic Response Analysis
with
Integrated Structural Finite Element Analysis

Version 24.0

NumSoft Technologies

1. Index of Commands

The following listing of commands is available on-line via the commands `index hydrodynamics`, `index general`, `index database`, `index matrix`, `index functions`, `index fe_commands`, `index_fe_elements`, and `index misc`. All categories are printed by the single command `index`.

1.1. Hydrodynamic Commands

hyd_analysis	added mass, hydrodynamic damping, exciting forces
hyd_analysis_response	wave-induced hydroelastic response
hyd_analysis_response_drag	wave-induced hydroelastic response
hyd_analysis_response_P	external force hydroelastic response
hyd_assign_mooring	assign mooring stiffness to a body
hyd_body_check	check body data
hyd_close_files	close HYDRAN-XR output files
hyd_convert_fea_mesh	convert FEA mesh to hydrodynamic panel mesh
hyd_coordaxs	specify coordinate axes
hyd_coord_trans	transform input coordinates to inertial coordinates
hyd_export_graphics	export hydro. panel mesh to graphics program
hyd_export_graphics_th	export time history motion to graphics program
hyd_flex_modes	input flexible modes
hyd_genmodes	transform to generalized coordinates
hyd_irregular	short-term extreme response
hyd_modal_pressure	print exciting and modal pressures
hyd_mooring_line	input mooring line data
phyd_mooring_line	print mooring line data
hyd_mooring_stiffness	input mooring stiffness
phyd_mooring_stiffness	print mooring stiffness
hyd_nodes	input nodal coordinates
phyd_nodes	print nodal coordinates
hyd_node_gen	node generation
hyd_node_tolerance	check nodes for still water and symmetry planes
hyd_panel	define 4-node (or 3-node) panel elements
phyd_panel	print panel elements
hyd_panel_rmap	create reverse mapping of panel numbers
hyd_parameters	input global control parameters
hyd_postresponse	obtain some post-processed responses
hyd_postresponse_P	obtain some post-processed responses
hyd_rigid_modes	generate rigid body modes
hyd_rmass	input mass matrix for user modes
hyd_surf_elevation	determine the "free" surface elevation
hyd_surf_nodes	input surface nodal coordinates
phyd_surf_nodes	print surface nodal coordinates
hyd_surf_node_gen	surface node generation
hyd_surf_node_tolerance	check surface nodes
hyd_surf_panel	define 4-node (or 3-node) surface elements
phyd_surf_panel	print surface elements
hyd_tf	calculate transfer functions
hyd_velocity	calculate fluid velocity at user-specified points
hyd_velocity_nodes	define velocity point coordinates
phyd_velocity_nodes	print coordinates
hyd_velocity_node_gen	generate velocity point coordinates
hyd_velocity_node_tolerance	check velocity nodes
hyd_wave	input wave frequencies and incidence angles

hyd_wave_dispersion	solve for wave length, wave number
hyd_wave_spectra	input wave spectra
phyd_wave_spectra	print wave spectra
hyd_wet	estimate "wet" natural frequencies

1.2. General Commands

break_loop	break do/while loop
date	prints the current date and time
do	do loop
filein	read commands from file
flush	flush the buffer for a file
help	on-line help
if	conditional if
index	index of commands by category
logfile	turn log file on/off
login	read commands from a log file
name?	echo current project name
new_project	start a new project (also newproj or newprob)
palias	print command aliases
quit	quit MANOA
read	read an array from file
rename_file	rename a file
return	return from batch mode
rm_file	remove a file
savequit	save database and quit
system_command	execute a system command
time	returns the system seconds
while	conditional while
write	write an array to file

1.3. Database Commands

clear	initialize the database
ls or dir	short listing of arrays in database
ll	long listing of arrays in database
memory	report memory used by database
mv or rename	rename an array
readdb	read a database from file
rm or del	remove an array from the database
rm* or del*	remove members from the database
save	save the database to file

1.4. Matrix Commands

add	matrix add
arpack	eigenvalue solver
array3d_slice	slice a 3-D array
array3d_unslice	insert a matrix into a 3-D array
cp	copy
cpdg	copy diagonal elements
diag_mult	multiply by a diagonal matrix
dim_reduce	reduce dimensions of a matrix
eigval	eigenvalue solver
extract	extract rows of a matrix

fft	compute Fourier or inverse Fourier transform
fft_helper	multiply transfer functions with Fourier coefficients
ftopro	full to profile storage
gauss	Gauss elimination
get_dim	get dimensions of a matrix
ident	create real identity matrix
identc	create complex identity matrix
identi	create integer identity matrix
input	input a real matrix
inputc	input a complex matrix
inputch	input a character matrix
inputi	input an integer array
input3d	input a real 3-D array
input3dc	input a complex 3-D array
inputi	input an integer 3-D array
interpolate	interpolate discrete data
invert	invert a square matrix
jacobi	jacobi eigenvalue solver
joinh	join two matrices horizontally
joinv	join two matrices vertically
max	find the maximum in a matrix column
min	find the minimum in a matrix column
mult	matrix multiply
mult_col	multiply columns of two matrices
mult_elem	multiply elements of two matrices
norm	matrix norm
pmult	matrix multiply for profile storage
print	print a matrix
printf	print a matrix in F format
printi	print a matrix in integer format
psolve	profile equation solver
psolve16	profile equation solver for real*16
ptoful	convert profile to full storage
ptosparse	convert profile to sparse storage
put	put matrix inside another
putdg	put vector on diagonal of a matrix
scale	scale an array
series	create a 1d series of real data
series2d	create a 2d series of real data
seti	set a character string
seti	set an integer scalar
seti	set a real scalar
sort	sort a matrix
sparse_matrix_clean	remove sparse matrix storage
sparse_mult	matrix multiply for sparse storage
split	split a matrix based on columns
sub	matrix subtraction
subcol	subtract columns of two matrices
sumcol	sum columns of a matrix
tmult	multiply by transpose
to_complex	convert to complex
to_complex2	convert to complex with two arguments
to_int	convert to integer
to_real	convert to real
to_real16	convert to real*16
to_vector	convert to vector
trans	matrix transpose
unsplit	unsplit a matrix

unwrap	unwrap the rows of a matrix
wrap	wrap the rows of a matrix
xprint	extended print
zero	zero a real matrix
zeroc	zero a complex matrix
zeroi	zero an integer matrix

1.5. Mathematical Functions

abs	absolute value of matrix elements
acos	arccosine of matrix elements
asin	arcsine of matrix elements
atan	arctangent of matrix elements
bessel_j or bessel_y	Bessel function of 1st/2nd kind of matrix elements
conjugate	complex conjugate of matrix elements
cosine	cosine of matrix elements
cosh	hyperbolic cosine of matrix elements
epsilon	returns a small value relative to 1
erf	error function
erfc (see erf)	complementary error function
erfc_scaled (see erf)	scaled complementary error function
exp	exponential of matrix elements
gamma	gamma function of matrix elements
log	natural log of matrix elements
log10	common log of matrix elements
pi	creates a scalar with the value of pi
power	power of matrix elements
sine	sine of matrix elements
sinh	hyperbolic sine of matrix elements
sqrt	square root of matrix elements
tan	tangent of matrix elements
tanh	hyperbolic tangent of matrix elements

1.6. Finite Element Commands

bcid	displacement boundary conditions
body_frc2d	define 2D body forces
check_diag	check diagonals of K for zero
conc_deck_loads	pontoon bridge concentrated deck loads
consolidation	carry out consolidation analysis
cp_tables	input tables of drag coefficient Cp
current_velocity	input current velocity
dampers	nodal dampers
direct_th	determines dynamic time history response
disp_cntl	displacement control
distr_deck_loads	pontoon bridge distributed deck loads
el_iso_matl	elastic, isotropic material matrix
elem_alias	print element aliases
elem_grp	add/delete element group
elemXX	input for element type XX
eq_direction	generate earthquake direction vector
export_graphics	export to graphics program
fem_error	estimate finite element error
form_G	form element matrices for smoothing constraint
form_c	form global damping
form_k	form global stiffness

form_m	form global mass
form_lagrangeG	form element matrices for Lagrange constraint
imposed_displ	imposed nodal displacements
initial_conditions	specify initial conditions for time history
load_summary	print load summary
lsolve	linear equation solver
mass	nodal mass definition
mass_summary	summary of total structural mass
merge_nodes	merge coincident nodes
modal_th	determines dynamic modal response
nodal_constraint	impose nodal constraints
nodal_disp	arrange displacements on node basis
nodal_pressure	nodal pressure definition
nodef	nodal load definition
nodes	node definition
node_gen	generate nodes
node_order	orders nodes for equation numbering
node_str	average nodal stresses
nsolve	nonlinear equation solver
num_eqs	number equations
pbcid	print nodal restraints
pbody_frc2d	print 2D body forces
pcurrentvelocity	print current velocity
pdampers	print nodal dampers
pdeck_loads	print pontoon bridge deck loads
pdisp	print displacements
pelemXX	print data for element type XX
peqns	print equation numbers
pimposed_displ	print imposed nodal displacements
pmass	print input nodal masses
pndisp	print nodal displacements
pndisp_th	print time history of nodal displacements
pnodef	print input nodal loads
pnodes	print nodes
presponse	print element response
pstate	print element state
response	determine element response
rigid_modes	generate rigid body modes
state	determine element state
water_waves	input water waves

1.7. Finite Elements

beam3d	linear, 3-D beam element
biot1d234	1-D element for linear, elastic consolidation
biot2d3to9	2-D element for linear, elastic consolidation
cable	elastic catenary cable
contact_spring	nonlinear, contact spring
d1l234	1-D, linear, elastic element
d1l234v2	1-D, linear, elastic element, v.2
d2l3to9	2-D, linear elastic element
d2ltri	2-D, linear elastic triangular element
iFEM2D	2-D, 3 to 9 node element for nonlinear iFEM
interface	quadrilateral (and triangular) interface element
isomin6	linear, triangular Mindlin plate
min3s	Mindlin 3-D triangular, linear shell
min5s	Mindlin 3-D quadrilateral, linear shell

min4t	quadrilateral, linear, Mindlin shell
min6	linear, triangular Mindlin plate
nbeam2d	large displacement, elastic 2D beam
pbridge	elastic, 3-D pontoon bridge element
ntruss	large displacement truss
smth1c	1-D cubic smoothing element
smth1l	1-D discrete least squares smoothing element
smth1q	1-D quadratic smoothing element
smth2l	2-D discrete least squares smoothing element
smth2q	2-D quadratic, triangular smoothing element
smthspr	2-D node SPR smoothing element
spring	nonlinear, elastic spring
stiff2n	linear elastic 2-node stiffness element
truss	linear truss

1.8. Miscellaneous Commands

fortran_kind	prints the number of bytes for standard types
gauss_int	Gauss integration
gauss_pts	Gauss integration points
poly	evaluate a 1-D or 2-D polynomial
tri_intpts	integration points for triangle
userf	user-defined functions

2. Command Reference

2.1. Hydrodynamic Commands

hyd_analysis

Command to determine the added mass, damping, and wave excitation forces

Command Syntax

```
hyd_analysis [#added_freqs=? -periods]  [-no_pot]  [-source]
```

#added_freqs is the number of wave frequencies to add to a previous analysis. The frequencies, in radians/sec, are to be listed on the input line immediately following this command. If -period is specified, the input values are interpreted to be wave periods. The frequencies will be inserted in numerical sequence into the frequencies previously specified by the hyd_wave command.

If -no_pot is specified, the velocity potentials are not saved. WARNING: Although this option saves memory in the database, specifying -no_pot means that the potentials and/or pressures cannot be calculated in the hyd_analysis_response command.

If -source is specified, the radiation source strengths are written to file *.rad by the Fortran statement

```
write(f_rad)omega, strength
```

and the diffraction source strengths are written to file *.dif by the Fortran statement

```
write(f_dif)omega, angle, strength
```

For the latter, the outer loop is on the wave frequency. The * in the above file names represents the project name. Contact NumSoft Technologies for more complete details regarding the information written to these files. Note that this option is not compatible at this time with the #added_freqs option.

Some basic commands to carry out the hydrodynamic analysis can inlude:

```
hyd_parameters
hyd_coordaxs
hyd_wave
hyd_nodes
hyd_coord_trans
hyd_panel
hyd_body_check
hyd_rigid_modes
hyd_rmass
hyd_flex_modes
hyd_genmodes
hyd_analysis
hyd_analysis_response
hyd_postreponse
```

Other commands can be used, as needed. In addition, the sequence of the commands as given above may be altered somewhat, but this command must precede the hyd_analysis_response command. The hyd_nodes command (and the hyd_coord_trans command, if it is required) must precede the hyd_panel command. Also, hyd_rigid_modes, if used, must be given before hyd_rmass and hyd_flex_modes. The hyd_flex_modes command is not needed to carry out the hydrodynamic analysis of single or multiple rigid bodies.

The hydrodynamic analysis is based on linear potential theory. The Green function method with constant source strengths over each fluid panel is used to solve for the hydrodynamic variables. All calculations are carried out in double precision, except the Green functions are evaluated to single precision accuracy and the solutions of the equations to obtain the radiation and diffraction potentials are carried out in single precision arithmetic. The program is applicable for both infinite and finite water depths. Results from this command are the added mass, hydrodynamic damping, and generalized wave exciting forces.

Single and double symmetry of the structure are exploited by using the composite source distribution method. For single symmetry, the panels should be generated corresponding to the y > 0 region. For double symmetry, the panels should be generated on the x > 0, y > 0 region of the body.

The results are written to files, whose names are the project name with an extension. In addition to the basic *.out file, the following files may be created:

*.pan --> panel data
*.adm --> added mass coefficients
*.adp --> hydrodynamic damping coefficients
*.exf --> generalized wave excitation forces
*.ot2 --> additional output
*.pot --> velocity potentials at panel centers

The following main arrays are created in the database:

.hyd_modesym(nmode) -> symmetry code for modes
.hyd_twnk(nfreq,2) -> wave period, wave number
.hyd_un(nmode,npanel) -> generalized normals
.hyd_xr(npanel,4) -> x coordinates of panels
.hyd_yr(npanel,4) -> y coordinates of panels
.hyd_zr(npanel,4) -> z coordinates of panels
.hyd_xyzc(npanel,3) -> x,y,z coordinates of panel centers
.hyd_panela(npanel) -> area of fluid panels
.hyd_paneln(3,npanel) -> x,y,z components of panel normals
.hyd_addm(nmode,nmode,nfreq) -> added mass coefficients
.hyd_damp(nmode,nmode,nfreq) -> hydrodynamic damping coefficients
.hyd_uz(nmode,npanel) -> vertical, z, displacements
.hyd_fd(nmode,nbeta,nfreq) -> generalized diffraction forces
.hyd_fi(nmode,nbeta,nfreq) -> generalized Froude-Krylov forces
.hyd_rhs(nmode) -> RHS of generalized equations
.hyd_dynk(nmode,nmode) -> dynamic stiffness (LHS)
.hyd_gencor(nmode,nfreq,nbeta) -> generalized coordinates
.hyd_potincm(npanel,nfreq,nbeta) -> incoming potentials
.hyd_potdiff(npanel,nfreq,nbeta) -> diffraction potentials
.hyd_potrad(npanel,nfreq,nmode) -> radiation potentials

where nmode is the number of modes, npanel is the number of active panels, nfreq is the number of wave frequencies, and nbeta is the number of wave angles. These quantities are specified by the hyd_parameters and hyd_panel commands. Additional, temporary, arrays are also created and deleted. They all begin with either ".hyd_" or "$". To avoid a collision in array names, the user should not create any other array with these prefixes.

See Also

  hyd_analysis_response hyd_panel hyd_parameters

hyd_analysis_response

Command to determine the wave-induced response

Command Syntax

```
hyd_analysis_response  [-residual] [symmetrize=on|off]
```

This command solves the equations of motion to determine the generalized response based on the added mass, damping, and exciting forces calculated by the hyd_analysis command, which must precede this command.

If -residual is specified, the residual error in the solution of the equations of motion is calculated. The maximum norm of the error is reported.

If symmetrize=on, the added mass and hydrodynamic damping matrices are symmetrized by averaging the corresponding off-diagonal terms. If symmetrize=off, no averaging is done (default). Theoretically, these matrices should be symmetric. However, because of discretization errors, some corresponding terms may not be symmetric. The unsymmetry reduces as the mesh is refined.

This command can be executed any number of times, for example, to evaluate changing the moments of inertia of a body. The hyd_analysis command does not need to be repeated as long as the added mass, hydrodynamic damping, and exciting forces are not affected. Before this command, the structural mass matrix must be defined as a database member and named hyd_mstr. For single or multiple rigid bodies, this matrix is created easily with the hyd_rmass command. If a viscous structural damping matrix (hyd_cstr) and a structural stiffness matrix (hyd_kstr) are not defined, they will be created and zeroed. (Constant structural hysteretic damping can be easily specified via the hyd_parameters command.) The dimensions of the structural matrices should be (nmode,nmode), where nmode is the number of modes. The hyd_flex_modes command is not needed to carry out the hydrodynamic analysis of single or multiple rigid bodies.

The results are written to files, whose names are the project name with an extension. In addition to the basic *.out file, the following file will be created:

```
 *.cor  --> generalized coordinates
```

The following main arrays are created in the database:

```
.hyd_rhs(nmode)                  -> RHS of generalized equations
.hyd_dynk(nmode,nmode)           -> dynamic stiffness (LHS)
.hyd_gencor(nmode,nfreq,nbeta)   -> generalized coordinates
```

where nmode is the number of modes, nfreq is the number of wave frequencies, and nbeta is the number of wave angles. Additional, temporary, arrays are also created and deleted. They all begin with either ".hyd_" or "$". To avoid a collision in array names, the user should not create any other array with these prefixes.

See Also

hyd_analysis hyd_postresponse

hyd_analysis_response_drag

Command to determine the wave-induced response including linearized quadratic drag

Command Syntax

```
hyd_analysis_response_drag  arg1  arg2  beta=?  conv=?  [maxit=?]
                            [wave=?]  [symmetrize=on|off]
```

This command solves the equations of motion to determine the generalized response based on the added mass, damping, and exciting forces calculated by the hyd_analysis command, which must precede this command. The command linearizes the Morison-type drag term as described below. Only one wave angle is considered.

arg1 is the vector of diagonal terms in the "viscous" damping matrix that is used for the v^2 drag term. Note that this matrix is required to be diagonal, and hence only the diagonal values must be specified. The vector can be either real or complex.

arg2 is the modal transformation matrix that is used to transform structural mass, damping and stiffness matrices from physical displacements to modal coordinates.

beta is the integer wave angle number (not the wave angle) to be used (default=1)

conv is the convergence criterion

maxit is the maximum number of iterations (default=10)

wave is the wave amplitude (default = 1)

If symmetrize=on, the added mass and hydrodynamic damping matrices are symmetrized by averaging the corresponding off-diagonal terms. If symmetrize=off, no averaging is done (default). Theoretically, these matrices should be symmetric. However, because of discretization errors, some corresponding terms may not be symmetric. The unsymmetry reduces as the mesh is refined.

The equations of motion are of the form

$$[-w^2 M + i w (C + C_d) + K]q = P + P_m$$

M and K are the total mass and stiffness matrices (structure + fluid), C is the structure plus radiation damping, C_d is the linearized drag term, P is the linearized exciting forces, and P_m are the exciting forces from the Morison-type drag term. q are the modal coordinates. This equation is identical to the equations solved by hyd_analysis_response, except for the terms C_d and P_m.

C_d is obtained as follows. The diagonal damping matrix represented by arg1 is multiplied with (v-u) to obtain effective damping terms. v is the velocity at each structural degree-of-freedom based on the incoming wave, and u is the corresponding structural velocity from the previous iteration. The modified diagonal damping matrix is then transformed to modal coordinates using arg2, to obtain C_d.

The modified damping matrix is multiplied by v to obtain the force in physical coordinates. The transformation matrix arg2 is used to transform this physical force vector to modal coordinates, giving P_m.

The iteration continues until the maximum difference between displacements u from one iteration to the next is less than conv. If convergence is not reached in maxit iterations, a warning is issued but the unconverged results will be treated as converged and the solution will proceed.

This command can be executed any number of times, for example, to evaluate multiple wave angles.

The results are written to files, whose names are the project name with an extension. In addition to the basic *.out file, the following file will be created:

*.cor --> generalized coordinates

If they do not exist, the following main arrays are created in the database:

.hyd_rhs(nmode) -> RHS of generalized equations
.hyd_dynk(nmode,nmode) -> dynamic stiffness (LHS)
.hyd_gencor(nmode,nfreq,nbeta) -> generalized coordinates

where nmode is the number of modes, nfreq is the number of wave frequencies, and nbeta is the number of wave angles. Additional, temporary, arrays are also created and deleted. They all begin with either ".hyd_" or "$". To avoid a collision in array names, the user should not create any other array with these prefixes.

The results from this command will be inserted in .hyd_gencor(nmode,nfreq,beta). Hence, all wave angles can be dealt with one at a time.

See Also
hyd_analysis hyd_postresponse

hyd_analysis_response_P

Command to determine response to harmonic loading

Command Syntax

```
hyd_analysis_response_P  [-eq] [-residual] [symmetrize=on|off]
```

This command is very similar to hyd_analysis_response, which is used for wave-induced motion, whereas this command determines the response to predefined load pattern "P", contained in hyd_Peiwt(nmode). This command uses the added mass and hydrodynamic damping determined by the hyd_analysis command, which must precede this command. Only the radiation potentials (i.e., added masss and hydrodynamic damping) are considered. The excitation frequencies are specified by the hyd_wave command. Specify one wave angle (0 degrees, for example). The wave exciting forces are ignored. Although it is a bit inefficient to calculated wave exciting forces and then ignore them, the computational effort to obtain the wave exciting forces for one wave angle is relatively small.

For earthquake ground motion, specify option -eq. In a usual earthquake analysis, the effective load vector Peiwt would be defined as the negative of the transpose of the generalized modes times the combined mass (structure mass Ms + added mass Mf) times an influence vector r (i.e., -psi^T * (Ms + Mf) * r). r specifies which displacement degrees of freedom move with the ground motion. However, the approach used here is to form the generalized added mass (Mf* = psi^T * Mf * psi) directly, and hence Mf is not available. It can be shown that the effective load vector can be approximated as

```
-(I + Mf*) * psi^T * Ms * r
```

if the generalized structural mass is the identity matrix; i.e., the modes are orthonormal with respect to the structure mass matrix. Therefore, when -eq is specificed, Peiwt should specified as psi^T * Ms * r, which will be multiplied internally by -(I + Mf*) for each frequency.

For earthquake ground motion, there is an additional term to the effective load vector because the hydrodynamic damping depends on the total velocity (as compared to viscous structure damping, which is assumed to depend on the relative velocity). Again assuming that the generalized structure mass matrix is the identity matrix, the additional term for the effective load vector is

```
+ i/w Cf* * psi^T * Ms * r
```

in which Cf* is the generalized hydrodynamic damping matrix.

The above expressions for the effective load vector are based on the assumption that the influence vector r can be represented by the generalized modes. There is typically an error in this representation, which results from using a truncated subspace of generalized modes. The error is

```
e = r - psi * p_r = r - psi * (Ms*)^-1 psi^T * Ms * r
```

in which psi * p_r is the approximation of r by the generalized modes. The more modes one uses, the better this approximation should be.

If -residual is specified, the residual error in the solution of the equations of motion is calculated. The maximum norm of the error is reported.

If symmetrize=on, the added mass and hydrodynamic damping matrices are symmetrized by averaging the corresponding off-diagonal terms. If symmetrize=off, no averaging is done (default). Theoretically, these matrices should be symmetric. However, because of discretization errors, some corresponding terms may not be symmetric. The unsymmetry reduces as the mesh is refined.

Some basic commands to carry out the analysis can include:

```
  hyd_parameters
  hyd_coordaxs
  hyd_wave
  hyd_nodes
  hyd_coord_trans
  hyd_panel
  hyd_body_check
  hyd_rigid_modes
  hyd_rmass
  hyd_flex_modes
  hyd_genmodes
  hyd_analysis
  hyd_analysis_response_P
  hyd_postresponse_P
```

Note that this list does not include the creation of the modal forces in hyd_Peiwt(nmode).

Other commands can be used, as needed. In addition, the sequence of the commands as given above may be altered somewhat. However, the hyd_analysis command must precede this command. Before this command, the structural mass matrix must be defined as a database member and named hyd_mstr. For single or multiple rigid bodies, this matrix is created easily with the hyd_rmass command. If a viscous structural damping matrix (hyd_cstr) and a structural stiffness matrix (hyd_kstr) are not defined, they will be created and zeroed. (Constant structural hysteretic damping can be easily specified via the hyd_parameters command.) The dimensions of the structural matrices should be (nmode,nmode), where nmode is the number of modes. The hyd_flex_modes command is not needed to carry out the hydrodynamic analysis of single or multiple rigid bodies.

The results are written to files, whose names are the project name with an extension. In addition to the basic *.out file, the following file may be created:

```
 *.cor  --> generalized coordinates
```

See the commands hyd_analysis and hyd_analysis_response for more details. This command creates the following array in the database:

```
.hyd_gencor(nmode,nfreq)         -> generalized coordinates
```

where nmode is the number of modes and nfreq is the number of frequencies. Additional, temporary, arrays are also created and deleted. They all begin with either ".hyd_" or "$". To avoid a collision in array names, the user should not create any other array with these prefixes.

See Also
  hyd_analysis hyd_analysis_response hyd_postresponse_P

hyd_assign_mooring

Assign mooring stiffnesses to bodies

Command Syntax

```
hyd_assign_mooring
body=?  mooring=?  attach=?,?,?  [theta=?]  [T=?,?,?,?,?,?,?,?,?]
```

body is the body number
mooring is the stiffness number in .hyd_mooring_K
attach is the x,y,z body coordinates of the attachment point on the body
theta is the rotation in degrees about the z-axis (see below)
T is a 3x3 transformation matrix in the order T(1,1),T(2,1),T(3,1),T(1,2),etc.

Assigns ("attaches") a mooring stiffness to a body. The stiffness is assembled into the structural stiffness matrix hyd_kstr. If hyd_kstr does not exist or is not the proper size (nmode x nmode), it is created.

Prior to adding the mooring stiffness to the structural stiffness, it is transformed to body coordinates. If the z-axis of the mooring stiffness coordinate system is parallel to the z-axis of the body coordinates, then the transformation is conveniently specified by theta, which is the z-rotation from the mooring x-axis to the body x-axis. Otherwise, the general 3x3 orthogonal coordinate transformation matrix, T, that transforms a vector from the body-fixed coordinate system to the mooring coordinate system may be input in column order.

The mooring stiffness is used to determine the contribution of the mooring stiffness to the stiffness of the rigid body modes, as defined by the command hyd_rigid_modes. A mooring stiffness assigned, for example, to body 2 will have stiffness contributions to rigid body modes 7 - 12, the surge, sway, heave, roll, pitch, and yaw of body 2. The mooring stiffnesses must be assigned prior to any transformation of the modes to assumed modes, for example by the command hyd_genmodes. Also, any contribution to flexible modes is not included by this command. The user must define that contribution "manually" when specifying hyd_kstr.

Every mooring line must be assigned explicitly. In particular, even for a symmetric mooring arrangement, each mooring line must be specified.

See Also

hyd_mooring_stiffness phyd_mooring_stiffness

hyd_body_check

Perform some checks on the panel mesh

Command Syntax

```
hyd_body_check  [body=?]  [nodes=?,?]  [panels=?,?]  [body_sym=?]
```

This command performs some checking of the body panel mesh. The arguments are only required in the case of multiple bodies, as defined by the parameter nbodies in the hyd_parameters command. In this case, the checking will be for the body number specified by the parameter body. Hence, one hyd_body_check command should be issued for each body for which a mesh is specified explicitly. The parameters nodes and panels specify the ranges of nodes and panels corresponding to body. For example, body=1 nodes=1,100 panels=1,81 mean that body 1 is represented by nodes 1 to 100 and panels 1 to 81. If symmetry in the command hyd_parameters is 1 or 2, then the parameter body_sym must be specified here. Its value must be: 0, if the mesh for the particular body is for the entire body; 1, if the mesh for the particular body is for 1/2 the body; and 2 if the mesh for the particular body is for 1/4 the body. The default for body_sym is the system symmetry specified by hyd_parameters.

This command will check that the nodal coordinates are consistent with the symmetry specification. E.g., in the case of double symmetry, only the body in the (+x,+y) quadrant should be meshed. A warning will be issued for all nodes outside this quadrant.

The volume of the mesh is reported, as calculated by the x, y, and z projections.

The inertial coordinates of the center of buoyancy are calculated, based on the average of the three volume calculations.

hyd_close_files
    Close output files

hyd_convert_fea_mesh

Convert FEA mesh to a hydrodynamic panel mesh

Command Syntax

```
hyd_convert_fea_mesh
```

Import the fea mesh defined in HYDRAN-XR to a hydrodyanmic panel mesh. All FEA nodes are converted to "hydrodynamic" nodes.

A hydrodynamic panel is created for each "wet" interface, min5s and min3s element. If the "positive" side of the finite element is wet, then the node order is reversed when defining the panel to make it consistent with the clockwise specification of nodes for the panels.

The interface elements are converted first, followed by the min5s elements and then the min3s elements.

This command means that a mesh does not have to be essentially redefined by the hyd_nodes and hyd_panel commands as long as a compatible finite element mesh has been defined.

NOTE: the fea nodal coordinates must be specified in the inertial coordinate system to use this command.

See Also

hyd_nodes hyd_panel interface min3s min5s

hyd_coordaxs

Define the coordinate systems for the hydrodynamic analysis.

Command Syntax

```
  hyd_coordaxs [origin=?,?,?] [angles=?,?,?] [xb=?] [yb=?] [zb=?]   &
               [theta=?] [zcg=?] [body=?]
```

origin = (x,y,z) inertial coordinates of the origin of the input coordinate system
angles = Euler (Bryant) angles that define the orientation of the input axes (degrees)
xb = inertial x-coordinate of the origin of the body fixed coordinate system
yb = inertial y-coordinate of the origin of the body fixed coordinate system
zb = inertial z-coordinate of the origin of the body fixed coordinate system
theta = angle in degrees between the inertial x-axis and the body-fixed x-axis (see below)
zcg = body-fixed z-coordinate of the center of gravity
body = the number of the body (see below)

Although the default for all values is 0, this command is required.

There is one inertial (global) coordinate system, and for each body there is an input coordinate system and a body-fixed coordinate system. The number of bodies is specified by the nbodies parameter in the hyd_parameters command. There is always at least one body. For a given body, the input coordinate system is used to specify the nodal coordinates for that body. The body-fixed coordinate system is used principally to define the rigid body modes for the body.

The inertial coordinate system is located on the still-water plane, with the z-axis positive upward. The hydrodynamic calculations are carried out in the inertial coordinate system.

The input coordinate system for a body is for input convenience. If the input and inertial coordinates systems are not the same, then after the nodes have been defined the user must transform the coordinates to the inertial system by the hyd_coord_trans command. The input coordinate system is defined relative to the inertial system by a shift (specified by the origin parameter) and a rotation (specified by the angles parameter). The orientation of the input coordinate system is obtained by sequentially imposing the rotations about first the input x-axis, then the rotated y-axis after 1 rotation, and finally the z-axis after 2 rotations.

The origin of the body-fixed coordinate system is specified by the parameters xb, yb, and zb. The origin of the body axes need not be at the center of gravity of the body. However, it must be on the same vertical line as the CG of the body. The z-axis of the body fixed coordinate system is positive upward. The angle theta is the angle between the inertial x-axis and the body x-axis, measured from the inertial x-axis with counterclockwise positive. The inertial z-axis and the body z-axis are parallel.

In the case of multiple bodies, as defined by nbodies in the command hyd_parameters, one hyd_coordaxs command must be input for each body, and the body number ranges from 1 to nbodies. In this case, the data specified here define the location and orientation of each body's input coordinates and body-fixed coordinates.

In the case of symmetry, it is possible that a body is a complete reflection of another body that has been input. There are no nodes or panels that are input explicitly for such a body. This command is still required, however, to specify zcg. Although the values are not used for calculations, it is recommended that the origin of the body-fixed coordinate system be given as well so that the correct values will be printed in the project summary.

The data are stored as:

```
  .hyd_coordaxs(11,nbodies) -> origin, angles, xb, yb, zb, theta, zcg
```

See Also

```
  hyd_coord_trans  hyd_nodes  hyd_rmass
```

hyd_coord_trans

Transform the nodal coordinates to the inertial coordinate system.

Command Syntax

```
hyd_coord_trans  [body=?]  [nodes=?,?]
```

This command is required if the input coordinate system and the inertial coordinate system are not the same, as defined by the command hyd_coordaxs. For multiple bodies, as specified by the nbodies parameter in the hyd_parameters command, the parameters body and nodes are required. body is the body number and nodes define the range of nodes that correspond to that body. E.g., body=1 nodes=1,100 would mean that the coordinates for nodes 1 to 100 would be transformed using the coordinate transformation data for body 1 that was specified in the hyd_coordaxs command. For multiple bodies, the command must be issued as many times as necessary to transform the nodal coordinates to the inertial coordinate system.

If this command is required, it must be issued after all the nodal coordinates have been defined (hyd_nodes) and before the panels are defined (hyd_panel).

See Also

hyd_coordaxs hyd_nodes hyd_panel

hyd_export_graphics

Export hydrodynamic panel mesh to graphics program input file

Command Syntax

```
  hyd_export_graphics  -target  [-modes T=?]                      &
                       [-displ  freq=?  angle=?  steps=?]         &
                       [-freesurface]  [-wetonly]  [file=filename]
```

Export the hydrodynamic panel mesh to a graphic program's input text file.

The graphics program is specified by the argument -target. Only the program Gmsh (http://www.geuz.org/gmsh/) is supported at this time. That is, the available option is -Gmsh.

The default is to plot the undeformed mesh.

If -modes is specified, the modes in .hyd_umx, .hyd_umy, and .hyd_umz (see hyd_rigid_modes and hyd_flex_modes command) are exported as displacements. T= is the name of a real vector of frequencies or periods, which is used for identification in the output file. The size of the vector is the number of modes. If a vector is not specified, one is created with the values 1,2, ...

If -displ is specified, then a time sequence of displacements will be determined for a given wave frequency and wave angle. freq is the integer number of the wave frequency/period, angle is the integer number of the wave angle, and steps is the number of steps in the wave period for which the real displacements will be determined.

If -freesurface is specified, then the free surface panels are exported as well. If modes is selected, then the free surface is flat. If deformed is selected, then the free surface elevation is also plotted. The displacement of the free surface is determined by the hyd_surf_elevation command.

If -wetonly is specified, only "wet" panels are exported (panels with a W or V code); see command hyd_panel for information.

If filename is specified, the results will be written to the file filename; othwerwise they will be written to the file project_name.msh (Gmsh).

hyd_export_graphics_th

Export time history of motion response to graphics program input file

Command Syntax

```
hyd_export_graphics_th  [-target]  time  xdisp  ydisp  zdisp      &
                        [surfzdisp]  [t1=?]  [t2=?]  [step=?]     &
                        [file=filename]
```

The graphics program is specified by the argument -target. Only Gmsh (http://www.geuz.org/gmsh/) is supported at this time. That is, the available option is -Gmsh, and it is the default.

The time history response must have been previously calculated, e.g., via a Fourier transform approach (see the command fft).

time is a vector of N time steps
xdisp is the (N,#nodes) matrix of x-displ. for the structure panels
ydisp is the (N,#nodes) matrix of y-displ. for the structure panels
zdisp is the (N,#nodes) matrix of z-displ. for the structure panels
surfzdisp is the (N,#surfnodes) matrix of z-displ. for the surface panels

If surfdisp is not given, then the surface displacements are not plotted.

If t1 is given, export begins at time(t1) (default = 1).
If t2 is given, export will stop at time(t2) (default = N).
If step is given, every step time steps will be exported (default = 1).

If filename is specified, the results will be written to the file filename; othwerwise they will be written to the file project_name.msh (Gmsh).

hyd_flex_modes

Input the flexible structural modes for the hydroelastic analysis. There are two Command Syntax options.

```
          ---------- OPTION 1 ----------
Command Syntax
  hyd_flex_modes arg

          ---------- OPTION 2 ----------
hyd_flex_modes  [-noread]
  mode_j  x_z_sym  y_z_sym
  i  ux(i,j)  uy(i,j)  uz(i,j)  thx(i,j)  thy(i,j)  thz(i,j)
```

Option 1

arg is the array in the database containing the structural mode shapes from the eigenvalue analysis; e.g., .phi (see command eigval). These correspond to the structural degrees-of-freedom, and the dimension is neq x #modes. For this option to work, the node numbers of the structural mesh and the node numbers of the panel mesh must be compatible. This can be ensured via the command hyd_convert_fea_mesh. Each column of arg is a mode shape. The first nmoder columns are skipped, and the next nmodef columns are processed. Hence, #modes must be greater than or equal to nmoder + nmodef. If nmoder > 0, then it is assumed that the arrays .hyd_umx, .hyd_umy, etc. (see hyd_rigid_modes for all the arrays expected) have been created and the first nmoder columns have been defined already (e.g., via the hyd_rigid_modes command or "manually"). This command will define the values for the nmodef modes. If symmetry is not 0, the values for the nmodef modes must be defined manually.

Option 2

```
mode_j  = mode number
x_z_sym = port-starboard symmetry code for mode_j
           0 -> port-starboard symmetric
           1 -> port-starboard anti-symmetric
y_z_sym = fore-aft symmetry code for mode_j
           0 -> fore-aft symmetric
           1 -> fore-aft anti-symmetric
ux,uy,uz(i,j)   = translation of node i in the j-th mode
thx,thy,thz(i,j)= rotation of node i in the j-th mode
```

Ranges of the indices are: i = 1, nnode; j = nmoder+1, nmode.

End input for each mode with a blank line. That is, the modes are separated by a blank line.

The use of -noread is expected to be uncommon. If the flag -noread is present, the data are not read and only the one command line should be specified. In this case it is assumed that the seven arrays .hyd_modesym, .hyd_umx,...,.hyd_thz (see below) have been defined elsewhere and the data are in them for further processing. .hyd_kf, .hyd_un, and .hyd_uz will be created.

The data are stored as:

```
.hyd_umx(nnode,nmode) -> translational x displacements
.hyd_umy(nnode,nmode) -> translational y displacements
.hyd_umz(nnode,nmode) -> translational z displacements
```

```
  .hyd_thx(nnode,nmode) -> rotational x displacements
  .hyd_thy(nnode,nmode) -> rotational y displacements
  .hyd_thz(nnode,nmode) -> rotational z displacements
```

The symmetry code for mode j is stored in .hyd_modesym(j). These codes are only used if the structural symmetry parameter on the hyd_parameters command is 1 or 2, indicating single (x-z) or double (x-z and y-z) structural symmetry, respectively. For single symmetry, the modal symmetry code is:

1 -> symmetric
2 -> antisymmetric

For double symmetry, the modal symmetry code is:

1 -> symmetric/antisymmetric
2 -> antisymmetric/symmetric
3 -> symmetric/symmetric
4 -> antisymmetric/antisymmetric

where, e.g., symmetric/antisymmetric means symmetric with respect to the x-z plane and antisymmetric with respect to the y-z plane.

Note: The modal displacements are given in the inertial coordinate system.

This command defines .hyd_un and .hyd_uz for the flexible modes and it forms an estimate - for the flexible modes - of the hydrostatic stiffness .hyd_kf based on the fluid term only. If a better hydrostatic stiffness matrix is available, it should replace the one created by this command (after the command finishes).

See Also
  hyd_coordaxs hyd_coord_trans hyd_nodes hyd_rigid_modes

hyd_genmodes

Transform matrices to generalized coordinates.

Command Syntax

```
hyd_genmodes [1=?/?  2=?/?]
```

This command transforms the input modes to assumed modes, as explained below. There must be the same number of assumed modes as input modes (nmoder + nmodef from hyd_parameters command).

This command is meant primarily for a system of multiple rigid bodies, in which the initial modes are specified to be the traditional surge, sway, heave, etc. of each body. This command allows the transformation to symmetric and antisymmetric modes. The array hyd_psi(nmode,nmode) must be defined prior to this command (e.g., by the input command). Array hyd_psi is defined such that

```
{d} = [psi] {u}
```

in which [psi] is hyd_psi, {d} is the displacement vector, and {u} is the vector of generalized displacements. For the cases of single and double symmetry of the system (as defined by the symmetry parameter on the hyd_parameters command), the symmetry/antisymmetry of the modes are specified by the parameters i=?/?, where i is the number of the mode (1 to nmode) and ?/? is of the form S or A for single structural symmetry and S/S, S/A, A/S, and A/A for double structural symmetry. S and A refer to symmetric and antisymmetric, respectively. This input is used to define the vector .hyd_modesym. See the help on commands hyd_rigid_modes and hyd_flex_modes for more discussion on .hyd_modesym.

The transformation is carried out by replacing the system matrices as indicated below:

```
hyd_mstr   <- [psi]^T * [hyd_mstr] * [psi]
hyd_kstr   <- [psi]^T * [hyd_kstr] * [psi]
hyd_cstr   <- [psi]^T * [hyd_cstr] * [psi]
.hyd_kf    <- [psi]^T * [.hyd_kf]  * [psi]

.hyd_umx   <- [.hyd_umx] * [psi]
.hyd_umy   <- [.hyd_umy] * [psi]
.hyd_umz   <- [.hyd_umz] * [psi]
.hyd_thx   <- [.hyd_thx] * [psi]
.hyd_thy   <- [.hyd_thy] * [psi]
.hyd_thz   <- [.hyd_thz] * [psi]
.hyd_un    <- [psi]^T * [.hyd_un]
.hyd_uz    <- [psi]^T * [.hyd_uz]
```

Note that the response given in the file *.cor will be the generalized coordinates. Usually, one will also want to define a "modal" matrix such that the response corresponding directly to the original modes will be determined by the command hyd_tf. If no other response components are desired, the modal matrix will be the same as hyd_psi. In this case, it is conveniently defined by the cp or mv commands.

See Also

hyd_flex_modes hyd_parameters hyd_rigid_modes hyd_tf

hyd_irregular

Calculate short-term extreme response in irregular seas

Command Syntax

```
  hyd_irregular arg1  arg2  [ext=extension]  [file=filename]
```

This command determines the short-term extreme response based on the transfer functions that have been determined with the hyd_tf command and the wave spectra specified with the hyd_wave_spectra command. arg1 is the name of the array with the transfer functions. The extreme responses are put in the array whose name is specified by arg2. The command creates this array with dimensions (nspectra,nbeta,ncomp), where nspectra and nbeta are the number of wave spectra and wave angles, respectively.

Note: The extreme values are defined as 3.72 * square root of the variance of the response, i.e., 3.72 * square root of the area of the RAO^2 * wave spectrum.

The extreme responses are written to a file. If extension is specified, the file name is project_name.extension. Otherwise, if filename is specified, the file will have the name specified by filename.

See Also

  hyd_tf hyd_wave_spectra

hyd_modal_pressure

Print modal pressures

Command Syntax

```
hyd_modal_pressure  [digits=?]
```

For each panel, prints the incoming + diffraction pressure and the pressure in each mode. The results are in file project_name.prs2. For this command to function, the potentials must have been saved in the hyd_analysis command.

digits is the number of significant digits to print (default is 5).

Note: The pressures determined by the hyd_analysis_response command are the total hydrodynamic pressures. This command prints the pressures in each mode, for a unit displacement of that mode. Therefore, for large problems, the file created may be quite large.

See Also

hyd_analysis hyd_analysis_postresponse

hyd_mooring_line

Mooring line stiffness calculated from elastic catenary cable

Command Syntax

```
  hyd_mooring_line  m=?  n=?  [maxiter=?]  [tol=?]
  m=seg_prop  e=emodulus  a=area  w=wx,wy,wz                  (m records)
  n=nel  anchor=x1,z1  end=x2,z2  [#segs=#segs]  [tension=tenX,tenY,tenZ]
    seg=seg  mat=seg_prop  L=length                         (#segs records)
```

m is the number of different mooring line properties
n is the number of mooring stiffnesses
maxiter is the max. # of iterations on the tension (default=30)
tol is the relative tolerance on the end point position (default=1.e-5)

For each set of mooring line segment properties:
seg_prop is the segment property number
emodulus is the modulus of elasticity
area is the effective cross sectional area
wx,wy,wz are the weight/unit length components in global coordinates

For each element:
nel is the stiffness number
x1,z1 are the x and z coordinates of the anchor point
x2,z2 are the x and z coordinates of the top point
#segs is the number of different segments (default=1)
tension is the initial estimate of the tension

seg is the segment number
seg_prop is the segment property number
length is the unstretched segment length

This command will calculate the 3x3 stiffness at the end (attachment) point for a mooring line. The stiffnesses are put in the array .hyd_mooring_K(3,3,n). The geometry of the mooring lines are defined in the plane of the line, and therefore only two coordinates are used to specify the anchor point and end point. The lines actual orientation in the global inertial coordinate system and their attachment to a particular body are specified by the command hyd_assign_mooring. The same stiffness can be assigned multiple times and to multiple bodies. Therefore, it is only necessary to define unique mooring lines once.

The element is based on small strain elastic catenary theory. A shooting method is used to solve the two-point boundary value problem. Specifically, iteration on the tension at the anchor point is carried out until the distance between the calculated position of the end and the specified position of the end, divided by the element length, is less than or equal to the tolerance (tol). For information on the formulation, see H.R. Riggs and T. Leraand, "Efficient Static Analysis and Design of Flexible Risers," J. Off. Mech. Arctic Engrg., Vol. 113, pp. 235-240, 1991, and H.R. Riggs and T. Leraand, "A Robust Element for Static Analysis of Marine Cables," Proc. Third International Offshore and Polar Engineering Conference, Singapore, Vol. 2, pp. 357-363, 1993. Note: the element described in those papers includes fluid drag; this element does not.

See Also

hyd_assign_mooring hyd_mooring_stiffness phyd_mooring_line

phyd_mooring_line

command Syntax

phyd_mooring_line

Print mooring line data as specified by hyd_mooring_line

See Also

hyd_mooring_line

hyd_mooring_stiffness

Specify mooring stiffnesses

Command Syntax

```
hyd_mooring_stiffness #=nstiff
n=?
(Input 3x3 stiffness matrix - 1 row/input record)
```

Reads nstiff 3x3 mooring stiffness matrices. n is the mooring stiffness number, which must be in the range 1 to nstiff. Multiple stiffness matrices may not be separated by blank or comment lines.

Each 3x3 matrix relates the displacements (u1,u2,u3) and forces of the mooring line at the point where it will be connected to a body. The forces and displacements are defined in a "mooring line coordinate system." A typical situation is: u1 is the horizontal displacement in the plane of the mooring line, u2 is the horizontal displacement normal to u1, and u3 is the vertical displacement. Only unique mooring stiffnesses must be defined.

End input with a blank line.

The command stores the stiffnesses in the array .hyd_mooring_K(3,3,nstiff).

See Also

hyd_assign_mooring hyd_mooring_line phyd_mooring_stiffness

phyd_mooring_stiffness

Print mooring stiffnesses

Command Syntax

```
phyd_mooring_stiffnesses
```

Print the mooring stiffnesses.

See Also

hyd_assign_mooring hyd_mooring_line phyd_mooring_stiffness

hyd_nodes

Command Syntax

```
hyd_nodes  #=?
n=node_no  x=x-coor  y=y-coor  z=z-coor  [lgen=lgen]
```

Reads and generates nodal coordinates. The value specified by # is used to define storage requirements, and it must be greater than or equal to the maximum node number. If this value is missing or 0, it is assumed that existing nodes are being changed or added to, and the previous value applies. lgen is the node number increment for linear generation. Nodes are generated equally spaced along a straight line if two adjacent records do not have sequential node numbers and if lgen on the second line is not zero or blank. Nodes need not be input in sequence.

End input with a blank line.

This command must precede the hyd_panel command.

The coordinates are stored in array .hyd_xyz(3,#), and the maximum possible node number (specified by #) is stored in .hyd_#nodes_tot.

Active nodes are those that are defined explicitly either by this command or another command that creates nodes. The node number of the maximum defined node is stored in .hyd_#nodes. The character vector .hyd_node_active has an "A" for active nodes. Only active nodes can be used.

The nodes command need not be executed as long as the coordinates, which could be generated by another program, are put in the array .hyd_xyz, .hyd_#nodes and .hyd_#nodes_tot are set, and .hyd_node_active is created.

See Also

hyd_coordaxs hyd_coord_trans hyd_node_tolerance phyd_nodes

phyd_nodes

Command Syntax

```
phyd_nodes  [nodes=?,?]  [-screen]
```

Print nodal coordinates. A range of node numbers can be specified by nodes=. The first value is the first node number in the range, and the second value is the last number in the range. The default is to print the coordinates for all nodes defined. The default is to print to output file only; if -screen is present, then output is to the screen as well.

See Also

hyd_nodes

hyd_node_gen

Command Syntax

```
hyd_node_gen
linear=node1,node2  [inc=?]  [w=?]
quad=node1,node2,node3,node4  [inc=inc1,inc2]  [w=w1,w2]
```

Generates nodal coordinates. The hyd_nodes command must preceed this command.

Linear generation is specified by the identifier linear. Nodes are generated from node1 to node2 with a node increment of inc (default=1). The spacing of nodes is equidistant unless the positive weight w (default=1) is specified, in which case the spacing between node n and n+1 is equal to w times the spacing between n-1 and n.

The identifier quad specifies node generation within the "quadrilateral" defined by the four nodes (see sketch below). Linear generation is done between node1 and node2, and between node3 and node4, based on inc1 and w1. Linear generation is also done between node1 and node3, and between node2 and node4, based on inc2 and w2. Then, linear generation is done between the nodes generated from 1-3 and 2-4, based on inc1 and w1. The number of interior lines generated is the same as the number of nodes generated from 1-3. If the number of nodes generated from 1-3 is larger than the number generated from 2-4, the "extra" lines will not be generated, as this would result in a redefinition of nodal coordinates along 3-4. If node3 and node4 are identical, the generated nodes are within a triangular domain. The nodes need not be coplanar.

End input with a blank line.

```
        node3                                        node4
          X----------O----------O----------O----------X
          |                                           |
          |                                           |
          |                                           |
          |                                           |
    node1 |       node1+inc2                          |
    +inc2 O          O +inc1    O           O         O
   ^      |                                           |
   |      |                                           |
 generate |                                           |
     w/   |                                           |
  inc2    |                                           |
          |                                           |
          |          generate w/ inc1 ->              |
          X----------O----------O----------O----------X
        node1    node1+inc1                         node2
```

X = specified node
O = generated node

See also

hyd_nodes

hyd_node_tolerance

Command Syntax

```
hyd_node_tolerance  [z_tol=?]  [x_tol=?]  [y_tol=?]
```

Checks for and corrects slight errors in nodal coordinates. This command checks for nodes that should be on the still water plane or planes of symmetry, but numerical precision-related errors cause them to be slightly off these planes. This can happen when coordinates are generated by a mesh generation or CAD program, or when input coordinates are transformed. If a nodal z-coordinate is within z_tol (plus or minus) of the still water plane, it is set to 0. For single or double symmetry, if a nodal y-coordinate is within y_tol of the x-z plane, it is set to 0. For double symmetry, if a nodal x-coordinate is within x_tol of the y-z plane, it is set to 0. The default for each tolerance is 0.001. Tolerances should be positive; if a negative value is specified, the default of .001 is used. (Note: the symmetry type must have been specified by the hyd_parameters command for x_tol and y_tol to be used.

See Also

hyd_nodes

hyd_panel

Constant pressure flat fluid panel

Command Syntax

```
hyd_panel n=?
n=nel  nodes=node1,node2,node3,node4  [gen=gen  inc=inc1,inc2] &
    [gen_2d=gen_2d  inc_2d=inc1_2d,inc2_2d  inc_el=inc_el]
```

n is the maximum panel (element) number

nel is the element number
node1 thru node4 are node numbers
inc1, inc2 are node increments in a "linear sequence"
gen is the number of elements to generate in a sequence
inc1_2d, inc2_2d are node increments between sequences
gen_2d is the number of linear sequences to generate
inc_el is the element increment between sequences

Nodes 1 to 4 are the corner nodes for quadrilateral elements. For triangular elements, node 4 should either be 0 or equal to node 3. The nodes are specified clockwise, looking from the fluid (see sketch below).

A "linear sequence" of elements can be generated by specifying inc1, inc2 and gen. In a linear sequence, nodes 1 and 2 are incremented by inc1; and nodes 3 and 4 are incremented by inc2. gen is the number of elements to generate, so a sequence will have gen+1 elements. To generate a 2D patch of elements, multiple sequences can be specified; inc1_2d and inc2_2d are used to increment the node numbers from one sequence to the next. gen_2d is the number of additional sequences. The element numbers in two successive sequences differ by inc_el (default = numgen+1).

End input with a blank line.

On input, created arrays are:

```
.hyd_panel_el(4,n)   -> node1 - node4
.hyd_#panels(1)      -> total # of defined panels
.hyd_#wetpanels(1)   -> total # of wet panels
.hyd_panela(nwet)    -> panel areas
.hyd_paneln(3,nwet)  -> components of panel normals
.hyd_paneltype(nwet) -> = 0 for quad; = 1 for triangle
.hyd_panel_code(n)   -> wet/dry panel codes
.hyd_panel_#map(n)   -> map panel numbers to internal numbers
.hyd_xr(nwet,4)      -> x coord. of the panel corners
.hyd_yr(nwet,4)      -> y coord. of the panel corners
.hyd_zr(nwet,4)      -> z coord. of the panel corners
.hyd_xyzc(nwet,3)    -> x,y,z coords. of the panel centers
```

where nwet are the number of wet panels (see below).

Panels that are out of the fluid are ignored for hydrodynamic calculations. The vector .hyd_panels_code contains a character code: blank -> panel not defined; W -> wet panel; D -> dry panel; V -> wet panel that is partially dry; and E -> dry panel that is partially wet. A panel with all nodes below the still water plane is W. A panel with all nodes above the still water plane is D. A panel with some nodes above the still water plane and some below is either V or E. It is V if the panel center is below the still water line. It is E if the center is above the still water line. Only wet panels (W and V) are included in the hydrodynamic calculations. These panels are numbered internally. The mapping from the "external" numbering to the internal numbering is in the vector .hyd_panels_#map.

If the 4 nodes of a quadrilateral panel are not coplanar, it is replaced by an "equivalent" flat panel; it is the data for this flat panel that are stored in the above arrays.

The hyd_nodes command must precede this command. Also, because the panel normals and the coordinates of the panel corners and the centroids are in the inertial coordinate system, the hyd_coords_trans command, if it is required, must precede this command.

```
   node2                                       node3
     X-------------------------------------------X
     |                                           |
     |                                           |
     |                                           |
     |                                           |
     |                                           |
     |                                           |
     |                                           |
     |                                           |
     |                                           |
     |                                           |
     |                                           |
     X-------------------------------------------X
   node1                                       node4
```

See Also

phyd_nodes phyd_panel

phyd_panel

Command Syntax

```
phyd_panel  [-wet_only]
```

Print hyd_panel element data.

If -wet_only is specified, only "wet" panels will be printed.

See Also

hyd_panel

hyd_panel_rmap

Create reverse mapping of panel numbers

Command Syntax

```
hyd_panel_rmap
```

Create the integer vector .hyd_panel_rmap(#wetpanels) that contains the input panel number for each wet panel. The wet panel number is assigned internally, and ranges from 1 to #wetpanels.

This command is called automatically by hyd_analysis if the potentials or pressures are requested. The mapping is needed to print the potentials and pressures.

See Also

hyd_analysis hyd_panel

hyd_parameters

Input control parameters for the hydrodynamic analysis.

Command Syntax

```
hyd_parameters h=?  [symmetry=?]  [nmoder=?]  [nmodef=?]  [nbodies=?]  &
               [gauge=?]  [grav=?]  [rho=?]  [kh=?]  [hdamp=?]
```

```
h is the water depth
symmetry specifies the symmetry of the panel mesh (see note below)
         = 0 -> no symmetry (default)
         = 1 -> port-stbd. (x-z) symmetry
         = 2 -> port-stbd. and fore-aft (x-z and y-z) symmetry
nmoder is the number of "user-defined" modes (see below)
nmodef is the  number of flexible modes
nbodies is the  number of multiple bodies
gauge is used for the panel characteristic length (default = 2.5)
grav is the gravitational acceleration (default = 9.80665)
rho is the mass density of the fluid (default = 1025)
kh is the limiting depth beyond which "deep" water is assumed (k is the
wave number; default kh = 10)
hdamp is the structural hysteretic damping (.02 means 2% damping)
```

Dimensional quantities, such as rho and grav, should be specified using consistent units throughout.

The number of user-defined modes are defined by nmoder. The default value for nmoder is 6. Typically, these modes are the traditional rigid body modes of surge, sway, heave, etc, which are defined with the hyd_rigid_modes command. However, any user-defined modes can be specified, including "flexible" modes. To define other than the traditional rigid body modes, the user must supply the matrices required to define these modes (e.g., mass matrix, stiffness matrix, etc.). See the help on the hyd_rigid_modes command for a description of the matrices that must be supplied.

nmodef is the number of flexible modes that are to be input with the command hyd_flex_modes.

The parameter nbodies may be specified if the analysis of multiple bodies (connected or unconnected) is to be carried out. In this case, nbodies is the total number of bodies. Such an analysis can be carried out without specifying nbodies, but defining the rigid body modes is simplified if this option is used. If nbodies is given, then nmoder is set to 6*nbodies.

Important note regarding symmetry:

The parameter symmetry specifies the structural symmetry with respect to the inertial coordinate system. Symmetry is used both to reduce user input and to reduce the computations of the added mass, hydrodynamic damping, and exciting forces. However, these quantities are determined for the entire structure, and the equations of motion are for the entire structure. Hence, the user must specify the structural matrices, such as mass and damping, for the entire structure. For a system of multiple bodies, the system is considered to be the "structure," and it is the structure that must be symmetric to exploit single or double symmetry. For additional discussion of symmetry as used in HYDRAN-XR, see the Getting Started section of the manual.

The characteristic length of a panel is defined as the product of gauge and the square root of the panel area, nondimensionalized with respect to the deep water wave number. Different integration techniques are used to compute the surface integrals involving the Green function and its derivative, depending on whether the distance between the field point and the source point is less than or greater than the characteristic length of the panel. In general, 2 < GAUGE < 4. Computational experiments by C.J. Garrison have indicated that gauge = 2.5 is an appropriate choice for the approximations to the surface integrations to be valid (see reference 10).

The data are stored as:

```
  .hyd_depth        -> h
  .hyd_symmetry     -> symmetry
  .hyd_nmoder       -> nmoder
  .hyd_nmodef       -> nmodef
  .hyd_nbodies      -> nbodies
  .hyd_deep_water   -> kh
  .hyd_gauge        -> gauge
  .hyd_grav         -> grav
  .hyd_hysdamp      -> hdamp
  .hyd_fluid_rho    -> rho
```

See Also

hyd_analysis hyd_flex_modes hyd_rigid_modes hyd_rmass

hyd_postresponse

Command to determine responses

Command Syntax

```
hyd_postresponse  [-disp] [-potential] [-pressure]
```

This command determines response quantities based on the solution of the equations of motion as determined by the hyd_analysis_response command, which must precede this command. The existing generalized coordinates from that command will be used.

If -disp is specified, the nodal displacements are determined based on the generalized displacements.

If -pressure is specified, the pressures are calculated based on the generalized displacements. The potentials must have been saved from the hyd_analysis command for this option to be possible.

If -potential is specified, the potentials for the entire structure are written. This means that for single and double symmetry, they are written for the "actual" panels and also for the "reflected panels." The file indicates how the reflected panels are identified. The potentials must have been saved from the hyd_analysis command for this option to be possible.

Results are written to files with names of the form *.ext, where * represents the project name. Output is:

```
-potential  -> velocity potentials at panel centers (*.pot)
-pressure   -> pressures at panel centers (*.prs and *.hpr)
-disp       -> nodal displacements (*.dis and *.res - a binary file)
```

If -pressure is specified, the hydrodynamic pressures are written to file *.prs and the change in hydrostatic pressures are written to file *.hpr. The change in hydrostatic pressure is defined as -rho * grav * u_z, where u_z is the displacement in the inertial z direction.

The pressures for the entire structure are written. This means that for single and double symmetry, they are written for the "actual" panels and also for the "reflected panels." The files indicate how the reflected panels are identified.

If -disp is specified, the nodal displacements are written to the formatted file *.dis and to the unformatted file *.res. The file *.res is written by the Fortran statement

```
write(f_res)omega, angle, disp
```

Omega and angle are the wave frequency (rad/sec) and wave angle (degrees). The displacements are in the vector disp(ndof), with ndof = 6*#nodes. The outer loop is on the wave frequency.

See Also

hyd_analysis hyd_analysis_response

hyd_postresponse_P

Command to determine responses

Command Syntax

```
hyd_postresponse_P  [-disp] [-potential] [-pressure]
```

This command determines response quantities based on the solution of the equations of motion as determined by the hyd_analysis_response_P command, which must precede this command. The existing generalized coordinates from that command will be used.

If -disp is specified, the nodal displacements are determined based on the generalized displacements.

If -pressure is specified, the pressures are calculated based on the generalized displacements. The potentials must have been saved from the hyd_analysis command for this option to be possible.

If -potential is specified, the potentials for the entire structure are written. This means that for single and double symmetry, they are written for the "actual" panels and also for the "reflected panels." The file indicates how the reflected panels are identified. The potentials must have been saved from the hyd_analysis command for this option to be possible.

Results are written to files with names of the form *.ext, where * represents the project name. Output is:

```
-potential  -> velocity potentials at panel centers (*.pot)
-pressure   -> pressures at panel centers (*.prs and *.hpr)
-disp       -> nodal displacements (*.dis and *.res - a binary file)
```

If -pressure is specified, the hydrodynamic pressures are written to file *.prs and the change in hydrostatic pressures are written to file *.hpr. The change in hydrostatic pressure is defined as -rho * grav * u_z, where u_z is the displacement in the inertial z direction.

The pressures for the entire structure are written. This means that for single and double symmetry, they are written for the "actual" panels and also for the "reflected panels." The files indicate how the reflected panels are identified.

If -disp is specified, the nodal displacements are written to the formatted file *.dis and to the unformatted file *.res. The file *.res is written by the Fortran statement

```
write(f_res)omega, angle, disp
```

Omega and angle are the wave frequency (rad/sec) and wave angle (degrees). The displacements are in the vector disp(ndof), with ndof = 6*#nodes. The outer loop is on the wave frequency.

See Also

hyd_analysis hyd_analysis_response

hyd_rigid_modes

Generate the traditional rigid body modes

Command Syntax

```
hyd_rigid_modes  [body=?]  [nodes=?,?]  [panels=?,?]  [body_sym=?]    &
                 [reflected=?,?,?]
```

This command generates surge, sway, heave, roll, pitch, and yaw rigid body modes relative to the body-fixed coordinate system. For a single body, no parameters are required. If this command is used, it must be given before the hyd_flex_modes command.

For multiple bodies, as defined by the parameter nbodies in the hyd_parameters command, it will generate the rigid body modes for the body number specified by the parameter body. Hence, one hyd_rigid_modes command must be issued for each body for which a mesh is specified explicitly. The first command must be for body 1. The parameters nodes and panels specify the ranges of nodes and panels corresponding to body. For example, body=1 nodes=1,100 panels=1,81 mean that body 1 is represented by nodes 1 to 100 and panels 1 to 81. If symmetry in the command hyd_parameters is 1 or 2, then the parameter body_sym must be specified here. It will be: 0, if the mesh for the particular body is for the entire body; 1, if the mesh for the particular body is for 1/2 the body; and 2 if the mesh for the particular body is for 1/4 the body. This specification is required so that the hydrostatic stiffness will be calculated correctly. Furthermore, if another body is represented by a reflection of this body, then that body number is specified by the parameter reflected. For single symmetry, at most one other body can be given. For double symmetry, either 0, 1, or 3 bodies can be specified.

Modes 1 - 6 correspond to body 1. For multiple bodies, modes 7 - 12 correspond to body 2, etc.

The following arrays are created, where nmode = nmoder + nmodef. This command populates the arrays with the appropriate data for the first nmoder modes.

```
.hyd_modesym(nmode)      -> symmetry codes for each mode
.hyd_kf(nmode,nmode)     -> hydrostatic stiffness matrix
.hyd_umx(nnode,nmode)    -> nodal x-displs.
.hyd_umy(nnode,nmode)    -> nodal y-displs.
.hyd_umz(nnode,nmode)    -> nodal z-displs.
.hyd_thx(nnode,nmode)    -> nodal x-rotations
.hyd_thy(nnode,nmode)    -> nodal y-rotations
.hyd_thz(nnode,nmode)    -> nodal z-rotations
.hyd_un(nmode,npanel)    -> panel normal displacements
.hyd_uz(nmode,npanel)    -> panel z-displacements
```

The symmetry code for mode j is stored in .hyd_modesym(j). These codes are only used if the structural symmetry parameter on the hyd_parameters command is 1 or 2, indicating single (x-z) or double (x-z and y-z) structural symmetry, respectively. For single symmetry, the modal symmetry code is:

1 -> symmetric
2 -> antisymmetric

For double symmetry, the modal symmetry code is:

```
  1 -> symmetric/antisymmetric
  2 -> antisymmetric/symmetric
  3 -> symmetric/symmetric
  4 -> antisymmetric/antisymmetric
```

where, e.g., symmetric/antisymmetric means symmetric with respect to the x-z plane and antisymmetric with respect to the y-z plane.

If user-defined modes are used but this command is not, then these arrays should be created with the appropriate data.

See Also

`hyd_flex_modes` `hyd_rmass`

hyd_rmass

Input the structural mass matrix.

Command Syntax

```
hyd_rmass  [body=?]
```

The mass matrix is specified "row-wise" immediately following the command, with 1 record per row. Note that only the mass associated with the user-defined modes is input with this command. This command is usually used to input a rigid body mass matrix. The mass matrix should be specified with respect to the same coordinates used to define the modes. If hyd_rigid_modes is used to define the rigid modes, then the mass matrix in the body-fixed coordinate system should be given here.

For a single body, the parameter body is not required. The command expects the (nmoder,nmoder) matrix to be input.

For multiple bodies, as specified by the parameter nbodies in the command hyd_parameters, the 6x6 mass matrix for body is expected. Hence, one hyd_rmass command must be given for each body. The first command must be for body 1.

The mass matrix is stored in the array hyd_mstr(nmode,nmode). If this matrix does not exist, it will be created. This command is not required. If it is not used, the structural mass matrix (hyd_mstr) must be defined in some other way.

See Also

hyd_coordaxs hyd_rigid_modes

hyd_surf_elevation

Command to determine the "free" surface elevation (i.e., the potentials) after a hydrodynamic analysis.

Command Syntax

```
  hyd_surf_elevation [incoming=?]  [diffraction=?]  [radiation=?]    &
                     [dif=difname]  [rad=radname]

    incoming    = 1 -> include the incoming potential (default)
                = 0 -> do not include the incoming potential
    diffraction = 1 -> include the diffraction potential (default)
                = 0 -> do not include the diffraction potential
    radiation   = 1 -> include the radiation potential (default)
                = 0 -> do not include the radiation potential
```

If dif= is specified, the diffraction source strengths are read from file difname; otherwise they are read from file project_name.dif.

If rad= is specified, the radiation source strengths are read from file radname; otherwise they are read from file project_name.rad.

The hyd_analysis and hyd_surf_nodes commands must precede this command. As of now, this command has only been implemented for the case of no symmetry (hyd_parameters command). Also, the -source option must have been specified on the hyd_analysis command to create the files with the source strengths.

The surface elevation is stored in the array .hyd_surf_disp(nnode,nfreq,nbeta), where nnode is the number of surface nodes (hyd_surf_nodes command).

The purpose of this command is to generate the surface elevations for export to graphics programs for visualization; see the command hyd_export_graphics.

NOTE: this command is only implemented for the case of no symmetry.

See Also

hyd_analysis hyd_export_graphics hyd_parameters hyd_surf_nodes
hyd_surf_panel

hyd_surf_nodes

Command Syntax

```
hyd_surf_nodes  #=?
n=node_no  x=x-coor  y=y-coor  z=z-coor  [lgen=lgen]
```

Reads and generates surface nodal coordinates in inertial coordinates. The surface nodal coordinates are used to plot surface elevation only. The value specified by # is used to define storage requirements, and it must be greater than or equal to the maximum node number. If this value is missing or 0, it is assumed that existing nodes are being changed or added to, and the previous value applies. lgen is the node number increment for linear generation. Nodes are generated equally spaced along a straight line if two adjacent records do not have sequential node numbers and if lgen on the second line is not zero or blank. Nodes need not be input in sequence.

End input with a blank line.

This command must precede the hyd_surf_panel command.

The coordinates are stored in array .hyd_surf_xyz(3,#), and the maximum possible node number (specified by #) is stored in .hyd_#surf_nodes_tot.

Active nodes are those that are defined explicitly either by this command or another command that creates surface nodes. The node number of the maximum defined surface node is stored in .hyd_#surf_nodes. The character vector .hyd_surf_node_active has an "A" for active nodes. Only active nodes can be used.

The hyd_surf_nodes command need not be executed as long as the coordinates, which could be generated by another program, are put in the array .hyd_surf_xyz, and .hyd_#surf_nodes and .hyd_#surf_nodes_tot are set, and .hyd_surf_node_active is created.

NOTE: Surface nodes cannot be on body panels (as specified by the hyd_nodes and hyd_panel commands. Locate these surface nodes (and panels) a small distance away from the actual body. If a surface node does lie on the body, the surface elevation for that node will likely be reported as NaN.

See Also

hyd_surf_node_gen hyd_surf_node_tolerance phyd_surf_nodes

phyd_surf_nodes

Command Syntax

```
phyd_surf_nodes  [nodes=?,?]  [-screen]
```

Print surface nodal coordinates. A range of node numbers can be specified by nodes=. The first value is the first node number in the range, and the second value is the last number in the range. The default is to print the coordinates for all nodes defined. The default is to print to output file only; if -screen is present, then output is to the screen as well.

See Also

hyd_surf_nodes

hyd_surf_node_gen

Command Syntax

```
hyd_surf_node_gen
linear=node1,node2  [inc=?]  [w=?]
quad=node1,node2,node3,node4  [inc=inc1,inc2]  [w=w1,w2]
```

Generates nodal coordinates. The hyd_surf_nodes command must preceed this command.

Linear generation is specified by the identifier linear. Nodes are generated from node1 to node2 with a node increment of inc (default=1). The spacing of nodes is equidistant unless the positive weight w (default=1) is specified, in which case the spacing between node n and n+1 is equal to w times the spacing between n-1 and n.

The identifier quad specifies node generation within the "quadrilateral" defined by the four nodes (see sketch below). Linear generation is done between node1 and node2, and between node3 and node4, based on inc1 and w1. Linear generation is also done between node1 and node3, and between node2 and node4, based on inc2 and w2. Then, linear generation is done between the nodes generated from 1-3 and 2-4, based on inc1 and w1. The number of interior lines generated is the same as the number of nodes generated from 1-3. If the number of nodes generated from 1-3 is larger than the number generated from 2-4, the "extra" lines will not be generated, as this would result in a redefinition of nodal coordinates along 3-4. If node3 and node4 are identical, the generated nodes are within a triangular domain. The nodes need not be coplanar.

End input with a blank line.

```
       node3                                        node4
         X----------O----------O----------O----------X
         |                                           |
         |                                           |
         |                                           |
         |                                           |
   node1 |       node1+inc2                          |
   +inc2 O          O +inc1    O           O         O
  ^      |                                           |
  |      |                                           |
generate |                                           |
   w/    |                                           |
 inc2    |                                           |
         |                                           |
         |          generate w/ inc1 ->              |
         X----------O----------O----------O----------X
       node1    node1+inc1                         node2
```

X = specified node
O = generated node

See also

hyd_surf_nodes

hyd_surf_node_tolerance

Command Syntax

```
hyd_surf_node_tolerance  [z_tol=?]  [x_tol=?]  [y_tol=?]
```

Checks for and corrects slight errors in nodal coordinates. This command checks for nodes that should be on the still water plane or planes of symmetry, but numerical precision-related errors cause them to be slightly off these planes. This can happen when coordinates are generated by a mesh generation or CAD program, or when input coordinates are transformed. Surface nodes must have z-coordinate of 0, so this command enforces that requirement. A warning is printed if the input z-coordinate is further away than ztol. For single or double symmetry, if a nodal y-coordinate is within y_tol of the x-z plane, it is set to 0. For double symmetry, if a nodal x-coordinate is within x_tol of the y-z plane, it is set to 0. The default for each tolerance is 0.001. Tolerances should be positive; if a negative value is specified, the default of .001 is used. (Note: the symmetry type must have been specified by the hyd_parameters command for x_tol and y_tol to be used.

See Also

hyd_surf_nodes

hyd_surf_panel

Surface panel for surface elevation. There are two Command Syntax options.

Command Syntax (option 1)

```
hyd_surf_panel n=?
n=nel  nodes=node1,node2,node3,node4  [gen=gen  inc=inc1,inc2]       &
        [gen_2d=gen_2d  inc_2d=inc1_2d,inc2_2d  inc_el=inc_el]
```

n is the maximum panel (element) number

nel is the element number
node1 thru node4 are node numbers
inc1, inc2 are node increments in a "linear sequence"
gen is the number of elements to generate in a sequence
inc1_2d, inc2_2d are node increments between sequences
gen_2d is the number of linear sequences to generate
inc_el is the element increment between sequences

Nodes 1 to 4 are the corner nodes for quadrilateral elements. For triangular elements, node 4 should either be 0 or equal to node 3. The nodes are specified clockwise, looking at the surface from above (see sketch below).

Command Syntax (option 2)

```
hyd_surf_panel  -subdivide  range=?,?  nxm=?,?
```

Option 2 subdivides previously defined panels
    range specifies a range of panel ID numbers; all panels
        in the range are divided
    nxm specifies how many panels to subdivide each panel into.
        For example, nxm=2,3 would subdivide each panel into 6
        elements; 2 in the 1-2 direction and 3 in the 1-4 direction.

A "linear sequence" of elements can be generated by specifying inc1, inc2 and gen. In a linear sequence, nodes 1 and 2 are incremented by inc1; and nodes 3 and 4 are incremented by inc2. gen is the number of elements to generate, so a sequence will have gen+1 elements. To generate a 2D patch of elements, multiple sequences can be specified; inc1_2d and inc2_2d are used to increment the node numbers from one sequence to the next. gen_2d is the number of additional sequences. The element numbers in two successive sequences differ by inc_el (default = numgen+1).

End input with a blank line.

On input, created arrays are:

```
.hyd_surf_panel_el(4,n)    -> node1 - node4
.hyd_#surf_panels(1)       -> total # of defined panels
.hyd_surf_panel_code(n)    -> see below
.hyd_surf_paneltype(nwet)  -> = 0 for quad; = 1 for triangle
.hyd_surf_panel_#map(n)    -> map panel numbers to internal numbers
```

where nwet are the number of wet panels (see below).

These panels are only used when exporting the surface elevation to graphics programs for visualization. They are not used for any calculations. The surface elevations are calculated at the surface nodes. The vector .hyd_panel_code contains a character code: blank -> panel not defined; W -> wet panel (defined); These panels are numbered internally. The mapping from the "external" numbering to the internal numbering is in the vector .hyd_surf_panel_#map.

The hyd_surf_nodes command must precede this command.

```
   node2                                          node3
     X-------------------------------------------X
     |                                           |
     |                                           |
     |                                           |
     |                                           |
     |                                           |
     |                                           |
     |                                           |
     |                                           |
     |                                           |
     |                                           |
     |                                           |
     X-------------------------------------------X
   node1                                          node4
```

See Also

phyd_surf_nodes phyd_surf_panel

phyd_surf_panel

Command Syntax

```
phyd_surf_panel
```

Print hyd_surf_panel element data.

Only defined panels will be printed.

See Also

hyd_surf_panel

hyd_tf

Calculate transfer functions

Command Syntax

```
  hyd_tf [arg1  arg2]  [ext=extension  file=filename]  [-radians]   &
         [-period]  [-RAO]
```

After the response analysis has been carried out with the hyd_analysis_response command, transfer functions for any response quantity can be obtained with this command. arg1 is the name of a user-defined 'modal matrix' of dimension (ncomp,nmode), where ncomp is the number of response components and nmode is the number of assumed modes used in the analysis. Column i of arg1 contains the response of each component in mode i. The default name for arg1 is hyd_modemat. The transfer functions are obtained by multiplying arg1 with the generalized coordinates calculated previously by the hyd_analysis command. The transfer functions are put in the complex array whose name is specified by arg2 (default name is hyd_tf). The command creates this array with dimensions (nfreq,nbeta,ncomp), where nfreq and nbeta are the number of frequencies and wave angles, respectively.

The transfer functions (magnitude and phase angle) are written to a file. If extension is specified, the file name is project_name.extension. If filename is specified, the file will have the name specified by filename. The default file is project_name.tf. If -radians is specified, the phase angle will be written in radians; otherwise, it will be written in degrees. If -period is specified, the transfer functions will be written as a function of wave period rather than wave frequency. If -RAO is specified, only the RAO (magnitude) will be printed. The default is to print both RAO and phase angle.

Note: transfer functions for the generalized displacements can be generated by specifying the matrix arg1 as an nmode x nmode identity matrix. If arg1 is not specified, and the default matrix (hyd_modemat) does not exist, then an identity matrix named hyd_modemat will be created and the transfer functions for the generalized displacements will be generated. If this is done for the 6 rigid body modes, for example, the transfer functions for rotations will be in radians. To obtain transfer functions in degrees, arg1 (hyd_modemat) should have the appropriate conversion factors. For example,the following commands would create such a matrix:

```
ident hyd_modemat r=6 c=6
pi PI
setr tmp v=180
scale tmp PI -inv
put tmp hyd_modemat r=4 c=4
put tmp hyd_modemat r=5 c=5
put tmp hyd_modemat r=6 c=6
rm tmp PI
```

Because of the defaults, the command

```
hyd_tf
```

is equivalent to

```
hyd_tf  hyd_modemat  hyd_tf  ext=tf
```

As many hyd_tf commands as desired can be used.

See Also

```
  hyd_analysis_response  hyd_irregular
```

hyd_velocity

Determine the fluid velocity after a hydrodynamic analysis.

Command Syntax

```
hyd_velocity del=delx,dely,delz  [incoming=?]  [diffraction=?]        &
            [radiation=?]  [dif=difname]  [rad=radname]  [-keep_pot]
```

delx,dely,delz are the differences in the coordinates to obtain the gradient of the potentials. If only delx is specified, dely and delz are set to delx.

```
incoming    = 1 -> include the incoming potential (default)
            = 0 -> do not include the incoming potential
diffraction = 1 -> include the diffraction potential (default)
            = 0 -> do not include the diffraction potential
radiation   = 1 -> include the radiation potential (default)
            = 0 -> do not include the radiation potential
-keep_pot   if present, then the velocity potentials are kept
```

If dif= is specified, the diffraction source strengths are read from file difname; otherwise they are read from file project_name.dif.

If rad= is specified, the radiation source strengths are read from file radname; otherwise they are read from file project_name.rad.

The hyd_analysis and hyd_velocity_nodes commands must precede this command. As of now, this command has only been implemented for the case of no symmetry (hyd_parameters command). Also, the -source option must have been specified on the hyd_analysis command to create the files with the source strengths.

The del values are stored in .hyd_vel_del(3).

The velocities are calculated for the nodes in .hyd_vel_xyz(3,nnodes) as follows. First, six points around the nodes, at +-delx, +-dely, and +-delz (in that order) are defined and stored in .hyd_vel6_xyz(3,6*nnodes). If a node is within delz of the still water plane or seafloor, the node is used as the point instead. The potentials for these 6 points are then calculated. The velocities are obtained as, for example, velocity_x = (phi(x+delx,y,z) - phi(x-delx,y,z)/(2 * delx). (The expression is adjusted if the points are not 2*delx apart.) The x,y,z velocity components are stored in .hyd_vel_nodes(3,nnodes,nfreq,nbeta), which is complex. nfreq and nbeta are the number of wave frequencies and angles, respectively. The velocity potentials are stored in .hyd_vel6pot(6*nnodes,nfreq,nbeta). This array is usually destroyed, but if -keep_pot is used, it will be kept in the database.

NOTE: this command is only implemented for the case of no symmetry.

See Also

hyd_analysis hyd_parameters hyd_velocity_nodes

phyd_velocity

Command Syntax

```
phyd_velocity  [-period]
```

Print fluid velocities. If -period is specified, the velocities will be written as a function of wave period rather than wave frequency.

See Also

hyd_velocity

hyd_velocity_nodes

Command Syntax

```
hyd_velocity_nodes  #=?
n=node_no  x=x-coor  y=y-coor  z=z-coor  [lgen=lgen]
```

Reads and generates inertial coordinates of "nodes" at which to calculate fluid velocities. The value specified by # is used to define storage requirements, and it must be greater than or equal to the maximum node number. If this value is missing or 0, it is assumed that existing nodes are being changed or added to, and the previous value applies.

lgen is the node number increment for linear generation. Nodes are generated equally spaced along a straight line if two adjacent records do not have sequential node numbers and if lgen on the second line is not zero or blank. Nodes need not be input in sequence.

End input with a blank line.

This command must precede the hyd_velocities command.

The coordinates are stored in array .hyd_vel_xyz(3,#), and the total number of defined nodes is put in the scalar .hyd_#vel_nodes. This may be less than # specified on the command line, which is stored in .hyd_#vel_nodes_tot. Currently, the defined nodes are assumed to be sequential from 1 to .hyd_#vel_nodes.

The hyd_velocity_nodes command need not be executed as long as the arrays created by this command are defined "manually".

NOTE: Velocity nodes cannot be on body panels (as specified by the hyd_nodes and hyd_panel commands). Locate the velocity nodes over del away from the actual body (see help on hyd_velocity). In addition, they cannot be above the still water line or below the sea floor. It is recommended to run the command hyd_velocity_node_tolerance before hyd_velocity.

See Also

hyd_velocity_node_gen hyd_velocity_node_tolerance phyd_velocity_nodes

phyd_velocity_nodes

Command Syntax

```
phyd_velocity_nodes  [nodes=?,?]  [-screen]
```

Print velocity nodal coordinates. A range of node numbers can be specified by nodes=. The first value is the first node number in the range, and the second value is the last number in the range. The default is to print the coordinates for all nodes defined. The default is to print to output file only; if -screen is present, then output is to the screen as well.

See Also

hyd_velocity_nodes

hyd_velocity_node_gen

Command Syntax

```
hyd_velocity_node_gen
linear=node1,node2  [inc=?]  [w=?]
quad=node1,node2,node3,node4  [inc=inc1,inc2]  [w=w1,w2]
```

Generates nodal coordinates. The hyd_velocity_nodes command must preceed this command.

Linear generation is specified by the identifier linear. Nodes are generated from node1 to node2 with a node increment of inc (default=1). The spacing of nodes is equidistant unless the positive weight w (default=1) is specified, in which case the spacing between node n and n+1 is equal to w times the spacing between n-1 and n.

The identifier quad specifies node generation within the "quadrilateral" defined by the four nodes (see sketch below). Linear generation is done between node1 and node2, and between node3 and node4, based on inc1 and w1. Linear generation is also done between node1 and node3, and between node2 and node4, based on inc2 and w2. Then, linear generation is done between the nodes generated from 1-3 and 2-4, based on inc1 and w1. The number of interior lines generated is the same as the number of nodes generated from 1-3. If the number of nodes generated from 1-3 is larger than the number generated from 2-4, the "extra" lines will not be generated, as this would result in a redefinition of nodal coordinates along 3-4. If node3 and node4 are identical, the generated nodes are within a triangular domain. The nodes need not be coplanar.

End input with a blank line.

```
        node3                                        node4
          X----------O----------O----------O----------X
          |                                           |
          |                                           |
          |                                           |
          |                                           |
    node1 |       node1+inc2                          |
    +inc2 O          O +inc1    O           O         O
   ^      |                                           |
   |      |                                           |
 generate |                                           |
    w/    |                                           |
  inc2    |                                           |
          |                                           |
          |          generate w/ inc1 ->              |
          X----------O----------O----------O----------X
        node1    node1+inc1                         node2
```

X = specified node
O = generated node

See also

hyd_velocity_nodes

hyd_velocity_node_tolerance

Command Syntax

```
hyd_velocity_node_tolerance  [z_tol=?]  [x_tol=?]  [y_tol=?]
```

Checks for and corrects slight errors in nodal coordinates. This command checks for nodes that are above the still water plane, below the seafloor, or should be on the planes of symmetry, but numerical precision-related errors cause them to be slightly off these planes. This can happen when coordinates are generated by a mesh generation or CAD program, or when input coordinates are transformed. Velocity nodes must have z-coordinates in the range -water depth <= z <= 0. This command enforces that requirement. A warning is printed if the input z-coordinate is adjusted. For single or double symmetry, if a nodal y-coordinate is within y_tol of the x-z plane, it is set to 0. For double symmetry, if a nodal x-coordinate is within x_tol of the y-z plane, it is set to 0. The default for each tolerance is 0.001. Tolerances should be positive; if a negative value is specified, the default of .001 is used. (Note: the symmetry type must have been specified by the hyd_parameters command for x_tol and y_tol to be used.

See Also

hyd_velocity_nodes

hyd_wave

Specify wave frequencies (or periods) and angles for the hydrodynamic analysis.

Command Syntax

```
hyd_wave nbeta=? nfreq=? [-freq] [-period] [-descending] [-delete]

  nbeta  = the number of wave angles (default = 1)
  nfreq  = the number of incoming wave frequencies (default = 1)
  If -period is specified, input wave periods in seconds (default)
  If -freq is specified, input wave frequencies in rad/sec
  If -descending is specified, the input frequencies or periods are
  sorted in descending order; otherwise they are put in ascending order.
```

Following the command, input the wave angles (in degrees) in 1 record and the frequencies/periods in a second record.

If -delete is specified, then nfreq wave frequencies or periods entered on the next line are deleted from the list of frequencies/periods previously input (wave angles are not input and cannot be deleted). At least one frequency must remain after deletion. Corresponding data such as added mass, potentials, and generalized coordinates are also deleted. However, transfer functions generated by the hyd_tf command are not modified. hyd_tf should be re-run after frequencies are deleted.

The data are stored as:

```
.hyd_wave(2)          -> nbeta, nfreq
.hyd_beta(nbeta)      -> wave angles in radians
.hyd_freq(nfreq)      -> wave frequencies in radians/sec
```

See Also

hyd_analysis

hyd_wave_dispersion

Command to determine the wave number and wave length

Command Syntax

```
hyd_wave_dispersion
```

Given the water depth from the hyd_parameters command, and the wave frequencies from the hyd_wave command, this command solves the dispersion equation to report the wave number and the wave length for each frequency.

See Also

hyd_parameters hyd_wave

hyd_wave_spectra

Specify wave spectra

Command Syntax

```
hyd_wave_spectra #=nspectra
n=?  name=?  (Additional data depends on wave spectrum; see below)
```

Reads the data to specify nspectra wave spectra. n is the spectrum number (1 to nspectra) and name is the spectrum name. Allowable spectra and their input are:

```
name=Bretschneider  Hs=significant_wave_height  To=modal_period

name=ISSC  Hs=significant_wave_height  Tv=visual_period

name=JONSWAP  To=modal_period  gamma=gamma  Xo=Xo
              (default gamma=3.3; default Xo=0)

name=ITTC  Hs=significant_wave_height  k=k (default=1)

name=P-M-Wind  U=wind_speed  (Pierson-Moskowitz)

name=P-M-Hs  Hs=significant_wave_height  (Pierson-Moskowitz)
```

End input with a blank line.

For those spectra that require the gravitational constant, the value specified via the hyd_parameters command is used.

Note: the modal period is the inverse of the (cyclic) frequency at which the frequency spectrum is a maximum.

This command stores the data in the array .hyd_wave_spectrum(6,nspectra).

See Also

hyd_irregular hyd_parameters phyd_wave_spectra

phyd_wave_spectra

Print wave spectra

Command Syntax

```
phyd_wave_spectra  [-spectrum]
```

Print the wave spectra information specified by the hyd_wave_spectra command. If the wave frequencies have been defined, e.g., by the hyd_wave command, then some statistics, such as calculated significant wave height, for the wave spectra are also reported. The statistics are approximate based on the wave frequencies and the trapezoidal rule of integration. In addition, if -spectrum is specified, the wave spectrum is also printed to the output file.

See Also

hyd_wave hyd_wave_spectra

hyd_wet

Command to estimate the "wet" natural frequencies and mode shapes

Command Syntax

```
hyd_wet  [shift=?] [error=?] [form=format]
```

The "wet" natural frequencies and mode shapes are estimated by solving the eigenvalue problem, for each wave frequency, using the added mass that was saved in the database from a previous hyd_analysis command.

Shift is an eigenvalue shift that is used to handle zero eigenvalues. The default shift is 1. Error is the maximum error allowed for a wet frequency, as explained below. Form is the Fortran format in which to write the mode shapes (no spaces are allowed in the character string).

An estimate of the error in the calculated wet frequency omega_n is

```
|omega - omega_n|/omega*100%
```

where omega is the wave frequency. Only those frequencies omega_n for which the estimated error is less than or equal to the value of error on the command line are reported. The default value for error is very large, so that all omega_n are reported.

Before the hyd_wet command is given, the structural mass and stiffness matrices must be defined as database members and named hyd_mstr and hyd_kstr, respectively. In addition, the hydrostatic stiffness .hyd_kf must be defined (do not forget the dot!). The dimensions of these matrices should be (nmode,nmode), where nmode is the number of modes. In the typical case, these matrices will have been defined already as part of the wave response analysis.

The "wet" natural frequencies are written to the output file as a function of wave frequency. Both the natural frequencies and mode shapes are also written to file probname.wet. In general, the mode shapes are in terms of the generalized coordinates. However, if the matrix hyd_modemat(ncomp,nmode) exists, the modes will be expressed in terms of the ncomp components. That is, the mode shapes will be multiplied by hyd_modemat, and the result will be written to *.wet. If hyd_modemat exists, the default format for writing is (i15,3x,1pe14.4); otherwise, the default format is (i17,1x,1pe14.4). The integer refers to the component/mode number. This format can be overwritten by specifying a format in the command line. The format must be of the same form as the default and without any blanks. This option is meant to allow the output of more digits for postprocessing. An example is: form=(i15,3x,1pe20.10). Note that the parentheses are required.

The following main arrays are created in the database:

```
.hyd_wet_freq(nmode)             -> wet natural frequencies
.hyd_wet_modes(nmode,nmode)      -> wet mode shapes
.hyd_wet_freq_cumulate(*)        -> converged wet natural frequencies
.hyd_wet_modes_cumulate(nmode,*) -> converged wet modes
```

Note that the first two matrices will contain the data only for the last wave frequency considered, while the second two accumulate the converged natural frequencies and mode shapes over all wave frequencies, i.e., those results that meet the convergence criterion.

Of course, the number of converged frequencies is unknown prior to the analysis.

See Also

hyd_analysis

2.2. General Commands

break_loop

Command Syntax

```
break_loop
```

Break out of a do/while loop. Note: in the case of nested loops, this command breaks out of the entire loop.

See Also

do while

date

Command Syntax

```
date  [arg]  [-noprint]
```

Prints the current date and time right justified on an 80 column page. If arg is specified, the date and time are also stored in a character array. If -noprint is specified, the date is not printed.

See Also

time

do

Command Syntax

```
do arg1 f=? l=? [s=?]
    block of commands
end_do
```

arg1 is the name of the loop variable. f is the first value, l is the last value, and s is the increment (default = 1). If arg1 is a real or integer matrix in the database, the first value is used, and the comparison is based on the type of arg1. If arg1 does not exist, an integer scalar with that name is created; it will not be deleted at completion.

While and do commands may have a combined nesting of 15 levels. The following restrictions apply:

> Do/while loops in separate files may not be active at one time. Specifically, if a do/while loop is in an input file, the filein command must not be issued from inside a do/while loop.
>
> A break_loop command causes the exit of an entire do/while loop, not just the "subloop" in the case of nested loops.
>
> A return command must not occur inside the body of a do/while loop. A return command may occur in an input file read inside a loop, however, to cause the input file to be exited.

See Also

break_loop if while

filein

Command Syntax

```
filein  [filename]  [-noecho]
```

Opens the file "filename" and reads commands for "batch" execution.

If filename is not given, the program tries to open a file with the same name as the project, but with the extension .txt; i.e., .txt is appended to the project name. If that fails, it tries to open the file without the extension.

If filename is specified, the extension .txt, if it exists, can be omitted.

NOTE: The input file must be a plain text file with Windows/Macintosh line endings on Windows/Macintosh platforms, respectively.

If the flag -noecho is present, the input is not echoed to the screen.

Execution returns to the previous input file or to interactive mode when "return" or end-of-file is read.

Although the actual number depends on the number of files the operating system/compiler allows to be open at one time, the program supports a nesting of filein commands up to 28 levels; that is, a filein command can occur in another input file.

The login command is ignored in input files.

See Also

login   return

flush

Command Syntax

```
flush  [unit=]
```

Flush the buffer for unit. If unit is not specified, standard out buffer is flushed (i.e., the buffer for file project_name.out).

This command is useful, for example, to force writing to the output file without closing the program.

help

Command Syntax

```
help (h) command [-f] [l=?]
```

On-line help for command. For an index of available commands use the command index.

If -f is given, then output is written to the output file; otherwise output is to the screen only. If l is specified, then l lines will be scrolled at a time.

Optional parameters are indicated as such by being enclosed in []. The [] are not part of the command.

if

Command Syntax

```
if arg1 [operator arg2]
    block of commands
[else]
    [block of commands]
endif
```

arg1 must be a real or integer matrix in the database. The first value in the matrix is used in the comparison.

The simplest form is when only arg1 is specified. In this case, a value of 0 is considered false, and any other value is true.

The more general case is when operator and arg2 are specified. Operator is one of the following FORTRAN conditional operators:

```
.eq.  .ne.  .lt.  .le.  .gt.  .ge.
```

If arg2 is a real or integer matrix in the database, then the comparison is with the first element of arg2. If arg1 and arg2 are of different types, the conditional is evaluated as a real. If arg2 is not found in the database, it is interpreted as a number of the same type as arg1. In this case, only one space may separate the operator and arg2.

If commands may be nested.

See Also

do while

index

Command Syntax

`index [topic] [-f] [l=?]`

Provides a one-line summary of available commands. If topic is given, topics are general, database, matrix, functions, fe_commands, fem_elements, and misc.

If -f is given, then output is written to the output file; otherwise output is to screen only. If l is specified, then l lines will be scrolled at a time.

logfile

Command Syntax

`logfile  [-on]  [-off]`

Turns writing to the log file on or off. Writing is always turned on at the start of a project, and the default option is to turn it on.

See Also

login

login

Command Syntax

`login  [-noecho]`

The file “project_name.log" contains a log of all commands entered in interactive mode. Since the log file is always appended, it contains a complete history of a previous run or sequence of runs (unless logging of commands has been turned off).

The log file can be processed via the login command, which will open the file "project_name.log" as an input file. In this case, a quit in an input file causes a return to the log file, while it is ignored if it is in the log file itself. A login command is only accepted in interactive mode.

If the flag -noecho is given, the input is not echoed to the screen.

See Also

filein logfile

name?

Command Syntax

`name? [-f]`

Prints project name. If -f is given, then output is written to the output file; otherwise output is to screen only.

new_project

Command Syntax

`new_project (or newproj or newprob)`

Initiates a new project.

See Also

quit save savequit

palias

Command Syntax

`palias`

Print command aliases.

quit

Command Syntax

`quit (q)`

Stops execution. This command is ignored if it is read from the log file via the login command.

See Also

login save savequit

read

Command Syntax

`read  arg filename  [-ext]`

Read array arg from filename. If -ext is present, filename is taken to be an extension, and the file project_name.filename is read. It is assumed that the file was created by the write command (unformatted option) and is in the proper form.

See Also

write

rename_file

Command Syntax

`rename_file  filename1  filename2`

Rename filename1 to filename2.

WARNING: Do not rename an open file. In general, commands close files once they finish. Files project_name.out and project_name.log remain open until the program terminates or a new project is opened.

See Also

rm_file

return

Command Syntax

`return`

A "batch" command that closes the input file opened with the filein command. Causes a return to wherever the filein command was issued (the previous input file or interactive mode).

See Also

filein login

rm_file

Command Syntax

`rm_file filename  [-ext]`

Remove file filename. If -ext is present, filename is taken to be an extension, and the file project_name.filename is deleted.

See Also

read rename_file write

savequit

Command Syntax

`savequit (sq)`

Saves the data base in the file "project_name.db" and stops execution, if not read from the log file via the login command.

See Also

save quit

system_command

Command Syntax

`system_command (term or comm) character_string`

Issues a command to the operating system. The aliases term (for terminal) and comm (for command prompt) are provided because typing system_command is a bit unwieldy. The format of character_string, which contains the command, is different between Macintosh and Windows.

Macintosh

An example of the command is

```
term 'cp  name.out  "name copy.out"'
```

Note that single quotes are required if there are spaces in character_string, but they are not allowed within character_string.

Windows

An example of the command is

```
term 'cmd.exe /c cp name.out  "name copy.out"'
```

Note that single quotes are required at the beginning and end of character_string, and they are not allowed within character_string. The /c is required to terminate the command. The command should also work without the ".exe".

time

Command Syntax

```
time  arg1  [arg2  arg3]
```

Puts the number of seconds since the start of execution in arg1. If arg2 and arg3 are given, arg2 is assumed to be the result of an earlier call to time, and the difference between arg1 and arg2 will be put in arg3. The arguments are real scalars.

Note: This command uses the Fortran function cpu_time.

See Also

date

while

Command Syntax

```
while arg1 [operator arg2]
   block of commands
end_while
```

arg1 must be a real or integer matrix in the database. The first value in the matrix is used in the comparison.

The simplest form is when only arg1 is specified. In this case, a value of 0 is considered false, and any other value is true.

The more general case is when operator and arg2 are specified. Operator is one of the following FORTRAN conditional operators:

```
.eq.  .ne.  .lt.  .le.  .gt.  .ge.
```

If arg2 is a real or integer matrix in the database, then the comparison is with the first element of arg2. If arg1 and arg2 are of different types, the conditional is evaluated as a real. If arg2 is not found in the database, it is interpreted as a number of the same type as arg1. In this case, only one space may separate the operator and arg2.

While and do commands may have a combined nesting of 15 levels. The following restrictions apply:

Do/while loops in separate files may not be active at one time. Specifically, if a do/while loop is in an input file, the filein command must not be issued from inside a do/while loop.

A break_loop command causes the exit of an entire do/while loop, not just the "subloop" in the case of nested loops.

A return command must not occur inside the body of a do/while loop. A return command may occur in an input file read inside a loop, however, to cause the input file to be exited.

See Also
  break_loop do if

write

Command Syntax

```
write arg filename  [-ext]  [-f]  [-tab]  [-a]
```

Write array arg to filename. If -ext is present, filename is assumed to be an extension to the project name, and the file project_name.filename will be written. The default file type is binary in which the information is written: array type (4 byte integer), dimension (4 byte integer), dimension integers specifying # rows, # cols, etc (each 4 bytes), and then the data. If -f is specified, then a formatted write of the data only is done. If -tab is also specified, then the columns will be tab separated. If -a is specified, then the file will be appended.

See Also
  read rm_file

2.3. Database Commands

clear

```
    Command Syntax
      clear
        Clear (initialize) the data base.

    See Also
      rm   save
```

dir

```
    Command Syntax
      dir (or ll or ls) [arg]  [-f]  [c=?]
        dir (or ls) or ll produce a short or long listing of arrays in memory,
        including the dimensions.
          If arg is given, information on that array only is given.
          If -f is given, writes to output file. Otherwise, output is only to
          screen.
          The output for dir (ls) will be in c columns (default = 4)

    See Also
      memory
```

ll

```
    Command Syntax
      dir (or ll or ls) [arg]  [-f]  [c=?]
        dir (or ls) or ll produce a short or long listing of arrays in memory,
        including the dimensions.
          If arg is given, information on that array only is given.
          If -f is given, writes to output file. Otherwise, output is only to
          screen.
          The output for dir (ls) will be in c columns (default = 4)

    See Also
      memory
```

memory

```
    Command Syntax
      memory
        Prints data memory used in database
```

mv

```
    Command Syntax
      mv (rename) arg1 arg2
        Move (rename) arg1 to arg2. If arg2 exists, it is first removed.

    See Also
      rm
```

readdb

```
    Command Syntax
      readdb  [arg]
        Reads the database in the file "arg.db" and adds to the existing
        database. If arg is not given, the file “project_name.db" is read.

    See Also
      save
```

rm

```
    Command Syntax
      rm (del) arg1
        Remove (delete) arg1 from database. Multiple arguments can be
        specified.

    See Also
      clear  mv   save
```

rm*

```
    Command Syntax
      rm* (del*) arg*
        Remove (delete) all members from database that begin with "arg". For
        example,

        rm* mat*

        will delete all members that start with "mat". Note that the last
        character of the argument must be *.

        The command syntax requiring * in both the command and the argument is
        a bit awkward. For example, it would be easier not to include the * in
        the argument. However, this command could be a bit dangerous in that it
        might make it easy to delete members accidentally. The syntax is meant
        to ensure that the intention is to delete everything that matches.

    See Also
      clear  mv  rm  save
```

save

```
    Command Syntax
      save  [arg1]
        Saves the data base to the file "arg1.db." If arg1 is not give, it is
        saved to the file “project_name.db."

    See Also
      quit  readdb  savequit
```

2.4. Matrix Commands

add

```
Command Syntax
  add arg1 arg2 [arg3] [r=?] [n=?] [c=?] [m=?]
    Adds arg2 to arg1.
      If arg3 is input, result is put in new matrix arg3.
      If r is input, start adding at row r.
      If n is input, only add n rows.
      If c is input, start adding at column c.
      If m is input, only add m columns.
```

Note: For 3-D arrays, only straight addition is supported. If any parameters are specified, they are ignored.

See Also
sub   subcol   sumcol

arpack

```
Command Syntax
  arpack  [#=?] [which=?] [maxit=?] [digits=?] [shift=?] [minv=?] [-freq]
    By using ARPACK routines, determine the eigenvalues and eigenvectors of
    the generalized eigenvalue problem:

               [K] {phi} = omega^2 [M] {phi}

      #      = # of eigenvalues to determine
      which  = Specify which of the Ritz values to compute
               LA - compute # smallest (algebraic) eigenvalues
               SA - compute # largest  (algebraic) eigenvalues
               LM - compute # smallest (in magnitude) eigenvalues (default)
               SM - compute # largest  (in magnitude) eigenvalues
               BE - compute # eigenvalues, half from each end of the
                    spectrum. When # is odd, compute one more from the
                    high end than from the low end.
      maxit    = max. # of iterations (default = 30)
      digits = convergence tolerance = 10^-digits (default = 8)
      minv   = minimum number of subspace vectors (default = min(2*#,#+8))
      shift  = frequency shift (default = 0)
      -freq -> the frequencies (omega) are determined
```

K and M are assumed to be symmetric matrices in sparse storage. The diagonals of K and M are in .sparse_diag and .sparse_diag2; the off-diagonals in .sparse_off and .sparse_off2, and the index arrays are .sparse_ptr and .sparse_indx. A diagonal M may be used, wherein .mstr is the vector of diagonal masses (do not provide .sparse_diag2 and .sparse_off2 in this case).

The eigenvalues (frequencies) are put in .omega2 (.omega), depending on the parameter -freq. The eigenvectors are put in .phi.

If K is singular then a shift must be applied prior to calling this command. For this command, the shifted eigenvalues are equal to the actual eigenvalues + shift. (That is, if a shift is needed for frequency analysis, use a positive shift.)

The ARPACK routines were obtained from Rice University; for more information see http://www.caam.rice.edu/software/ARPACK/index.html

See Also

eigval jacobi ptosparse

array3d_slice

Command Syntax

```
array3d_slice arg1 arg2 [r=?] [c=?] [t=?]
```

Extracts a "plane" of a 3-D array.

arg1(n,m,p) is a 3-D array and arg2 is a matrix. arg2 is created as follows:

```
If r is specified, arg2(1:m,1:p) = arg1(r,1:m,1:p)
If c is specified, arg2(1:n,1:p) = arg1(1:n,c,1:p)
If t is specified, arg2(1:n,1:m) = arg1(1:n,1:m,t)
```

Hence, if t is specified, arg2 is "table" t from arg1. This can also be obtained from the cp command.

See Also

array3d_unslice cp

array3d_unslice

Command Syntax

```
array3d_unslice arg1 arg2 [r=?] [c=?] [t=?]
```

Inserts 2-D matrix arg1 into a "plane" of 3-D array arg2. This operation is the opposite of array3d_slice. The two dimensions of arg1, n and m, must agree with two of the three dimensions of arg2 as follows.

```
If r is specified, arg2(r,1:n,1:m) = arg1(1:n,1:m)
If c is specified, arg2(1:n,c,1:m) = arg1(1:n,1:m)
If t is specified, arg2(1:n,1:m,t) = arg1(1:n,1:m)
```

Note: Only one of r, c, and t should be given. If two or more are specified, only one is used, in the priority order r, c, t.

See Also

array3d_slice

cp

Command Syntax

```
cp (copy) arg1 arg2 [r=?] [n=?] [c=?] [m=?] [table=?]
```

Copies arg1 to arg2. For a straight copy of any dimension array, do not include any parameters. A partial copy of an array can be made by specifying parameters.

If r is input, start copying at row r.
If n is input, only copy n rows.
If c is input, start copying at column c.
If m is input, only copy m columns.
If table is input, operate on one table of a 3-D array; i.e., extract the table.

```
        For 3D arrays, this command can only duplicate the array (no parameters
        specified) or extract one table, or part thereof.

    See Also
      cpdg   put   putdg
```

cpdg

```
    Command Syntax
      cpdg arg1 arg2 [r=?] [n=?]
        Copies the diagonal elements of arg1 to arg2.
          If r is input, start copying at row r.
          If n is input, only copy n rows.

    See Also
      cp  put  putdg
```

diag_mult

```
    Command Syntax
      diag_mult arg1 arg2 arg3
        Multiply arg1 times arg2. arg1 is a diagonal matrix stored as a vector.
        The result is stored in arg3. The dimension of arg1 must equal the
        number of rows of arg2.

    See Also
     mult  tmult  mult_elem
```

dim_reduce

```
    Command Syntax
      dim_reduce arg1 arg2 index=?

        Copies all but the last dimension of a multi-dimensional array. If arg1
        is an n-dimensional array, arg2 will be an (n-1)-dimensional array that
        is obtained by setting the last index of arg1 to the value specified by
        index. The default for index is 1.
```

eigval

```
    Command Syntax
      eigval  [#=?] [maxit=?] [digits=?] [shift=?] [ss_size=?]              &
              [rigid=?,?,?,?,?,?  cg=?,?,?  node_range,?,?] [-vec_init]     &
              [-freq] [-disk] [-sparse] [-random] [seed=?]
        Eigenvalue solution by either subspace (SS) iteration or Jacobi method
        of the generalized eigenvalue problem:

                    [K] {phi} = omega^2 [M] {phi}

          #          = 0 -> use Jacobi and solve for all eigenvalues
                     > 0 -> use SS to solve for # of lowest eigenvalues
          maxit      = max. # of iterations (default = 30)
          digits     = convergence tolerance = 10^-digits (default = 8)
          ss_size    = number of subspace vectors (SS only; default=max(2#,#+8)
          shift      = frequency shift (default = 0)
          rigid      = six values, corresponding to 6 rigid body modes in
                       global coordinates (SS iteration only)
```

```
                 0 -> do not form corresponding rigid body mode
                 1 -> form corresponding rigid body mode
    cg         = point about which rigid modes are calculated
                 (default = 0,0,0)
    node_range = range of nodes to calculate rigid modes
                 (default is to include all nodes)
    -vec_init -> use #vecs initial vectors in matrix
                 eigval_init(neq,#vecs)- SS only
    -freq     -> the frequencies (omega) are determined
    -disk     -> the factored and original stiffnesses will be swapped
                 to/from disk for SS interation (profile only)
    -sparse   -> a sparse stiffness matrix is used

    -random   -> use a random number generator to create
                 the starting vectors (SS iteration only)
    seed     .ge. 0 -> seed for generator (default = 1)
             .lt. 0 -> use next random number
```

K and M are assumed to be symmetric matrices. The default option is to use an upper profile storage scheme. In this case, K and M are assumed to be in .kstr and .mstr, respectively. The locations of the diagonal elements are assumed to be in .kdiag_loc. If a sparse storage option is chosen, then the diagonals of K and M are in .sparse_diag and .sparse_diag2; the off-diagonals in .sparse_off and .sparse_off2, and the index arrays are .sparse_ptr and .sparse_indx. In either case a diagonal M may be used, wherein .mstr is the vector of diagonal masses.

The eigenvalues (frequencies) are put in .omega2 (.omega), depending on the parameter -freq. The eigenvectors are put in .phi.

For subspace iteration, K and M are unchanged if there is no shift; if there is a shift, .kstr will be the shifted matrix.

For both SS and Jacobi, if K is singular then a shift must be applied. For this command, the shifted eigenvalues are equal to the actual eigenvalues + shift. (In other words, for frequency analysis typically use a positive shift.)

For the jacobi solution, M must be positive definite.

See Also

arpack jacobi ptosparse

extract

Command Syntax

```
extract arg1 arg2  c=?  v=? [tol=?] [-eq -ne -gr -gt -le -lt]
```

Extracts rows in arg1 for which the value (v=) in column (c=) matches the criterion:

```
-eq -> equal to value +- tol (default)
-ne -> not equal to value
-ge -> greater than or equal to value
-gt -> greater than value
-le -> less than or equal to value
-lt -> greater than or equal to value
```

The result is put in arg2. A row vector is treated as a column vector.

For complex numbers, value is complex but the comparison is done on absolute values, except for equality and inequality.

fft

Command Syntax

```
fft  arg1  [-ferziger]  [-ooura]  [-forward] [-inverse]
```

The default is to determine the forward (direct) Fourier transform of a function arg1. arg1 is replaced with the frequency coefficients. If -inverse is specified, the inverse transform is determined, in which case the input arg1 should be the frequency coefficients and the output arg1 is the time history.

arg1 must be an N x m dimensional array of equally spaced function values. N is the number of sample points and m is the number of sequences to be transformed.

Default Algorithm

The default is to use fftpack available at netlib.org that was originally written by Paul N. Swarztrauber, as modified by P. Bjorstad (version 3, June 1979). The command creates the work array $fft_wsave, which must not be changed between calls to fft. That is, the inverse transform will use the previously determined $fft_wsave vector created by the forward transform. This array can be deleted if no further inverse transformations with the same value of N are to be done. This command assumes arg1 is a real matrix for a real function.

Ooura Algorithm

If option -ooura is specified, the Ooura FFT routines available at http://faculty.prairiestate.edu/skifowit/fft/ are used (July 2010). arg1 can be either double precision real or double precision complex. N must be a power of two. The work arrays $fft_ip and $fft_w are used, which can be deleted when they are no longer needed.

Ferziger Algorithm

If option -ferziger is specified, the FFT routine by Wouk based on the algorithm by Ferziger in Numerical Methods for Engineering Applicatons is used, but converted to double precision. The original code is available at http://www.netlib.no/netlib/misc/fft.f (July 2010). arg1 is double precision complex. N must be a power of two. No work array is used.

You should clearly understand the scaling, the convention, and the storage scheme that these algorithms use. A first step is to refer to the simple test file in the examples folder.

See Also

fft_helper

fft_helper

Command Syntax

```
fft_helper  H  P  HP
```

This is a special purpose function to multiply transfer functions H with frequency components P, with the result HP, which is stored in a form consistent with P.

The use of this routine is designed for dyanamic structural analysis, where the work flow is as follows: 1) use the command fft to obtain the frequency coefficients P from the time history of the loading; 2) multiply the transfer functions H with the corresponding P, forming HP; 3) use fft to obtain carry out an inverse Fourier transform to obtain the time history of the response.

In theory both H and P are complex and the same size, in which case a simple element-by-element multiply is sufficient. But this is not usually the case. It is more likely that H is complex of dimension N/2+1, in which N is the number of points used for the discrete Fourier transform (this command requires that N is even). If the command fft with the default algorithm is used to obtain P, then the complex P is stored in a real matrix of dimension N. The storage scheme for P is revealed by the example fft in the folder distr_examples. This command will do the multiplication directly. The complex product is stored in the real matrix HP, using the same storage scheme as used for P. This is the storage scheme required by the default algorithm in the command fft to obtain the inverse Fourier transform.

Multiple transfer functions can be processed at the same time. In summary, the input and output are as follows:

```
H(N/2+1,#func) - complex, #func is the number of transfer functions
P(N)           - real
HP(N,#func)    - real
```

N must be even.

A typical analysis would then involve three commands:

```
fft P
fft_helper H  P  HP
fft HP -inverse
```

In the first command, input P is the loading sampled at N equidistant time points and output P is the frequency components. The second command is as explained above. In the third command, output HP is the time history of the response at the N equidistant time points.

See Also

fft

ftopro

Command Syntax

`ftopro  arg1  arg2  arg3`

Store "full" symmetric matrix arg1 in row vector arg2 using upper profile storage. arg3 is the vector of diagonal locations in arg2. Length of arg3 is arg1_rows + 1. Length of arg2 is arg3(arg1_rows+1) - 1.

See Also

ptoful pmult

gauss

Command Syntax

`gauss arg1 arg2  [-pp]  [ #digits=?]`

Uses elimination (LU factorization) to solve a system of linear equations (real or complex). arg1 is the coefficient (square) matrix and arg2 is the "RHS." The solution is put in arg2.

For real matrices, a warning is printed if the number of "digits" lost in a diagonal is greater than or equal to #digits (default = 7).

If -pp is specified, partial pivoting with implicit scaling is carried out; otherwise, elimination without pivoting is done.

For large, symmetric matrices, use psolve.

See Also

ftopro invert psolve

get_dim

Command Syntax

`get_dim arg1 arg2  [-rows]  [-cols]`

Gets the dimensions of arg1 and stores them in arg2. If -rows is present, only the number of rows is stored. If -cols is present, only the number of columns is stored.

ident

Command Syntax

`ident (identi or identc) arg [r=?] [c=?]`

Transform matrix arg such that all terms are zero except arg(i,i), the "diagonals", which are one. If arg is square, it becomes an identity matrix.

If r and c are input, integer (identi), real (ident), or complex (identc) matrix arg(r,c) is created.

See Also

input   zero

input

Command Syntax

`input (inputi or inputch or inputc) arg [r=?] [c=?]  [-null]`

Input a real (input), integer (inputi), character (inputch) or complex (inputc) matrix or vector.

r is the number of rows (default = 1).

c is the number of columns (default = 0).

For character matrices and vectors, the rows are null-terminated if the parameter -null is specified.

See Also

ident input3d zero

input3d

Command Syntax

```
input3d (input3di or input3dc) arg [r=?] [c=?] [t=?]
```

Input a real (input3d), integer (input3di), or complex (input3dc) 3-D array with dimensions (r,c,t).

r is the number of rows (default = 1).
c is the number of columns (default = 1).
t is the number of "tables" (default = 1).

A 3-D array is a collection of 2-D matrices, each of which is considered a "table." That is, arg(:,:,1) is table 1, arg(:,:,2) is table 2, and so forth. Each table is input as a 2-D matrix (see the command input), and the tables are input sequentially.

See Also

Input

interpolate

Command Syntax

```
interpolate  tin  fin  tout  fout  [prepad=?]  [postpad=?]
```

Interpolate using the data set (tin,fin) to create (tout,fout).

The input data set is defined by tin(n) and fin(n,nfunc), where tin is a vector of real "time" values, fin is a matrix of real or complex function values. The output values are defined at the discrete "times" specified in tout(N). The real (or complex) vector fout(N,nfunc) is created based on linear interpolation of the data in [tin,fin].

If the range of tout is larger than the range of tin, the output values are pre-padded and/or post-padded. The default is to prepad with the 1st value in fin and to postpad with the last value in fin. If a value is given for the argument prepad, prepadding will use that value. If a value is given for the argument postpad, postpadding will use that value. (Use these options to prepad and postpad with 0, for example.)

It is assumed that tin and tout are in ascending order.

See Also

series

invert

Command Syntax

```
invert arg1 arg2
```

The inverse of arg1 is calculated and stored in arg2. Straight Gauss elimination is used. The matrix arg1 is modified. For large, symmetric matrices, use psolve.

See Also

ftopro psolve

jacobi

Command Syntax

```
jacobi k m phi lambda  [d=?] [maxit=?]  [digits=?] [-freq]
```

Eigenvalue solution by Jacobi method for the generalized eigenvalue problem:

```
[K] {phi} = lambda [M] {phi}
```

where K and M are symmetric, square, positive definite matrices, which will be modified, and

```
phi    = matrix of all eigenvectors (to be created)
lambda = vector of all eigenvalues (to be created)
d      = 0 -> mass matrix is input as 2-D matrix(default)
       = 1 -> diagonal mass "vector" (not implemented)
maxit  = maximum # of iterations (default = 30)
digits = # digits accuracy in eigenvalues (default = 8)
-freq  = if present, then lambda is the square root of the
         eigenvalues.
```

See Also

Eigval

join

Command Syntax

`joinh (or joinv) arg1 arg2 arg3`

arg3 is formed by joining (concatenating) arg1 and arg2
joinh (horizontal joining) puts the matrices "side-by-side."
joinv (vertical joining) puts arg1 "on top of" arg2.

max

Command Syntax

`max arg1 arg2  [c=?]  [-abs]`

Finds the maximum value in column c=? of arg1; the corresponding row is put in vector arg2. The default is c=1. If the flag -abs is present, the element with the maximum absolute value is found. If arg1 is a row vector, it is treated as a column vector.

See Also

min norm

min

Command Syntax

`min arg1 arg2  [c=?]`

Finds the minimum value in column c=? of arg1; the corresponding row is put in vector arg2. If arg1 is a row vector, it is treated as a column vector.

See Also

max

mult

Command Syntax

`mult (or tmult) arg1 arg2 arg3`

For mult, arg3 = arg1 * arg2
For tmult, arg3 = Transpose(arg1) * arg2

See Also

mult_col mult_elem   scale

mult_col

Command Syntax

```
mult_col arg1 arg2 arg3
```

Multiply the columns of arg1 and the columns of arg2, and store the result in the row vector arg3. Hence, arg3(i) is the dot product of columns i of arg1 and arg2. The dimensions of arg1 and arg2 must be identical.

See Also

mult mult_elem

mult_elem

Command Syntax

```
mult_elem arg1 arg2 [arg3] [-inv]
```

Multiply each element of arg1 with the corresponding element of arg2, and store the result in arg3. If arg3 is not given, the result is stored in arg1. If -inv is specified, then the elements of arg1 are divided by arg2. The dimensions of arg1 and arg2 must be identical.

See Also

mult mult_col

norm

Command Syntax

```
norm arg1 arg2  [-max] [-l2]
```

Finds the vector "norm" of the columns of arg1; results are put in vector arg2. The default is to find the maximum value in each column. If the flag -max is present, the maximum absolute value of the elements is found. Otherwise, if the flag -l2 is present, the square root of the sum of the squares is found. If arg1 is a row vector, it is treated as a column vector.

See Also

sumcol

pmult

Command Syntax

```
pmult  arg1  arg2  arg3  arg4
```

Multiply arg1 * arg2 = arg3, where arg1 is a symmetric matrix stored in upper profile form. arg4 is the vector of diagonal locations in arg1. Length of arg4 is arg2_rows + 1

See Also

ptoful ftopro

print

Command Syntax

```
print or printf (p or pf) arg1 [r=?] [c=?] [cols=?] [l=?]         &
    [-nohead] [row#=?] [-f] [form=format] [-phase] [table=?]
```

Prints arg1. Printing begins at row r and column c (default = 1). If l is given, then l lines will be scrolled at a time. If -nohead is given, then the usual row/column headings will not be printed. The default is to print every fifth row number (1, 5, 10, 15, etc.) in the heading;

this can be overridden by specifying a different value with row#=. If -f is specified, then printing is to the output file only (and not to the screen). In this case, scrolling is not done. If -phase is specified and arg1 is complex, then the magnitude and phase angle (degrees) will be printed rather than the real and imaginary components.

The default output formats are ni6 and 1pne12.4, where n=12, n=6, or n=3 for integer, real, and complex matrices, respectively. If cols is specified, n=cols. (pf prints using nf12.5 format.) Alternative formats can be specified by the parameter form=. In this case, format must be a valid FORTRAN format, enclosed in (), without any embedded blanks and with a maximum length of 160 characters. If a format is specified, the parameters cols=, l=, and -nohead are ignored (row/column headings are not printed).

If arg1 is a 3 dimensional array, then it is printed as a sequence of 2 dimensional arrays, called "tables." For example, table 1 is arg1(*,*,1). If table is specified, only that table is printed. For dimensions greater than 4, only the first table is printed.

See Also

xprint

psolve

Command Syntax

`psolve arg1 arg2 arg3 [digits=?]`

Solves arg1 * arg2 = arg2, where arg1 is a symmetric matrix stored in upper profile form. arg2 is the "load" vector, with neq rows, which is replaced by the solution. arg3 is the vector of diagonal locations in arg1. Length of arg3 is neq + 1. The size of arg1 = arg3(neq+1) - 1. A warning is printed if the loss of precision in a diagonal element is greater than or equal to digits (default = 7).

See Also

ftopro gauss invert pmult ptoful

psolve16

Compiler does not support real*16

ptoful

Command Syntax

`ptoful arg1 arg2 arg3`

Put symmetric matrix arg1 stored in profile form into the square matrix arg2. arg3 is the vector of diagonal locations in arg1. The length of arg3 is assumed to be the # diagonals of arg1 + 1. The dimensions of arg2 are arg2(#diag,#diag).

See Also

ftopro gauss invert pmult

ptosparse

Command Syntax

`ptosparse  arg1  arg2  [arg3]`

Put symmetric matrix arg1 stored in profile form, with diagonal locations specified in arg2, into sparse matrix. The diagonals are stored in .sparse_diag and the nonzero off-diagonals are stored in .sparse_off. The number of nonzeroes in each row are stored in .sparse_ptr, and the column number for each nonzero is stored in .sparse_indx. The number of nonzero off-diagonals is stored in .sparse_size. If arg3 is specified, then it is interpreted as a matrix with the same profile as arg1. Its values are stored in .sparse_diag2 and .sparse_off2.

See Also

sparse_mult

put

Command Syntax

`put arg1 arg2 [r=?] [c=?]`

Puts arg1 into arg2.

If r is input, starts putting at row r.

If c is input, starts putting at col c.

See Also

putdg

putdg

Command Syntax

`putdg arg1 arg2 [r=?]`

Puts arg1 into diagonals of arg2.

If r is input, starts putting at row r.

See Also

put

scale

Command Syntax

`scale arg1 [arg2 e=?,?]  [v=?] [-inv]`

Multiplies arg1 by the scalar arg2(e1,e2) or by the value v. If -inv is present, multiplies by the inverse of the scalar.

See Also

mult

series

Command Syntax

`series arg  x=x1,x2  [inc=?]  [w=?]  [max=?]`

Creates a real data series starting at x1 and ending at x2. The series is put in the row vector arg. The initial spacing is inc (default=1); the spacing between points 2 and 3 is inc * w, and the spacing between points n and n+1 is inc * w^(n-1). However, the spacing between the last two points may differ, as the last value will not be larger than x2. If w < 1, care must be taken to ensure that the series will reach x2. However, at most max (default=100000) number of points are generated. By specifying a large enough x2, one can generate a series with exactly max points.

See Also

series2d

series2d

Command Syntax

```
series2d arg  p1=x1,y1,z1 p2=x2,y2,z1 p3=x3,y3,z1 p4=x4,y4,z1     &
    [w=?,?]  [n=?,?]  [dim=?]
```

Creates a real data series on a 2-d grid, which is put in the matrix arg(n1*n2,dim). The series data can be viewed as points in an X-Y-Z coordinate system, which are generated within the "quadrilateral" defined by the corner points pj with coordinates (xj,yj,zj), j=1,4; see sketch below. A grid of n1 x n2 points is generated, as follows. Linear generation is used to generate n1 points between p1 and p2, and between p3 and p4, based on w1. Linear generation also is used to generate n2 points between p1 and p3, and between p2 and p4, based on w2. The points generated between p1 and p3 are connected through linear generation, based on n1 and w1, with the points generated between p2 and p4. A gridlike ordering of points is carried out such that p1 is point 1, p2 is point n1, p3 is point n1*(n2-1)+1, and p4 is point n1*n2. If n2 is zero, then generation is along the line p1 to p2 only.

dim can be 1, 2 or 3 (default=2). Only the first dim coordinates are kept in arg.

If arg exists, and if the # rows = n1*n2 and the # columns is at least dim, then the data are put in the first dim columns of the existing matrix.

Linear generation is defined such that the initial spacing between points is equidistant unless the positive weight w (default=1) is specified, in which case the spacing between points k and k+1 is equal to w times the spacing between k-1 and k.

An alternative type of generation can be carried out using the commands nodes and node_gen.

```
p3                    p4
X-----------------------X
|                       |
|                       |
|                       |
|                       |
|                       |
|                       |
|                       |
X-----------------------X
p1                      p2
```

See Also

series

seti

Command Syntax

```
seti (setr) arg v=?  or
```

setch arg s=? [-null]

Set scalar arg to value. Seti and setr set integer and real scalars, respectively, equal to the value v. Setch sets a character string (a row matrix with a maximum length of 80, no blanks allowed) specified by s. If -null is present, the character string in arg will be null-terminated.

This command is more convenient than input and inputi for scalars and inputch for character constants.

sort

Command Syntax

sort arg1 [arg2] [key=?,?,?] [-reverse]

Sorts arg1 in ascending order. A maximum of three sort keys (columns on which sorting is done) may be specified by the identifier key; the first value is the first key, etc. If no keys are specified, the first key is taken to be column 1. The result is put in arg2, if it is given; otherwise arg1 is changed. If -reverse is present, sort is in descending order. If arg1 is a row vector, it is sorted as a column vector.

sparse_matrix_clean

Command Syntax

sparse_matrix_clean

Removes arrays created by the command ptosparse as well as some arrays that are created internally as part of the sparse solver. Arrays that will be removed if they exist are: .sparse_mmik, .sparse_mmim, .sparse_mmfn, .sparse_addr, .sparse_ptr, .sparse_indx, .sparse_diag, .sparse_diag2, .sparse_off, .sparse_off2, .sparse_ordr, and .sparse_size.

See Also

lsolve ptosparse

sparse_mult

Command Syntax

sparse_mult arg1 arg2

Premultiplies the matrix arg1 by a (symmetric) sparse matrix and puts the result in arg2. The sparse matrix is assumed to be in a form compatible with the command ptosparse, and the arrays .sparse_diag, .sparse_off, .sparse_ptr, .sparse_indx, and .sparse_size are assumed to exist. The array .sparse_ordr defines the reordering of the matrix; if it does not exist, it is assumed that the matrix has not been reordered.

See Also

ptosparse

split

Command Syntax

split arg1 [arg2] [c=?]

Splits arg1 based on columns and joins the matrices vertically into arg2, with c columns (default=1). For example, let arg1 have 10 rows and 8 columns and let c=4. Then arg2 will have 20 rows and 4 columns, such that the first 10 rows will be the same as the first 4 columns of arg1, and the last 10 rows will be the same as the last 4 columns of arg1. If arg2 is not given, then arg1 is replaced with the new matrix. If c does not divide evenly into the # of columns of arg1, then the additional columns in the last rows of arg2 will be zero. If c is greater than or equal to the # of columns of arg1, a simple copy of arg1 is done. This command could be used to form a column vector from a row vector by specifying c=1. (The transpose command - trans - is preferred in this case.)

See Also

unsplit

sub

Command Syntax

```
sub arg1 arg2 [arg3] [n=?] [r=?] [c=?] [m=?]
  Subtracts arg2 from arg1.
    If arg3 is input, result is put in new matrix arg3.
    If r is input, start subtracting at row r.
    If n is input, only subtract n rows.
    If c is input, start subtracting at column c.
    If m is input, only subtract m columns.
```

See Also

subcol add sumcol

subcol

Command Syntax

```
subcol arg1 arg2 arg3 [c=?]
```

Subtracts arg2 from arg1.

Result is put in new matrix arg3. If c is input, start subtracting at column c. This command is used to subtract all but the first c-1 columns of two equal size matrices.

See Also

sub   add   put   putdg

sumcol

Command Syntax

```
sumcol arg1 arg2  [-abs]
```

Sums the columns of arg1; results are put in vector arg2. If the flag -abs is present, the absolute values of the elements are summed. If arg1 is a row vector, it is treated as a column vector.

See Also

add   sub   subcol

to_complex

Command Syntax

```
to_complex arg1 [arg2] or
to_complex2 arg1 arg2 arg3
```

The command to_complex converts agr1 into a complex matrix. arg1 becomes the real part of arg2, if it is given; otherwise arg1 is replaced.

The command to_complex2 creates the complex array agr3 using arg1 as the real part and arg2 as the imaginary part.

See Also

`to_int` `to_real`

to_complex2

Command Syntax

```
to_complex arg1 [arg2] or
to_complex2 arg1 arg2 arg3
```

The command to_complex converts agr1 into a complex matrix. arg1 becomes the real part of arg2, if it is given; otherwise arg1 is replaced.

The command to_complex2 creates the complex array agr3 using arg1 as the real part and arg2 as the imaginary part.

See Also

`to_int` `to_real`

to_int

Command Syntax

```
to_int arg1 [arg2]  [-trunc]
```

Converts arg1 into an integer matrix. The values in arg1 are converted to the nearest 4 byte integers and placed in arg2, if it is given; otherwise arg1 is replaced. If -trunc is given, the values are truncated instead (for complex arg1, the values are always truncated).

See Also

`to_complex` `to_real`

to_real

Command Syntax

```
to_real arg1 [arg2]
```

Converts agr1 into a real matrix. The values in arg1 are converted to 8 byte reals and placed in arg2, if it is given; otherwise arg1 is replaced. If arg1 is complex, the real part is returned.

See Also

`to_complex` `to_int`

to_real16

Command Syntax

```
to_real16 arg1 [arg2]
```

Converts agr1 into a real*16 matrix. The values in arg1 are converted to 16 byte reals and placed in arg2, if it is given; otherwise arg1 is replaced.

Note: Only limited support is provided for real*16 data types.

See Also

psolve16 to_complex to_int to_real

to_vector

Command Syntax

```
to_vector arg1
```

Converts a matrix with only one dimension greater than 1 to an N-dimensional vector. For example, A(N,1) and A(1,N) will both be converted to A(N).

tmult

Command Syntax

```
mult (or tmult) arg1 arg2 arg3
  For mult, arg3 =  arg1 * arg2
  For tmult, arg3 = Transpose(arg1) * arg2
```

See Also

mult_col mult_elem   scale

trans

Command Syntax

```
trans arg1 [arg2]
```

Transposes arg1 to arg2, if it is given. Otherwise, replaces arg1 with its transpose.

unsplit

Command Syntax

```
unsplit  arg1  [arg2]  [r=?]
```

Unsplits arg1 based on rows and joins the matrices horizontally into arg2, with r rows (default=1). For example, let arg1 have 20 rows and 4 columns and let r=10. Then arg2 will have 10 rows and 8 columns, such that the first 4 columns will be the same as the first 10 rows of arg1, and the last 4 columns will be the same as the last 10 rows of arg1. If arg2 is not given, then arg1 is replaced with the new matrix. If r does not divide evenly into the # of rows of arg1, then the additional rows in the last columns of arg2 will be zero. If r is greater than or equal to the # of rows of arg1, a simple copy of arg1 is done. This command could be used to form a row vector from a column vector by specifying r=1. (The transpose command - trans - is preferred in this case.)

See Also

split

unwrap

Command Syntax

```
unwrap  arg1  [arg2]  [r=?]
```

"Unwraps" the rows of arg1 puts the result into arg2, which will have r rows (default=1). For example, let arg1 have 20 rows and 4 columns and let r=10. Then arg2 will have 10 rows and 8 columns, such that the first row will be the same as the first 2 rows of arg1, row 2 will be the same as rows 3 and 4 of arg1, etc. If arg2 is not given, then arg1 is replaced with the new matrix. If r is greater than or equal to the # of rows of arg1, a simple copy of arg1 is done. Otherwise, r must divide evenly into the number of rows in arg1.

See Also

split unsplit wrap

wrap

Command Syntax

`wrap  arg1  [arg2]  [c=?]`

"Wraps" the rows of arg1 and puts the result into arg2, which will have c columns (default=1). For example, let arg1 have 10 rows and 8 columns and let c=4. Then arg2 will have 20 rows and 4 columns, such that the first 2 rows will be the same as the first row of arg1, rows 3 and 4 will be the same as the second row of arg1, etc. If arg2 is not given, then arg1 is replaced with the new matrix. If c does not divide evenly into the # of columns of arg1, then the additional columns in arg2 will be zero. If c is greater than or equal to the # of columns of arg1, a simple copy of arg1 is done.

See Also

split unsplit unwrap

xprint

Command Syntax

`xprint (xp) arg1  [cols=?] [row#=?]`

Expanded print of arg1 with virtually all significant digits. If cols is given, then a maximum of cols columns will be printed at a time. row# sets the increment used to print row headings (default = 5).

If t is given, arg is a 3-D array rather than a matrix, but the rules above apply.

See Also

print

zero

Command Syntax

`zero (zeroi or zeroc) arg  [v=value]  [r=?  c=??  t=?]`

Zeroes arg.
If value is given, the matrix is filled with value rather than zero. If r or c is input, matrix arg(r,c) is created of type real (zero), integer (zeroi), or complex (zeroc).

See Also

input   ident

2.5. Mathematical Functions

abs

Command Syntax

`abs arg1 [arg2] [x=?] [y=?] [c=?]`

Takes absolute value of elements in arg1. The result is put in arg2, if it is given; otherwise the values in arg1 are replaced.

The default is to operate on every element in arg1. However, to operate on a single column only, use the parameter x to specify the column number. If x is specified and arg2 is not specified, the result is put in column y of arg1 (default=x). If c is specified and x is not, the first c-1 columns are the same as arg1.

bessel_j or bessel_y

Command Syntax

`bessel_j (or bessel_y) [n=] arg1 [arg2] [x=?] [y=?] [c=?]`

Takes Bessel function of 1st kind (bessel_j) or Bessel function of the 2nd kind (bessel_y) of order n of elements in arg1. The result is put in arg2, if it is given; otherwise the values in arg1 are replaced.

n=0 is the default

The default is to operate on every element in arg1. However, to operate on a single column only, use the parameter x to specify the column number. If x is specified and arg2 is not specified, the result is put in column y of arg1 (default=x). If c is specified and x is not, the first c-1 columns are the same as arg1.

conjugate

Command Syntax

`conjugate arg1 [arg2] [x=?] [y=?] [c=?]`

Calculates the complex conjugates of elements in arg1. The result is put in arg2, if it is given; otherwise the values in arg1 are replaced.

The default is to operate on every element in arg1. However, to operate on a single column only, use the parameter x to specify the column number. If x is specified and arg2 is not specified, the result is put in column y of arg1 (default=x). If c is specified and x is not, the first c-1 columns are the same as arg1.

epsilon

Command Syntax

`epsilon arg1 arg2`

Returns a positive small value relative to 1 of the same real kind as arg1. arg2 will be a scalar.

erf

Command Syntax

`erf (or erfc or erfc_scaled) arg1 [arg2] [x=?] [y=?] [c=?]`

Takes error function (erf), complementary error function (erfc) or the scaled complementary error function (erfc_scaled) of elements in arg1. The result is put in arg2, if it is given; otherwise the values in arg1 are replaced.

The default is to operate on every element in arg1. However, to operate on a single column only, use the parameter x to specify the column number. If x is specified and arg2 is not specified, the result is put in column y of arg1 (default=x). If c is specified and x is not, the first c-1 columns are the same as arg1.

exp

Command Syntax

`exp arg1 [arg2] [x=?] [y=?] [c=?]`

Calculates the exponential of elements in arg1. The result is put in arg2, if it is given; otherwise the values in arg1 are replaced.

The default is to operate on every element in arg1. However, to operate on a single column only, use the parameter x to specify the column number. If x is specified and arg2 is not specified, the result is put in column y of arg1 (default=x). If c is specified and x is not, the first c-1 columns are the same as arg1.

gamma

Command Syntax

`gamma arg1 [arg2] [x=?] [y=?] [c=?]`

Takes the gamma function of elements in arg1. The result is put in arg2, if it is given; otherwise the values in arg1 are replaced.

The default is to operate on every element in arg1. However, to operate on a single column only, use the parameter x to specify the column number. If x is specified and arg2 is not specified, the result is put in column y of arg1 (default=x). If c is specified and x is not, the first c-1 columns are the same as arg1.

log

Command Syntax

`log arg1 [arg2] [x=?] [y=?] [c=?]`

Calculates the natural log of elements in arg1. The result is put in arg2, if it is given; otherwise the values in arg1 are replaced.

The default is to operate on every element in arg1. However, to operate on a single column only, use the parameter x to specify the column number. If x is specified and arg2 is not specified, the result is put in column y of arg1 (default=x). If c is specified and x is not, the first c-1 columns are the same as arg1.

log10

Command Syntax

`log10 arg1 [arg2] [x=?] [y=?] [c=?]`

Calculates the common log of elements in arg1. The result is put in arg2, if it is given; otherwise the values in arg1 are replaced.

The default is to operate on every element in arg1. However, to operate on a single column only, use the parameter x to specify the column number. If x is specified and arg2 is not specified, the result is put in column y of arg1 (default=x). If c is specified and x is not, the first c-1 columns are the same as arg1.

pi

Command Syntax

`pi arg1`

Creates the scalar arg1 with the value of pi.

power

Command Syntax

`power arg1 [arg2] [p=?] [-r] [x=?] [y=?] [c=?]`

Raises each element in arg1 to the power p (default = 2). If -r is present, the power p is real; otherwise, it is an integer (default). The result is put in arg2, if it is given; otherwise the values in arg1 are replaced.

The default is to operate on every element in arg1. However, to operate on a single column only, use the parameter x to specify the column number. If x is specified and arg2 is not specified, the result is put in column y of arg1 (default=x). If c is specified and x is not, the first c-1 columns are the same as arg1.

sqrt

Command Syntax

`sqrt arg1 [arg2] [x=?] [y=?] [c=?]`

Takes square root of elements in arg1. The result is put in arg2, if it is given; otherwise the values in arg1 are replaced.

The default is to operate on every element in arg1. However, to operate on a single column only, use the parameter x to specify the column number. If x is specified and arg2 is not specified, the result is put in column y of arg1 (default=x). If c is specified and x is not, the first c-1 columns are the same as arg1.

trig

Command Syntax

`"trig_function" arg1 [arg2] [x=?] [y=?] [c=?]`

Evaluates the function "trig_function" for each element in arg1.The result is put in arg2, if it is given; otherwise the values in arg1 are replaced. Valid functions are

```
cos  sin  tan  asin  acos  atan  sinh  cosh  tanh
```

The default is to operate on every element in arg1. However, to operate on a single column only, use the parameter x to specify the column number. If x is specified and arg2 is not specified, the result is put in column y of arg1 (default=x). If c is specified and x is not, the first c-1 columns are the same as arg1.

2.6. Finite Element Commands

bcid

Command Syntax

```
bcid [#=?]  [r=?,?,?...]  [print=?]
n=node_no  r=?,?,?...  [print=?]  [inc=inc]
```

Specifies restraints on nodal degrees-of-freedom (DOFs).

```
  #     = number of DOFs per node (default = 6)
  r     = comma-separated string of 0's and 1's, # values long
          0 -> DOF is free (value to be calculated)
          1 -> DOF is "fixed", i.e., value is set to 0
  print = print code
          0 -> displacements will not be printed by pdisp
          1 -> displacements will be printed by pdisp (default)
```

Values specified on the bcid line are nodal defaults. The most common situation is when there are 6 DOFs per node. r=0,0,0,0,0,0 would then correspond such that 1 to 3 are for the translations along the X, Y, and Z axes, and 4 to 6 are the corresponding rotations. For example, r=0,0,1,1,1,1 would imply only X and Y displacements are non-zero. Some elements require more than 6 DOFs per node; this can be specified by #=. Note that if a number less than 6 is specified, 6 will be used. That is, there are at least 6 DOFs per node.

Subsequent lines can be used to specify different values for specific nodes. However, all nodes have the same maximum number of DOFs, which is stored in .#dofs_per_node. If a restraint value < 0 is specified instead of a 0 or 1, then the corresponding displacement for the node is the same as for node number |value|. Generation occurs if two records do not have sequential nodes and inc, the node number increment, is not 0. The generated nodes have the same restraints as the last node entered. Nodes need not be input in sequence.

End input with a blank line.

The displacement restraints are stored in array .bcid(#,#nodes), which is required by the num_eqs or form_k command to number the equations, and the pdisp command to print displacements. The print codes are stored in .bcid_print(#nodes).

The term "displacement" is used for the nodal variables, but actually it can be any quantity (e.g., temperature, etc.).

See also

form_k nodes num_eqs pbcid pdisp

body_frc2d

Command Syntax

```
body_frc2d  n=?
n=distribution#  c=coefficients
```

Input polynomial coefficients used to define n "distributions" of 2d body forces for elements in a plane. A maximum of a 5th degree polynomial in two variables (21 coefficients) is allowed. The coefficients are stored in the array .body_f2d(21,n). These can then be

```
      used by elements to define body forces.

      End input with a blank line.

  See Also
    d2l3to9  pbody_frc2d
```

check_diag

```
  Command Syntax
    check_diag  arg1  arg2  [zero=?] [v=?]  [-noprint]
      Check the diagonal elements of matrix arg1, stored in profile form, for
      zeroes. arg2 is the vector of diagonal locations in arg1. The length of
      arg2 is assumed to be the # diagonals of arg1 + 1. If zero (default =
      0.0) is given, then any diagonal with an absolute value less than zero
      is replaced by v (default = 1.0). The equation numbers corresponding to
      the changed diagonals are reported unless the the flag -noprint is
      given. This command is useful to prevent a stiffness matrix from being
      singular as a result of "unconnected" nodes.

  See Also
    form_k
```

conc_deck_loads

```
  Command Syntax
    conc_deck_loads
    element=? f=? dist=? [DG=?] [y=?]

      Input vertical concentrated deck loads for pontoon bridge element.

        element is the element number on which the load acts
        f is the load magnitude
        dist is the distance, from the beginning of the element, to the load
        DG is the vertical distance from the deck to the center of gravity of
        the load
        y is the transverse offset from the center line of the element

    End input with a blank line.

    As many concentrated loads as desired may be entered for a given element.
    They are accumulated.

    Concentrated load data are stored in the array .pbridge_conclds(5,*), in
    the order element #, f, dist, DG, y.

    This command, if used, must be given after the elements are defined with
    the pbridge command.

  See Also
    distr_deck_loads  pbridge  pdeck_loads
```

consolidation

```
  Command Syntax
    consolidation steps=? [dt=?] [theta=?] [dprin=?] [fprin=?] [k=?] [time=?]
                  [history=?]
      Consolidation time integration command
        steps   = number of time steps
        dt      = time step (default is to use existing time step)
        theta   = 2-step integration parameter (default=1.0 -> forward Euler)
        dprin   = print flag for nodal displacements
                  = 0 -> not printed from command (default)
                  = N -> print every N time steps
        fprin   = print flag for element state
                  = 0 -> not printed from command (default)
                  = N -> print every N time steps
        k       = 0 -> form stiffness by calling routine form_k (default)
                = 1 -> use existing stiffness
        time    = initial time (default = 0)
        history = time increment at which results are saved (see below)
```

A form_k command must precede the first consolidation command. It is also important that the nodef command follow the form_k command.

The increment in external loads is determined from the .load_pat array created by the command nodef. For this command, the load "patterns" refer to the time step. Imposed displacements and pressures are specified by the imposed_displ command. Again, for this case the pattern number is the time step. This command interprets the values specified in the nodef and the imposed_displ command as incremental values.

At each time step, the calculated displacements and pressures are placed in .disp. If history is not zero, then the displacements at every history time units (e.g., seconds) is saved in .disp_history(neq+1,*). The last value in each column is the time of the results. For each node, DOF 4 is the pressure.

Note: Using an existing stiffness (k=1) should only be done if the time step and theta parameter are unchanged and the degrees-of-freedom with imposed displacements have not changed.

See Also

imposed_displ nodef

cp_tables

Command Syntax

```
  cp_tables  t=?  maxdrafts=?  maxtrims=?
    t=?  submerged=?
    1st row is trim angles in degrees
    draft  Cp1 Cp2 etc.
```

Input tables of drag coefficients, Cp, for pbridge element. The command line parameter t specifies the number of tables. maxdrafts and maxtrims are the maximum number of drafts and trims that will be input, considering all tables. The first input record for a table is the table number and the value of Cp when the element is submerged (draft and/or trim is outside the range of the table). The subsequent records contain the draft, and then up to maxtrims Cp values for that draft. The drafts and trims must be in increasing numerical order.

The Cp tables are stored in the array .cp_tables(maxdrafts+2,maxtrims+1,t). The last row in the matrix contains the actual number of drafts and trims for that table, as well as the submerged Cp value. Therefore, there are at least 3 columns in .cp_tables.

End input with a blank line.

current_velocity

Command Syntax

```
current_velocity  [v=?]  [angle=?]  [-variable]
[ n=node#  v=?  angle=? [inc=?] ]
```

v   = current speed
angle = angle in degrees the current velocity vector makes with the global X-axis, measured counterclockwise

This command defines the current velocity to be used with the pontoon bridge element (pbridge). For a uniform current, the speed and angle are specified on the command line. If -variable is present, a current that varies in the horizontal (X-Y) plane can be specified. In this case, the current speed and angle at each node with a nonzero current are specified on records immediately following the command record. Generation occurs if two records do not have sequential nodes and inc, the node number increment, is not 0. The current for the generated nodes are interpolated linearly from the current specified on the previous record and the present record. The interpolation is based on number of nodes generated, not on nodal coordinates. For example, if the current is to be generated for 1 node, its speed and angle are the averages of the previous values and the present values.

For a uniform current, the data are stored in .current_velocity(2). For a varying current, the data are stored in .current_velocity(2,#nodes). In either case, angles are converted to radians prior to storage.

For a variable current, the nodes must be defined prior to this command, and the command is terminated by a blank line.

See Also
pbridge pcurrentvelocity

dampers

Command Syntax

```
dampers  [-noecho]
n=node_no  c=cx,cy,cz,cxx,cyy,czz  [inc=]
```

Reads nodal dampers.

If -noecho is specified, the data will not be echoed to the output file.

cj and cjj are the nodal dampers with respect to the j axis. Generation occurs if two records do not have sequential nodes and inc, node number increment, is greater than 0. The generated nodal data will have the same value as the record with inc.

Dampers for repeated nodes are accumulated.

End input with a blank line.

The data are stored in the array .dampers_inp(7,*), in the order node #, 6 damping components.

The nodal dampers are assembled into the structure damping by the form_c command.

See Also

form_c pdamper

pdampers

Command Syntax

```
pdampers
```

Print input nodal dampers.

See also

dampers

direct_th

Command Syntax

```
direct_th  arg1  dt=?  endtime=?  [-linear]  [damping=?,?]  [begintime=]
           [save_disp=?]  [save_restart=?]  [beta=]  [gamma=]  [conv=?,?]
           [maxit=?]  [k_iter=?]  [-morison]  [-strict_conv]
           [conv_option=1]
```

Determine the direct time history response.

```
arg1         = the time function (nptsx2 matrix)
dt           = time step
endtime      = simulation time
-linear      = response is linear; no iteration (see below)
damping      = mass and stiffness coefficients for Rayleigh damping
begintime    = time to begin the simulation (default=0)
save_disp    = integer to specify that the displ. are saved every
               disp_save time steps (default=1)
save_restart= integer to specify that the displ., velocities and
               accelerations are saved every save_restart time steps
               (see below)
beta         = coefficient in Nemark method (default=1/4)
gamma        = coefficient in Nemark method (default=1/2)
conv         = convergence tolerances on displacement and forces
               (see below)
maxit        = maximum # of iterations within a time step (default=10)
k_iter       = # of iterations with constant stiffness (default=2)
-morison     = determine incoming wave velocities and accelerations
-strict_conv= requires force and displacement convergenece
conv_option = if specified, then "small" displacements are ignored
               for convergence
```

This command integrates the equations of motion directly, using Newmark's method. The default beta, gamma coefficients correspond to the unconditionally stable constant average acceleration method. The command assumes the stiffness matrix is in .kstr, in profile storage, and the mass matrix is in .mstr and is either diagonal or in profile storage. The damping matrix is in .cstr and can be either diagonal or in profile storage. If it does not exist, the values specified by damping are used to form a Rayleigh damping matrix. The first value for damping is the mass coefficient, and the second value is the stiffness coefficient. Provide non-zero initial displacements in u_init(neq), nonzero initial velocities in v_init(neq), and nonzero initial accelerations in a_init(neq). Otherwise, they are assumed to be zero.

The default is to begin the simulation at time = 0. In this case, any existing time history results created by previous calls of this command are deleted. A nonzero time can be specified with begintime. In this case, if there are existing displacements from previous calls of this command, then the new nodal displacements are appended to those results.

The displacements are saved every save_disp time steps. The displacements, velocities, and accelerations will be saved every save_restart time steps and can be used for a restart analysis. These are always saved at endtime.

The number of time steps is the nearest integer to (endtime-begintime)/dt. Hence, the simulation may be slightly longer or shorter than endtime.

The load patterns are defined by the nodef command. If the array load_comb exists, it is interpreted as load combination factors to be applied with the "load patterns" in .load_pat. Otherwise, the first load pattern is used. Element loads are included if they correspond to the load pattern used.

The time variation of the loads is defined by arg1, which is a 2D matrix defining f(t). The first column is time, and the second column is the function value. If the time function is not defined out to endtime, the function is continued at the last defined value out to endtime.

The displacements, that is the time history nodal response, is stored in .nodal_th_disp(neq,nsaved), where neq is the number of equations. The value for nsaved = (endtime-begintime)/(dt*save_disp)+1. The times are saved in .nodal_times(nsaved).

The displacements, velocities and accelerations are saved in .th_restart_disp(neq,*), .th_restart_vel(neq,*), and .th_restart_acc(neq,*), where the number of columns is the number of times at which the results are saved. For linear analysis, the results are only saved at endtime. For nonlinear analysis, the number of columns depends on the parameter save_restart. The default is to save these only at endtime. In any event, the last columns are the results at endtime. The times are saved in .th_restart_times(*). Because nodal displacements, velocities, and accelerations are saved, multiple direct_th commands can be used in a "restart" fashion.

If -linear is specified, no iteration is carried out and no check on equilibrium is made. It assumes strictly linear behavior. In this case, maxit, save_restart, conv, and k are ignored. Displacements, velocities, and accelerations are saved only at endtime. However, save_disp is used to save the displacements.

For a nonlinear analysis, the time stepping is done as follows. Within a time step, at the end of the first iteration the unbalance in the equation of motion is calculated. If the maximum absolute value in the unbalance is less than conv(2), then no iteration is carried out; the analysis proceeds to the next time step and any unbalance is applied in that time step. Otherwise, iteration is carried out based on the unbalance. Iteration continues until the displacement increment for each DOF in the current iteration is not greater than conv(1) * the previously calculated displacement increment in the current time step, up to maxit iterations. If -strict_conv is specified, then both displacement and force convergence must be satisfied. Sometimes, the solver can get stuck trying to satisfy the convergence criterion for "small" displacements, for example, those that should be nearly zero. Specifying conv_option will ignore the DOFs for which the displacement increment in the time step is less than 10^-4 times the maximum displacement increment in the time step. The stiffness is kept constant for k_iter iterations.

WARNING: If nonzero initial conditions are provided, including nonzero forces at start time, they should reflect dynamic equilibrium. The nonlinear solver may be able to adapt errors in the initial conditions, but the linear solver cannot. This warning is mostly applicable to user-defined intial conditions. Initial conditions based on a previous analysis should already satisfy dynamic equilibrium.

See Also

form_c form_k form_m initial_conditions modal_th nodef nsolve

disp_cntl

Command Syntax

```
disp_cntl
```

Read displacement control information. The data is input immediately following the command in the format:

```
Node  DOF  Factor
```

The command creates the vector .disp_cntl_vec, which is used to calculate the generalized displacement in a displacement control solution.

The command is terminated by a blank line.

See Also

nsolve

distr_deck_loads

Command Syntax

```
distr_deck_loads
element=? f=? start=? end=? [DG=?] [y=?]
```

Input distributed vertical deck loads for pontoon bridge element.

element is the element number on which the load acts
f is the distributed load (force/unit length)
start is the distance from the beginning of the element to the start of the load
end is the distance from the beginning of the element to the end of the load
DG is the vertical distance from the deck to the center of gravity of the load
y is the transverse offset from the center line of the element

End input with a blank line.

Only one distributed load may act on an element.

Distributed load data are stored in the array .pbridge_distrlds(6,*), in the order element #, f, start, end, DG, y.

This command, if used, must be given after the elements are defined with the pbridge command.

See Also

conc_deck_loads pbridge pdeck_loads

el_iso_matl

Command Syntax

```
el_iso_matl arg1 e=?  nu=? [-stress] [-strain] [-axisym]
  arg1 = 2D stress-strain constitutive matrix
  e  = modulus of elasticity
  -stress -> plane stress
  -strain -> plane strain
  -axisym -> axisymmetric case
```

Creates the stress-strain constitutive matrix for a linear elastic, isotropic material.

elem_alias

Command Syntax

```
elem_alias
  Print aliases for elements.
```

elem_grp

Command Syntax

```
elem_grp e=? code=?
  Adds or deletes element groups to be processed

         e = element group number
      code = integer code used by the element during processing
```

The group number is the same as the element "number" (1 for elem01, 2 for elem02, etc.). The code for group "i" is stored in location "i" of the vector .elem_grp. Groups with a nonzero code will be processed by element processing commands, such as form_k, state, and response. The array and codes are set up during element definition, but this command allows specific groups to be removed and/or added from processing.

See Also

elem_alias form_k form_m response state

eq_direction

Command Syntax

```
eq_direction arg1 [-x]  [-y]  [-z]
```

Define the seismic direction vector arg1 (often denoted "r" in structural dynamics). arg1 is a vector of 1's and 0's. If -x is specified, x degrees-of-freedom have a 1; other degrees-of-freedom have a 0. -y and -z are similar. Only one of the coordiante directions can be specified.

The equations must have been numbered before this command can be used. It expects the number of equations to be in .#eqs and the equation numbers to be in .node_eqs.

See Also

form_k   num_eqs

export_graphics

Export mesh to graphics program input file

Command Syntax

```
  export_graphics  [-target]  [-deformed  case=?  T=?  -autoscale scale=? &
                               steps=?]  [file=?]
```

Export the mesh to a graphic program's input text file.

The graphics program is specified by the argument -target. Gmsh (http://www.geuz.org/gmsh/) is the only option supported at this time. That is, the available option is -Gmsh, and it is automatically chosen.

The default is to plot the undeformed mesh.

If -deformed is specified, then the displaced shapes, one for each displacement set (column) in .disp will be created. If case is specified, only the column number specified by the integer case is processed. In that case, the displacements can be scaled if steps is omitted. If -autoscale is specified, then a scale factor will be determined automatically for the displacment set. The scale factor will be determined such that the displacement is scale*length, where length is the maximum length of a bounding box enclosing the model. The default is 2% (i.e., scale=2). If -autoscale is not specified, then scale specifies the scaling factor. The default is no scaling.

If -displ is specified, then the x,y,z displacements (u,v,w) in .disp are the plot variables.

T= is the name of a real vector of "times". The size of the vector is the same as the number of columns in .disp. This option can be used as follows. If mode shapes are being plotted, then T could contain the natural periods. If .disp contains a time sequence of displacements, then T could contain the time for each shape. If .disp contains static displacements, then T could contain the load case number. If a vector is not specified, one is created with the values 1,2, ...

The normal mode shapes from an eigenvalue analyis can also be displayed. Just copy the modes to .disp; i.e., run the following command prior to this command:

```
cp .phi .disp
```

The HYDRAN-XR GUI can display the mesh and static displaced shapes. Only one displaced shape per file is allowed, however (i.e., use case= option). It cannot display animations.

The displacements in .disp (including mode shapes) can be animated to more easily visualize the shape. Use case to specify the column number and provide a value for steps, the number of frames per cycle that will be generated. These can be viewed in an external program such as Gmsh.

If filename is specified, the results will be written to the file filename; othwerwise they will be written to the file project_name.msh (Gmsh).

fem_error

Command Syntax

`fem_error  [-SE]  [-user]`

Estimate the error in a finite element solution.

The total error (intialized to -1) is put in .fem_error. In general, the elements also calculate their relative error (element error/global error) in response to this command.

The parameter -SE specifies that the estimate of the error in strain energy is used (default).

The parameter -user specifies that the error is based on a user-defined error function (see the appropriate element for details.

Each element is responsible for consistency with the specification of error type. See the appropriate element help page.

form_G

Command Syntax

`form_G`

Command has not been implemented in this verison.

form_c

Command Syntax

`form_c  [-diag]`

Form global damping and put in vector .cstr. The default is to form the consistent damping matrix and store it in upper profile form. The command expects the location of the diagonals to be in the vector .kdiag_loc, which is formed when the equations are numbered by the command form_k or num_eqs. The element groups to be included in the damping assembly are specified by nonzero codes in the array .elem_grp, which is defined during element definition.

If the flag -diag is specified, a diagonal damping matrix is formed.

The scalar .cstr_type is created with a value of 0 for consistent damping and 1 for diagonal damping matrix.

If a diagonal damping matrix is requested and an element returns a consistent matrix, the element damping is "lumped" by scaling the diagonals of Ce, the element consistent damping, by the factor (u^T Ce u)/(sum of diagonals), where u is a vector of ones. Hence, the lumped damping and the consistent damping will have the same value for (u^T C u). This lumping procedure is not appropriate for all types of elements.

See Also

elem_grp form_m num_eqs

form_k

Command Syntax

```
form_k [-#eqs] [-stiff] [-loads]
```

Form global stiffness and loads. The element groups to be included in the stiffness assembly are specified by nonzero codes in the array .elem_grp, which is defined during element definition. The default operation is to number the equations, if necessary, form the stiffness, and form the loads. If any flag is specified, then the operation is determined by the flags specified, as follows:

```
-#eqs    -> number the equations
-stiff   -> form stiffness
-loads   -> form nodal loads
```

If -#eqs is not specified, the nodal equation numbers in .node_eqs are used. If .node_eqs does not exist, or if -#eqs is specified, the equations are numbered based on the restraint codes in .bcid, which is established by the bcid command. The nodal equation numbers are put in .node_eqs(#dofs_per_node,#nodes), and the number of equations is put in the scalar .#eqs. The locations of the diagonal elements of the stiffness are stored in .kdiag_loc and the size of the stiffness matrix is stored in .kstr_size; hence, the elements must have been defined.

If the stiffness is formed, the upper profile is stored in the vector .kstr. A scalar .k_status is also created and zeroed. This value is modified when the stiffness is factored.

If the loads are formed, the load array .load_pat(#eqs,#patterns), where #patterns is the number of load patterns defined in .load_inp (see command nodef). If this array does not exist, then the zero load pattern array .load_pat(#eqs,1) is created. Equivalent nodal loads from element loads are assembled into .load_pat.

See Also

bcid elem_grp load_summary nodef num_eqs peqs

form_m

Command Syntax

```
form_m  [-diag]
```

Form global mass and put in vector .mstr. The default is to form the consistent mass matrix and store it in upper profile form. The command expects the location of the diagonals to be in the vector .kdiag_loc, which is formed when the equations are numbered by the command form_k or num_eqs. The element groups to be included in the mass assembly are specified by nonzero codes in the array .elem_grp, which is defined during element definition.

If the flag -diag is specified, a diagonal mass matrix is formed.

The scalar .mstr_type is created with a value of 0 for consistent mass and 1 for diagonal mass.

Nodal masses input with the mass command are assembled into the global mass matrix, also.

If a diagonal mass matrix is requested and an element returns a consistent matrix, the element mass is "lumped" by scaling the diagonals of Me, the element consistent mass, by the factor (u^T Me u)/(sum of diagonals), where u is a vector of ones. Hence, the lumped mass and the consistent mass will have the same value for (u^T M u). This lumping procedure is not appropriate for all types of elements.

See Also

elem_grp form_k mass mass_summary num_eqs

form_lagrangeG

Command Syntax

form_lagrangeG

Form element G matrices for Lagrange constraint in smoothing analysis. This command requires the global equation numbers. The commands num_eqs (equation numbering) and/or node_order may be issued prior to this command. If the equations have not been numbered, this command will do it.

See Also

node_order num_eqs smth2q

imposed_displ

Command Syntax

```
imposed_displ
n=node_no  dof=dof  disp=p1,p2,...
```

Reads imposed nodal displacements.

n is the node number
dof indicates the nodal degree of freedom, that is, 1 for x translational displacements, 2 for y, ..., and 6 for z rotational displacements
pi is the displacement corresponding to load pattern "i"

The last values input for a node and dof are used. Degrees of freedom for which displacements are not specified are assumed to be "free", that is, displacements are not imposed. However, although different displacements can be imposed for different load patterns, a degree of freedom with specified displacements in one or more patterns are assumed to be constrained in all patterns. If the displacement value is not specified for a load pattern, it is assumed to be zero.

If the optional identifiers (those in [], e.g., n=) are used in an input record, all data in that record must have the correct identifier.

End input with a blank line.

The data are stored in the array .disp_inp(2+#patterns,*). Each column contains the node #, DOF # (1-6), displacements in each load pattern. As many columns as necessary are used.

The nodef command, which defines the number of load patterns, must precede this command.

Caution: Displacements may not be imposed with all solution commands, and some commands may interpret these values differently. Check the help for the particular solution command.

See also

nodef pimposed_displ

initial_conditions

Command Syntax

```
initial_conditions  arg  [-noecho]
n=node_no   u=x,y,z,xx,yy,zz  [inc=]
```

Reads initial values for nodal displacements, velocities or accelerations.

If -noecho is specified, the data will not be echoed to the output file.

arg is the name of the neq vector to save the initial conditions.
u are the six values for node_no. Generation occurs if two records do not have sequential nodes and inc, node number increment, is greater than 0.

The last values specified for a node is used.

End input with a blank line.

The data are stored in the vector arg(neq). The equations must have been numbered prior to this command; see the form_k command.

This command is meant to specify nonzero initial displacements and/or velocities and/or accelerations for a time history analysis. To specify multiple quantities, repeat the command with different names for arg.

Note: initial conditions should be consistent with the equation of motion.

See Also

direct_th form_k modal_th

load_summary

Command Syntax

```
load_summary
```

The sum of the loads in .load_pat in each coordinate direction is reported. The sums of the nodal moments are also printed. The summary only considers specified nodal loads and equivalent nodal loads from the elements. The effect of imposed displacements is not included.

See Also

form_k

lsolve

Command Syntax

```
lsolve [k=?] [-sparse]
```

```
Linear solution
  k = 0 -> form stiffness by calling routine form_k
    = 1 -> use existing, unfactored stiffness
    = 2 -> use existing, factored stiffness
```

For an existing stiffness, the profile stiffness is expected in array .kstr and the locations of the diagonals are assumed to be in array .kdiag_loc. Loads are assumed to be in .load_pat. If the array load_comb exists, it is interpreted as load combination factors to be applied with the "load patterns" in .load_pat. The solution is placed in .disp.

If -sparse is specified, a sparse equation solver is used. If k > 0, the arrays as defined in the help for command ptosparse are expected. The data for the factored sparse stiffness are in arrays .sparse_mmik, .sparse_mmim, .sparse_mmfn, and .sparse_addr. These arrays can be deleted when the factored stiffness is no longer needed.

Note 1: The sparse option is recommended for all large problems because it is much more efficient.

Note 2: To include geometric stiffness in a linear analysis, precede this command with command form_k and then use k=1 or 2 here.

The profile solver modifies .kstr. Although the sparse solver puts the factored matrix in a copy, it does reorder the original arrays.

See Also

form_k nodef ptosparse

mass

Command Syntax

```
mass  [-noecho]
n=node_no  m=mx,my,mz,Ixx,Iyy,Izz  [inc=]
```

Reads nodal masses.

If -noecho is specified, the data will not be echoed to the output file.

mj and Ijj are the nodal mass/inertia with respect to the j axis. Generation occurs if two records do not have sequential nodes and inc, node number increment, is greater than 0. The generated nodal data will have the same value as the record with inc.

Masses for repeated nodes are accumulated.

End input with a blank line.

The data are stored in the array .mass_inp(7,*), in the order node #, 6 mass components.

The nodal masses are assembled into the structure mass by the form_m command.

See Also

form_m pmass

mass_summary

Command Syntax

```
mass_summary  [shift=?,?,?]  [-save]
```

A 6x6 "rigid body" mass matrix M is determined as follows. Six "rigid body modes" are formed in the matrix u(#dof,6), and then the 6x6 mass matrix is formed as u^T * .mstr * u. The default is to form the modes with respect to the origin, but if shift=x,y,z is specified, they are formed with respect to the point (x,y,z).

For an unconstrained structure, the modes are true rigid body modes. If degrees-of-freedom have been restrained to be zero, any mass associated with those degrees of freedom are not included.

If -save is specified, the 6x6 mass matrix is stored in .mass_summary6x6 and the estimates of the center of mass (see below) are stored in the 3x3 matrix .mass_summaryCG.

Note: The form_m command must precede this command. Additionally, the command assumes the mass is in either diagonal or profile form.

The masses in the different directions, i.e. x-mass, y-mass, and z-mass - M(1,1), M(2,2) and M(3,3) - may not be the same, e.g., because of different nodal boundary conditions. Three different estimates of the center of mass (xbar, ybar, zbar) relative to the shift point are calculated based on the 3 different masses:

x-mass: sum of (nodal x-masses times x, y, z coordinates) divided by x-mass

y-mass: sum of (nodal y-masses times x, y, z coordinates) divided by y-mass

z-mass: sum of (nodal z-masses times x, y, z coordinates) divided by z-mass

Because calculations involving the center of mass can be sensitive to the number of significant figures, these values are reported to high precision to aid the user. Additionally, the option to save the results in the database for further use is provided. Arrays can be printed to higher precision than the default using the options on the print command or the command xprint.

Note: This command operates on the mass matrix, .mstr. Calculation of the center of gravity will be affected by any masses associated with constrained nodes. These masses will have already been transformed to the master node, and will be associated with the coordinates for that node.

See Also

form_m rigid_modes

merge_nodes

Command Syntax

```
merge_nodes  [tol=?] [xtol=?] [ytol=?] [ztol=?] [-noprint]
```

Merges the equation numbers of coincident nodes. Specifically, if the x, y, and z coordinates of node j are within tolerance tol (default = 10^-8) of the coordinates of node i, i < j, then the equations for node j will be the same as for node i. This is carried out via the array .bcid. If node numbering arrays exist, they will be removed. Different tolerances for the coordinates may be specified via xtol, ytol, and ztol. If -noprint is specified, the report of merged nodes will be suppressed.

See also
  bcid

modal_th

```
Command Syntax
  modal_th  arg1  dt=?  endtime=?  [xi=]  [begintime=]  [save_velocities=?]
    Determine the modal time history response.
      arg1 is the time function (nptsx2 array)
      dt        = time step
      endtime   = simutlation time
      xi        = constant modal damping (xi=.02 means 2%)
      begintime = time to begin the simulation (default=0)
      save_velocities is an integer to specify the time step increment to
      save the modal velocities
```

This command integrates the modal equations exactly, assuming that the time variation of the force varies linearly between the points in arg1. The command assumes the natural frequencies are in .omega(nmodes) and the normal modes are in .phi(neq,nmodes). It also assumes the modes have been mass-normalized. If xi is missing or zero, and an array xi(nmodes) exists, those values are used for the damping for each mode. For non-zero initial conditions, the initial displacements should be given in u_init(nmodes) and the initial velocities should be given in v_init(nmodes). If either or both of these arrays don't exist, they are assumed to be zero.

The default is to begin the simulation at time = 0. In this case, any existing time history results created by previous calls of this command are deleted. A nonzero time can be specified with begintime. In this case, if there are existing modal displacements from previous calls of this command, then the new modal displacements are appended to those results.

The modal velocities will be saved every save_velocities time steps. The modal velocities at endtime are always saved.

The load patterns are defined by the nodef command. If the array load_comb exists, it is interpreted as load combination factors to be applied with the "load patterns" in .load_pat. Otherwise, the first load pattern is used. Element loads are included if they correspond to the load pattern used.

The modal "coordinates", that is the time history modal response, are stored in .modal_disp(nmodes,nsteps), where nmodes is the number of frequencies and modes in .omega and .phi. The minimum value for nsteps = (endtime-begintime)/dt+1. However, the solution is also determined at each time for which the time function is defined (up to endtime). If the time function is not defined out to endtime, the function is continued at the last defined value out to endtime. The times are saved in .modal_times(nsteps). Note that regardless of dt, the solution is calculated at each time for which the time function has been defined. At each of these points, the next time is obtained by adding dt.

The modal velocities are saved in .modal_vel(nmodes,*), where the number of columns depends on the parameter save_velocities. The last column is the velocities at endtime. The times corresponding to saved velocities are saved in .modal_vel_times(*). Because both the modal displacements and velocities are saved, multiple modal_th commands can be used in a "restart" fashion.

See Also

initial_conditions lsolve nodef

nodal_constraint

Command Syntax

```
  nodal_constraint  maxnodes=?
  type=general c=n cdof=cdof r=n1,n2,... rdof=r1,r2,... factor=f1,f2,...
  type=body c=n r=n1
  type=body_trans c=n r=n1
```

Input linear kinematic constraints on the nodal degrees-of-freedom. There are three possibe constraint types: general, body, and body_trans.

maxnodes is the maximum number of nodes that appears in any single constraint equation (default=4). For example, an equation for a body constraint involves 4 nodes, in which case maxnodes should be no less than 4.

type is the type of the constraint

Type general:
  c is the node number that has a constrained DOF
  cdof is the node-local, constrained degree-of-freedom (1-6)
  r is the node numbers, nj, with independent DOFs
  rdof is the degree of freedom, rj, for node nj
  factor is the numerical factor, fj, in the constraint equation

Type body (rigid body constraint),
  c is the node number that is constrained
  r is the node number with the independent DOFs ("master" node)

Type body_trans (rigid body but only translational DOFs are constrained):
  c is the node number that is constrained
  r is the node number with the independent DOFs ("master" node)

End input with a blank line.

The constraint information is stored columnwise for each constraint in

```
  .const_nodes(maxnodes,#constraints) = n,n1,n2,...
  .const_dofs(maxnodes,#constraints) = cdof,r1,r2,...
  .const_factor(maxnodes,#constraints) = 1,f1,f2,...
```

A constrained degree-of-freedom depends on the independent degrees-of-freedom via a constraint equation. For the general constraint, the equation is

```
    constrained dof = f1*n1(r1) + f2*n2(r2) + ...
```

in which ni(rj) represents the rj_th displacement of node ni. These displacements must be independent; i.e., they cannot be constrained by another constraint equation. For body constraints, the constraint equations are similar, but they express the rigid body constraint. The factors are calculated automatically based on the nodal coordinates.

For type=general, these values are specified directly. For type=body, the factors for each degree-of-freedom are calculated based on a rigid body constraint. Hence, each body constraint actually corresponds to 6 constraint equations. For type=body_trans, each constraint corresponds to 3 constraint equations.

The constraint information is stored columnwise for each constraint equation in

```
.const_nodes(maxnodes,#constraint_eqs) = n,n1,n2,...
.const_dofs(maxnodes,#constraint_eqs) = cdof,r1,r2,...
.const_factor(maxnodes,#constraint_eqs) = 1,f1,f2,...
```

This is an optional command. The command bcid must be processed prior to this command. The command modifies the array .bcid(7,#nodes), created by the bcid command, such that the value 2 is inserted in .bcid corresponding to constrained DOFs. If a DOF has been previously constrained by the bcid command, to be either 0 or identical to another DOF, the constraint specified by this command is ignored. This command, if used, must be processed prior to num_eqs and form_k.

See also

bcid form_k num_eqs pdisp

nodal_disp

Command Syntax

```
nodal_disp  arg  [nodes=?,?]  [-node#]
```

Arrange the nodal displacements in the first column of .disp into a rectangular array arg(#active_nodes,6), where the displacements for each active node are in a row. A range of node numbers can be specified by nodes=. The first value is the first node number in the range, and the second value is the last number in the range. If the flag -node# is specified, then arg will have 7 columns, the first of which is the node number.

Note: Only active nodes are processed.

See Also

pdisp pndisp pndisp_th

nodal_pressure

Command Syntax

```
nodal_pressure
n=first,last,incr  pat=pat  [p=pressure]  [-hydro  density=density]   &
  [z=Z0]  [-no_negative]
```

Reads nodal pressures.

first is the first node number in the sequence
last is the last node number in the sequence (blank for a single node)
incr is the node number increment in the sequence (default = 1)
pat is the load pattern to which the pressure is assigned
pressure is the pressure for the node (constant for the node sequence)

If -hydro is specified, hydrostatic pressure is generated as follows. The pressure is calculated as -density*(z - Z0), where z is the z-coordinate of the node. Z0 is the z-coordinate at which the pressure is zero. If -no_negative is specified, then calculated negative pressures are set to zero. (This option is active only if hydrostatic pressures are calculated.)

If n=all is specified instead of a node range, then it is applied to all nodes.

End input with a blank line.

The data are stored in the array .nodal_pressure(nodes,*), in which nodes is the largest active node number and the number of columns is the number of load patterns with pressures assigned. The integer vector .nodal_pressure_pat(*) is created as well; the ith element holds the pattern number for the ith column of .nodal_pressure. Note: the largest allowable load pattern is defined by the nodef command. If a larger pattern is specified by this command, the program will generate an error when the loads are formed.

See Also

nodef

nodef

Command Syntax

```
nodef  [#=?]
n=node_no  p=pat  f=?,?,?...
```

Reads nodal load patterns.

# = number DOFs per node (default = 6)
pat is the pattern to which the load is assigned
f specifies the loads in a comma-separated list of # values

Loads for repeated nodes are accumulated.

End input with a blank line.

The data are stored in the array .load_inp(#dofs_per_node+2,*), in the order node #, pattern, load components.

The maximum load pattern number specified in this command becomes the maximum load pattern number. If no nodal loads exist, but all loads come from element loads, the maximum load pattern number is still defined by this command, in which case a zero load should be specified. (If the number of load patterns equals 1, this is unnecessary.)

See Also

pnodef

nodes

Command Syntax

```
nodes  #=?
n=node_no  x=x-coor  y=y-coor  z=z-coor  [lgen=lgen]
```

Reads and generates nodal coordinates. The value specified by # is used to define storage requirements, and it must be greater than or equal to the maximum node number. If this value is missing or 0, it is assumed that existing nodes are being changed or added to, and the previous value applies. lgen is the node number increment for linear generation. Nodes are generated equally spaced along a straight line if two adjacent records do not have sequential node numbers and if lgen on the second line is not zero or blank. For other generation options, see the command node_gen. Nodes need not be input in sequence.

End input with a blank line.

The coordinates are stored in array .xyz(3,#), and the maximum possible node number (specified by #) is stored in .#nodes_tot.

Arrays associated with the equation numbers are deleted if # is specified; i.e., .bcid, .node_eqs, .#eqs, .node_eq_order, .node_active are deleted.

Active nodes are those that are defined explicitly either by this command or another command that creates nodes. The node number of the maximum defined node is stored in .#nodes. The character vector .node_active has an "A" for active nodes. Only active nodes can be used.

The nodes command need not be executed as long as the coordinates, which could be generated by another program, are put in the array .xyz, .#nodes and .#nodes_tot are set, .node_active is created, and any other associated arrays are deleted.

See also

node_gen pnodes bcid

node_gen

Command Syntax

```
node_gen
linear=node1,node2  [inc=?]  [w=?]
quad=node1,node2,node3,node4  [inc=inc1,inc2]  [w=w1,w2]
```

Generates nodal coordinates. The nodes command must preceed this command.

Linear generation is specified by the identifier linear. Nodes are generated from node1 to node2 with a node increment of inc (default=1). The spacing of nodes is equidistant unless the positive weight w (default=1) is specified, in which case the spacing between node n and n+1 is equal to w times the spacing between n-1 and n.

The identifier quad specifies node generation within the "quadrilateral" defined by the four nodes (see sketch below). Linear generation is done between node1 and node2, and between node3 and node4, based on inc1 and w1. Linear generation is also done between node1 and node3, and between node2 and node4, based on inc2 and w2. Then, linear generation is done between the nodes generated from 1-3 and 2-4, based on inc1 and w1. The number of interior lines generated is the same as the number of nodes generated from 1-3. If the number of nodes generated from 1-3 is larger than the number generated from 2-4, the "extra" lines will not be generated, as this would result in a redefinition of nodal coordinates along 3-4. If node3 and node4 are identical, the generated nodes are within a triangular domain. The nodes need not be coplanar.

End input with a blank line.

```
node3                                    node4
  X----------O----------O----------O----------X
```

```
                |                                           |
                |                                           |
                |                                           |
                |                                           |
          node1 |       node1+inc2                          |
          +inc2 O          O +inc1    O           O         O
         ^      |                                           |
         |      |                                           |
       generate |                                           |
          w/    |                                           |
        inc2    |                                           |
                |                                           |
                |          generate w/ inc1 ->              |
                X----------O----------O----------O----------X
              node1    node1+inc1                         node2

        X = specified node
        O = generated node
```

See also
  nodes

node_order

Command Syntax

```
node_order [-cm]  [-rcm]
```

Internally orders nodes based on Cuthill-McKee (-cm) or Reverse Cuthill-McKee (-rcm) algorithms for equation numbering. RCM is the default option. The node numbers are stored in .node_eq_order(.#nodes) in the order in which the equations will be numbered.

This command is recommended for large problems because it can reduce the size of the stiffness and mass matrices, and hence the efificiency of the solution. If this command is used, it must come after all elements and nodal constraints have been defined. Furthermore, it should be either immediately before a num_eqs command, if that command is used, or otherwise immediately before a form_k command.

Warning: This command can only be used if all active nodes with independent degrees of freedom are connected to each other through elements. That is, there must be a path between any two nodes that can be traversed, much as a path exists between any two leaves in a tree. Two separate, unconnected structures (two separate trees) do not satisfy this requirement. If the model includes two unconnected structures, connect them using a "dummy" element, such as a truss element with zero stiffness and zero mass. Otherwise, the command will fail.

node_str

Calculate average nodal stresses.

Command Syntax

```
node_str  arg1  arg2  arg3  [n=?]  [-x]  [-y]  [-z]  [tol=?]
[node=node]  [inc=inc  gen=gen]
```

n is the number of nodes specified in this command
-x, -y, -z specify which coordinates are given (see below)

tol is the tolerance on individual coordinates to determine to which node a particular point corresponds (default=1.e-8).

node is the node number
inc is the node increment used for generation
gen is the number of nodes to generate.

End input with a blank line.

This command determines the arithmetic average of stresses at the nodes specified. If the number of nodes on the command line is left blank, all nodes are used. If it is given, as many generation commands as necessary may be used to define the node numbers.

Nodal coordinates are expected in array .xyz. Coordinates and stresses are in array arg1 in the form coord1, ..., stresses. Which coordinates appear in this array are specified by the flags -x, -y, and -z. For example, if -x and -z are given, then the first column in arg1 contains x-coordinates, the second column contains z-coordinates, and the remaining columns are the stresses at the corresponding (x,z) coordinates. Coordinates that are not specified (y in this example) are assumed to be zero.

The node numbers for which nodal stress averages are calculated are kept in array arg3(n).

The average stresses are kept in arg2(n,*), where the number of columns is the same as in arg1. The rows of arg2 may be less than n, because nodes for which no stress values are found are eliminated from arg2 and arg3.

nsolve

Command Syntax

```
nsolve steps=? tol=?,? maxit=? [k=?] [control=?] [dlim=?,?] [gdisp=?]
       [dprin=?] [fprin=?] [state=?]
```

```
  Nonlinear solution command
    steps   = number of load steps
    tol     = load and displacement tolerances
    maxit   = maximum # of iterations within a load step
    k       = # of iterations with constant stiffness (K)
            < 0 -> form tangent stiffness and use for abs(k) iter.
            = 0 -> form and use initial stiffness iter.
            > 0 -> use existing stiffness for k iter., then reform
    control = 0 -> load control
            = 1 -> displacement control
    dlim    = max. transl. and rotational displ. increment in iteration
              for load control
    gdisp   = magnitude of the generalized displacement for displ.
              control; see command disp_cntl
    state   = 1 -> do state determination in first step
    -morison= determine incoming current velocities for drag

    dprin   = print flag for nodal displacements
            = 0 -> not printed from nsolve (default)
            = 1 -> print at each load step
            = 2 -> print at each iteration
    fprin   = print flag for element forces
            = 0 -> not printed from nsolve (default)
            = 1 -> print at each load step
            = 2 -> print at each iteration
```

A form_k command must precede this command.

The increment in external loads is determined from the .load_pat array created by the command nodef. If the array load_comb exists, it is assumed that it is a vector of load combination factors, and the load patterns are combined to form the load increment. If load_comb does not exist, the first load pattern is taken as the load increment. The displacements are placed in .disp, and a history of displacements are kept in .disp_history.

For additional information on the displacement control strategy, see Powell, G.H. and Simons, J., "Improved Iteration Strategy for Nonlinear Structures," IJNME, 17:1455-1467 (1981).

num_eqs

Command Syntax

```
num_eqs  [-#eqs]
```

Number nodal equations and determine stiffness memory requirements. The element groups to be included are specified by nonzero codes in the array .elem_grp, which is defined during element definition. The default operation is to number the equations if the array .node_eqs does not exist. If the flag -#eqs is specified, the equations are numbered regardless.

The equations are numbered based on the restraint codes in .bcid, which is established by the bcid command and modified by the nodal_constraint command. The nodal equation numbers are put in .node_eqs(#dof_per_node,#nodes), and the number of equations is put in the scalar .#eqs. The locations of the stiffness diagonals are determined based on the element connectivity and stored in .kdiag_loc; hence, the elements must have been defined. The size of the stiffness matrix is stored in the scalar .kstr_size.

This command is not normally used. The command form_k will number the equations by default, and that is the preferred approach. The command is provided in case a form_k command is not used, such as when only the mass matrix is formed. If used, it must follow the definition of all nodes and nodal boundary conditions, elements, and nodal constraints.

See Also

bcid elem_grp form_k nodal_constraints peqs

pbcid

Command Syntax

```
pbcid
```

Print nodal displacement restraints.

See also

bcid

pbody_frc2d

Command Syntax

```
pbody_frc2d
```

Print the body force coefficients defined in .body_f2d.

See Also

body_frc2d

pcurrentvelocity

Command Syntax

```
pcurrentvelocity
```

Print current velocity

See Also

current_velocity

pdeck_loads

Command Syntax

```
pdeck_loads
```

Print deck loads for pontoon bridge.

See also

conc_deck_loads distr_deck_loads pbridge

pdisp

Command Syntax

```
pdisp  [nodes=?,?]  [-screen] [form=format] [-file] [-append]
```

Print nodal displacements in array .disp for all active nodes with a print code of 1. A range of node numbers can be specified by nodes=. The first value is the first node number in the range, and the second value is the last number in the range. The default is to print the displacements for all active nodes. The default is to print to the output file only; if -screen is present, the displacements will also be printed to the screen. The default format is (i5,2x,1p6e12.3). Alternative formats can be specified by the parameter form=. In this case, format must be a valid FORTRAN format, enclosed in () with a maximum length of 160 characters and without any blank spaces.

If -file is specified, the displacements will also be written to the unformatted file project_name.dis. If -append is present, this file will be appended.

See Also

bcid pndisp pndisp_th

peqns

Command Syntax

```
peqns
```

Print equation numbers.

See also

bcid form_k num_eqs

pimposed_displ

Command Syntax

`pimposed_displ`

Print imposed displacements.

See also

imposed_displ

pmass

Command Syntax

`pmass`

Print input nodal masses.

See also

mass

pndisp

Command Syntax

`pndisp n=?`

Print the "history," in terms of load increments and load steps, of displacements for node n. The displacements are stored in .disp_history, which is created by command nsolve.

See Also

nsolve pdisp pndisp_th

pndisp_th

Command Syntax

`pndisp_th  n=?  arg1  arg2`

Print the time history of displacements, velocities, or accelerations for node n as determined by the direct_th command. arg1 is the array of displacements (or velocities, or accelerations) and arg2 is the vector of corresponding times. For example, to print the time history of displacements for node 10 at the saved time steps the command would be

```
pndisp_th n=10 .nodal_th_disp .nodal_times
```

and to print the velocites at the restart times it would be

```
pndisp_th n=10 .th_restart_vel .th_restart_times
```

See Also

direct_th pdisp pndisp

pnodef

Command Syntax

`pnodef`

Print input nodal loads.

```
    See also
      nodef
```

pnodes

```
    Command Syntax
      pnodes  [nodes=?,?]  [-screen]
        Print nodal coordinates of active nodes. A range of node numbers can be
        specified by nodes=. The first value is the first node number in the
        range, and the second value is the last number in the range. The
        default is to print the coordinates for all active nodes. The default
        is to print to output file only; if -screen is present, then output is
        to the screen as well.

    See also
      nodes
```

presponse

```
    Command Syntax
      presponse (or presp)  [-file] [-append]
        Print element response to output file.

        If -file is specified, the response will also be written to an
        unformatted file. If -append is present, this file will be appended.
        Not all elements support this option. Check the help for individual
        elements.

    See Also
      elem_grp  response  state
```

pstate

```
    Command Syntax
      pstate
        Print element state determined from state command.

    See Also
      elem_grp  state  response  presponse
```

response

```
    Command Syntax
      response
        Calculate element response.
          The element groups for which the response is determined are specified
          by nonzero codes in the array .elem_grp, which is defined during
          element definition, and can be modified by the command elem_grp.

    See Also
      elem_grp  state  presponse
```

rigid_modes

```
    Command Syntax
      rigid_modes arg [cg=?,?,?] [rigid=?,?,?,?,?,?] [node_range=first,last]
```

Generate rigid body modes relative to the "center of gravity" specified by cg=x-coor,y-coor,z-coord. The default is the origin. The modes are defined relative to axes that are parallel to the global coordinate axes. The default is to define six modes (surge, sway, heave, roll, pitch, yaw, in naval architecture parlance), but this can be controlled by the optional parameter rigid:

```
rigid    = six values, corresponding to 6 rigid body modes
             0 -> do not form corresponding rigid body mode
             1 -> form corresponding rigid body mode
```

A subset of nodes can be defined by the node_range option. Nodes not in this range will have zero displacement.

The modes are placed in the array specified by arg. The dimensions of this array are #dofs by #modes (default = 6).

This command expects the nodal coordinates in .xyz and the equation numbers in .node_eqs.

WARNING: This command has unreliable results when constrained nodes are involved and its use in that situation is not recommended. The problem is that the two constraints may not be compatible.

See Also

nodes num_eqs

state

Command Syntax

```
state
```

Calculate element state.

The element groups for which state determination is carried out are specified by nonzero codes in the array .elem_grp, which is defined during element definition, and can be modified by the command elem_grp.

See Also

elem_grp response   pstate

water_waves

Command Syntax

```
water_waves  #=#waves  [h=?]  grav=?  [-current #pts=?  current_beta=?]
n=?  ampl=?  period=?  phase=?  beta=?
```

End wave data with a blank line. If current exists, then enter current data, #pts records:

```
n=?  z=?   v=?
```

Reads wave and current data.

#waves is the total number of wave components.
h is the water depth. h=0 means deep water (default).
grav is the acceleration of gravity.

For each wave component:
    n ranges from 1 to #waves
    ampl is the component amplitude

```
        period is the component period (s)
        phase is the component phase angle (rad)
        beta is the component wave angle (degrees)
    If current, for #pts records:
        n is the record number (between 1 and #pts)
        z is the z-coordinate
        v is the current velocity

    End input with a blank line.

  It is assumed that the origin of the global coordinate axes is on the
  free surface and that the global Z axis is directed upwards.

  If there is a current, use -current. #pts must be at least 2. The first
  value must be for z=0 and the last value (negative) must be below the
  last node. current_beta is the current angle in degrees.

  If there is a current but no wave, enter a wave with zero amplitude

  Note: Only the first wave input is used at the present time (i.e., only a
  single regular wave is used).

  The following arrays are created:
    .water_wavesg(2)       -> h, grav
    .water_waves(#waves,5) -> ampl, period, phase, beta (rad), wave number
    .water_current_beta    -> current angle (rad)
    .water_current(#pts,2) -> z, velocity v

See Also
  direct_th
```

2.7. Finite Element Library

beam3d

Linear, 3-D beam element

There are two Command Syntax options.

```
          ---------- OPTION 1 ----------
Command Syntax (option 1)
  beam3d m=? n=? [-kg]
  m=matl e=emodulus  g=gmodulus  a=area  j=jsec iy=iyy  iz=izz          &
      [asy=asy  asz=asz]  [mbar=density mxyz=mx,my,mz]                  &
      [mI=mIxx,mIyy,mIzz] [cbar=cdense cxyz=cx,cy,cz] (1 record/matl)   &
  n=nel  mat=mat  nodes=node1,node2  node3=node3  [print=print]         &
      [gen=gen  inc=inc  inc2=inc2]                                     &
      [tension=tension tension_last=tension_last]
```

m is the number of different materials
n is the number of elements
-kg is a flag to include geometric stiffness

matl is the material number
emodulus is the modulus of elasticity
gmodulus is the shear modulus
density or mx,my,mz is the mass/unit length
area is the cross sectional area
jsec is the torsional constant
iyy,izz are area moments of inertia in local coordinates
asy, asz are the shear areas in y and z, respectively
    (0 -> the corresponding shear deformation is ignored)
mIxx, mIyy, mIzz are mass moments of inertia (per unit length)
    in local coordinates
cdense or cx,cy,cz is the damping/unit length

nel is the element number (ID#)
node1 and node2 are the node numbers
mat is the material number for the element
print .ne. 0, element results not printed
inc is the node 1 increment used for generation
inc2 is the node 2 increment used for generation (default=inc)
gen is the number of elements to generate
node3 lies in the local x-z plane
tension is the inital tension (for geometric stiffness only)
tension_last is the initial tension for the last element in a series

For generated elements, the initial tensions for the geometric stiffness are interpolated linearly, from tension in the first element to tension_last in the last element. If tension_last is not specified, it is set equal to tension.

There are two options to specify mass density. The usual option is to specify density with the mbar= identifier. In some special cases, it may be useful to specify a different mass for different directions of the local axes. In that case, use the identifier mxyz= to specify the three values separately. If both mbar= and mxyz= are specified, mxyz= will be ignored if density is a nonngegative value. The element creates a lumped, diagonal mass matrix in local coordinates. However, if the mxyz are not equal, and/or the mI are not equal, then the mass matrix when transformed to global coordinates may not be diagonal.

There are two options to specify damping, which is analogous to specifying mass and the formulation is similar. The usual option is to specify cdense with the cbar= identifier. It may be useful to specify a different damping for different directions of the local axes. In that case, use the identifier cxyz= to specify the three values separately. If both cbar= and cxyz= are specified, cxyz= will be ignored if cdense is a nonngegative value. The element creates a lumped, diagonal damping matrix in local coordinates. However, if the cxyz are not equal, then the damping matrix when transformed to global coordinates may not be diagonal.

End input with a blank line.

---------- OPTION 2 ----------

Command Syntax (option 2)

```
beam3d  -cylinder  mat=mat  node3=node3  [print=print]                 &
        p1=x1,y1,z1  p2=x2,y2,z2  R=R1,R2  CxL=Cseg,Lseg  [face=face]   &
        tension=tension_first  tension_last=tension_last]
```

Option 2 generates a cylindrical mesh interface elements around a 'spine' of beam elements.

p1 are the center coordinates of the cylinder start
p2 are the center coordinates of the cylinder end
R1,R2 are the radii at the start and end, respectively
Cseg are the number of interface elements around the circumference
Lseg are the number of interface elements along the length

All stiffness and mass properties are modeled by the 'spine' of beam elements along the center of the cylinder. The nodes to the interface elements are constrained via rigid body constraints to the nodes of the spine. For this option, the material must have been previously defined by an option 1 command, even if no elements were specified.

Interface nodes and elements are numbered around the circumference and then down the length. The interface elements are defined such that the local x-axis is down the length of the cylinder and the positive z-face is on the inside of the cylinder. Nodes are numbered clockwise looking from the outside, i.e., looking at the -1 face. See the interface element for details.

There is one cylinder per command line. To generate multiple cylinders, use multiple commands.

---------- ALL OPTIONS -------

The local (principal) axes of the beam are defined as follows:
The local x-axis is directed from node1 to node2
The local y-axis = (x-axis) X (vector from node1 to node3)
The local z-axis = (x-axis) X (y-axis)

If node3 is -1, -2, or -3, then the "vector to node3" is a unit vector in the direction of the negative X, Y, or Z global axes, respectively.

On input, created arrays are:

```
.beam3d_mp(m,19) -> emodulus, gmodulus, unused, area, jsect,
                    iyy, izz, asy, asz, mIxx, mIyy, mIzz, mx, my, mz
```

```
                       cdense, cx, cy, cz
    .beam3d_el(6,n)  -> node1, node2, material, print code, node3, ID#
    .beam3d_len(n)   -> element length
    .beam3d_st(12,n) -> Axial Force, Vy, Vz, Torque, My, Mz at node1
                        Axial Force, Vy, Vz, Torque, My, Mz at node2
    .beam3d_kg       -> 0 or 1; w/o or w/ geometric stiffness
```

This element calculates a lumped mass matrix in local coordinates.

For state calculation, element forces in local coordinates are put in .beam3d_st. Positive forces follow the right hand rule, not "beam" sign convention.

For response calculation, element does nothing.

For state output, results in .beam3d_st are printed, but using beam sign convention for shear and moment; torque at nodej is positive in the local x-axis.

For response output, no results are printed.

The element can be used with nonlinear elements, but the element response will be linear.

See Also

interface pbeam3d pstate presponse

pbeam3d

Command Syntax

```
  pbeam3d
    Print beam3d element data
```

See Also

beam3d

biot1d234

1-D element for consolidation of a linear, elastic medium

Implementation assumes element is directed along the positive X-axis.

Command Syntax

```
  biot1d234 m=? n=? [disp=?] [pressure=?]
  m=matl  e=emodulus  a=area  kx=kx  gammaw=gw  gammas=gs  (1/matl)
  n=nel  nodes=node1,node2  mat=mat [print=print]  [inc=inc gen=gen]

    m is the number of different materials
    n is the number of elements

    matl is the material number
    disp is linear (default), quad, or cubic (variation of displacement)
    pressure is linear (default), quad, or cubic (variation of pressure)
    emodulus is the uniaxial strain modulus of elasticity
    area is the cross sectional area
    kx is the soil permeability
    gw is the weight density of water
    gs is the effective weight density of the soil
    nel is the element number
```

node1 and node2 are end node numbers
mat is the material number for the element
print .ne. 0, element results not printed
inc is the node increment used for generation
gen is the number of elements to generate

End input with a blank line.

Whether the elements have linear, quadratic, or cubic displacement and/or pressure variation, only the two end nodes are specified. The variation for displacements is specified by disp, and the variation for the pressure is specified by pressure. For quadratic and cubic elements, the interior nodes are generated automatically. E.g., disp=cubic pressure=quad would mean an element would have 4 displacement nodes and 3 pressure nodes. For a quadratic variation, node 3 is placed in the center of the element. For a cubic variation, nodes 3 and 4 are placed at the third points. If a physical node already exists at this location, then that node is used. If a node does not exist, then a new node is created. Therefore, be sure to specify a sufficient number of nodes in the nodes command to include generated nodes.

Prior to forming the stiffness of this element, the time step must be defined in the variable .biot_dt and the integration factor theta (2-step family from forward Euler (theta = 0) to backward Euler (theta = 1) must be defined in .biot_theta. Note: if the boundary conditions involve specified nonzero flow, then use theta = 1; otherwise, errors will result.

Regardless of the number of nodes, 3-pt Gauss integration is used to calculate the stiffness. This scheme is exact for all combinations of displacement and pressure variation.

On input, created arrays are:

```
  .biot1d234_mp(m,5) -> modulus, area, kx, gw, gs
  .biot1d234_el(10,n) -> nodes, material #, print code, #disp_nodes,
                         # pressure_nodes
  .biot1d234_len(n)  -> element length
  .biot1d234_st(n,15)-> coordinate, stress, excess pore pressure, flow,
  and liquifaction ratio for each gauss point
```

For stiffness calculation, 3-pt Gauss integration is used.

For state calculation, global coordinate, effective stress, excess pore pressure, flow and liquifaction ratio at each integration point are put in .biot1d234_st.

For response calculation, global coordinate, displ., effective stress, excess pore pressure, and flow are calculated for local coordinates in biot1d234_lc. Results are put in .biot1d234_resp(*,5).

For state output, results in .biot1d234_st are printed.

For response output, results in .biot1d234_resp are printed.

See Also

pbiot1d234   pstate   presponse

pbiot1d234

Command Syntax

```
pbiot1d234
  Print biot1d234 element data
```

See Also

biot1d234

biot2d3to9

2-D element for consolidation of a linear, elastic medium

Implementation assumes element is in the X-Z plane, with gravity acting in the negative Z direction and the medium's surface is at Z=0.

Command Syntax

```
biot2d3to9 m=? n=?  [type=?]  [gauss=?]
m=mat#  e=emodulus  nu=nu  kx=kx  kz=kz                            &
    gammaw=gw  gammas=gs  (1 record/matl)
n=nel  nodes=node1,node2,...,node9  mat=mat  [print=print]         &
    [gauss=gauss]  [inc=inc1,inc2,inc3  gen=gen]                   &
    [inc_2d=inc1_2d,inc2_2d,inc3_2d  gen_2d=gen_2d  inc_el=inc_el] &
    [nodesp=nodep1,nodep2,...,nodep9]
```

m is the number of different materials
n is the number of elements
type is the element type

```
     2 -> plane strain (default)
     3 -> axisymmetric
```

gauss is the order of Gauss integration for the stiffness matrices

```
     1 -> 1 x 1
     2 -> 2 x 2
     3 -> 3 x 3 (default)
     ...
    10 -> 10 x 10
```

mat# is the material number
emodulus is the modulus of elasticity
nu is the Poisson ratio
kx and kz are the coefficients of permeability
gw is the weight density of water
gs is the effective weight density of the soil

nel is the element number
node1 thru node9 are node numbers (3 to 9 nodes)
mat is the material number for the element
print .ne. 0 -> element results not printed
gauss overrides the previously specified integration order
inc1, inc2, inc3 are node increments in a "linear sequence"
gen is the number of elements to generate in a sequence
inc1_2d, inc2_2d, inc3_2d are node increments between sequences
gen_2d is the number of linear sequences to generate
inc_el is the element increment between sequences
nodep1 thru nodep9 are the nodes with pressure DOFS; if these are the same as the displacement nodes, they are not given. That is, the default is nodep1 = node1, etc.

Nodes 1 to 4 are the corner nodes for quad elements and are specified counterclockwise. Nodes 5 to 8 are the midnodes on the edges (see sketch below), while node 9 is the center node. For triangular elements, only the first three nodes are to be specified. The same holds true for the pressure nodes.

A "linear sequence" of elements can be generated by specifying inc1, inc2, inc3, and gen. In a linear sequence, nodes 1, 2, and 5 are incremented by inc1; nodes 6, 8, and 9 are incremented by inc2; and nodes 3, 4, and 7 are incremented by inc3. gen is the number of elements to generate, so a sequence will have gen+1 elements. To generate a 2D patch of elements, multiple sequences can be specified; inc1_2d, inc2_2d, and inc3_2d are used to increment the node numbers from one sequence to the next. gen_2d is the number of additional sequences. The element numbers in two successive sequences differ by inc_el (default = numgen+1).

End input with a blank line.

On input, created arrays are:

```
  .biot2d3to9_et(1)    -> 2 or 3 for plane strain or axisymmetric
  .biot2d3to9_mp(m,7)  -> emodulus, Poisson ratio, thickness,
                          kx, kz, gw, gs
  .biot2d3to9_el(21,n) -> node1 - node9, nodep1 - nodep9, material #,
                          print code, # gauss pts
```

This element uses a value of 1 for the thickness.

Prior to forming the stiffness of this element, the time step must be defined in the variable .biot_dt and the integration factor theta (2-step family from forward Euler (theta = 0) to backward Euler (theta = 1) must be defined in .biot_theta. Note: if the boundary conditions involve specified nonzero flow, then use theta = 1; otherwise, errors will result.

For state calculation, X-Z coordinates, effective stress, excess pore pressure, flow, and liquefaction ratio at each Gauss point are put in .biot2d3to9_st(n,10*gauss^2). The stresses are in the global coordinate system and are stored in order Sxx, Syy, Sxy, Szz for each point. Similarly, the flow is a vector with X and Z components.

For response calculation, the same quantities as for state are calculated, but this time for local coordinates in biot2d3to9_lc(#pts,2). Results are put in .biot2d3to9_resp(#pts*n,10). If instead global coordinates are given in biot2d3to9_gc(#pts,2), then results for those points are put in .biot2d3to9_resp(#pts,10). The vector .biot2d3to9_index(#pts) maps the data points to the element in which it falls.

For state output, results in .biot2d3to9_st are printed.

For response output, results in .biot2d3to9_resp are printed.

```
         node4                 node7                 node3
           X---------------------X---------------------X
           |                                           |
           |                                           |
           |                                           |
           |                                           |
           |                                           |
           X node8               X                     X node6
           |                   node9                   |
           |                                           |
           |                                           |
           |                                           |
           |                                           |
           X---------------------X---------------------X
         node1                  node5                node2


    See Also
      pbiot2d3to9   presponse   pstate
```

pbiot2d3to9

```
    Command Syntax
      pbiot2d3to9
        Print biot2d3to9 element data

    See Also
      biot2d3to9
```

cable

```
    Elastic catenary cable element

    Command Syntax
      cable  m=?  n=?  [maxiter=?]  [tol=?]  [-restart]
      m=seg_prop  e=emodulus  a=area  w=wx,wy,wz                    (m records)
      n=nel  nodes=node1,node2  [#segs=#segs]  [tension=tenX,tenY,tenZ]
        seg=seg  mat=seg_prop  L=length                         (#segs records)

        m is the number of different cable properties
        n is the number of elements
        maxiter is the max. # of iterations on the tension (default=30)
        tol is the relative tolerance on the end point position (default=1.e-5)

        If -restart is specified, then the data from the database is used to
        initialize the element. This option must be used if a database file
        containing cable data is read.

        For each set of cable properties:
          seg_prop is the segment property number
          emodulus is the modulus of elasticity
          area is the cross sectional area
          wx,wy,wz are the weight/unit length components in global coordinates

        For each element:
          nel is the element number
          nodes are the two nodes of the element
          #segs is the number of different segments (default=1)
```

tension is the initial estimate of the tension

seg is the segment number
seg_prop is the segment property number
length is the unstretched segment length

The element is based on small strain elastic catenary theory. A shooting method is used to solve the two-point boundary value problem. Specifically, iteration on the tension at end 1 is carried out until the distance between end 2 and node 2, divided by the element length, is less than or equal to the tolerance (tol). For information on the formulation, see H.R. Riggs and T. Leraand, "Efficient Static Analysis and Design of Flexible Risers," J. Off. Mech. Arctic Engrg., Vol. 113, pp. 235-240, 1991, and H.R. Riggs and T. Leraand, "A Robust Element for Static Analysis of Marine Cables," Proc. Third International Offshore and Polar Engineering Conference, Singapore, Vol. 2, pp. 357-363, 1993. The element described in those papers includes fluid drag; this element does not.

See Also
pcable

pcable

command Syntax

```
pcable
  Print cable element data
```

See Also
cable

contact_spring

Nonlinear, contact spring element

Command Syntax

```
contact_spring m=? n=?
m=mat k=Kx,Ky,Kz                                  (1 record/material)
n=nel  node=node  mat=mat  [print=print]  [dist=distance]  [dir=lx,ly,lz]
```

m is the number of different materials
n is the number of elements

mat is the material number
Kx is the spring stiffness in the local x-direction
Ky is the spring stiffness in the local y-direction
Kz is the spring stiffness in the local z-direction

nel is the element number
node is the node number to which the contact spring is attached
distance is the distance from the node to 'ground'
mat is the material number
print .ne. 0 -> element printout suppressed
lx, ly, lz element orientation vector (see below)

End input with a blank line.

The contact spring element can be assigned to a node that may contact rigid 'ground'. The distance in the local x-direction from the original position of the node to ground is defined by dist. The local x-direction is defined by the orientation vector, which is directed from ground to the node. The default values for this direction vector are dir=0,0,1. In this case, the node is dist "above" ground in the global Z direction. (At present, only dir=0,0,1 is supported.) Once a node contacts ground, the spring stiffness specified by Kx, Ky, and Kz is introduced. The stiffness Kx tries to keep the node from going below ground. Ky and Kz prevent slipping along the ground (a plane normal to the direction vector). Clearly, Kx, Ky, and Kz function as penalty parameters, and therefore they should be relatively large. The element deformations are defined as the 'distance' below ground of the node, and the amount of slip along the normal plane.

On input, created arrays are:

```
  .contactspring_mp(m,3)    -> Kx, Ky, Kz
  .contactspring_el(n,3)    -> node, material, print
  .contactspring_dr(3,n)    -> lx, ly, lz
  .contactspring_dist(n)    -> distance
  .contactspring_st(5,n)    -> element deformations
```

For state calculation, element deformations are put in .contactspring_st.

For response calculation, element does nothing.

For state output, results in .spring_st are printed.

For response output, results in .spring_st are printed for those springs that are in contact. If the unformatted write options on the presponse command are specified, the results for all elements are written to the file project_name.cspr. The data are written: element #, node, indentation, slip-1, slip-2, displ-1 at slip, and displ-2 at slip.

See Also

pcontact_spring   pstate   presponse

pcontact_spring

Command Syntax

```
pcontact_spring
  Print contact spring element data
```

See Also

contact_spring

d1l234

1-D, linear, 2,3, or 4 node, isoparametric "rod" element

Implementation assumes element is parallel to X-axis.

Command Syntax

```
d1l234 m=? n=?
m=matl  e=emodulus  a=area  [mbar=density]  [k=kdsp] (1 record/matl)
n=nel  nodes=node1,node2,node3,node4  mat=mat  [print=print]     &
    [inc=inc  gen=gen]

  m is the number of different materials
  n is the number of elements
```

```
      matl is the material number
      emodulus is the modulus of elasticity
      area is the cross sectional area
      density is density/unit length (unused)
      kdsp is a distributed spring stiffness along length of member

      nel is the element number
      node1 thru node4 are node numbers (2,3 or 4 nodes)
      mat is the material number for the element
      print .ne. 0, element results not printed
      inc is the node increment used for generation
      gen is the number of elements to generate

    End input with a blank line.

    On input, created arrays are:
      .d1l234_mp(m,4) -> modulus, area, dens, ksp
      .d1l234_el(n,7) -> node1 - node4, material, print code, #nodes
      .d1l234_len(n)  -> element length
      .d1l234_st(n,8) -> coordinate and force for each gauss point

    For stiffness calculation, exact integration is used.

    For state calculation, coordinates and stresses are put in .d1l234_st.

    For response calculation, global coordinate, displ., and force are
    calculated for local coordinates in d1l234_lc. Results are put in
    .d1l234_resp(*,3).

    For state output, results in .d1l234_st are printed.

    For response output, results in .d1l234_resp are printed.

  See Also
    pd1l234   pstate   presponse
```

pd1l234

```
  Command Syntax
    pd1l234
      Print d1l234 element data

  See Also
    d1l234
```

d1l234v2

```
  1-D, linear, 2,3, or 4 node, isoparametric "rod" element, version 2
      Implementation assumes element is parallel to X-axis.

  Command Syntax
    d1l234v2 m=? n=? [-linear] [-quad] [-cubic]
    m=matl  e=emodulus  a=area [mbar=density]  [k=kdsp]  (1 record/matl)
    n=nel  nodes=node1,node2  mat=mat [print=print]  [inc=inc  gen=gen]

      m is the number of different materials
      n is the number of elements
```

-linear indicates a 2-node, linear displ element
-quad indicates a 3-node, quadratic displ element
-cubic indicates a 4-node, cubic displ element
matl is the material number
emodulus is the modulus of elasticity
area is the cross sectional area
density is density/unit length (unused)
kdsp is a distributed spring stiffness along length of member

nel is the element number
node1 and node2 are the two end node numbers
mat is the material number for the element
print .ne. 0, element results not printed
inc is the node increment used for generation
gen is the number of elements to generate

End input with a blank line.

Whether the elements have linear, quadratic, or cubic displacement variation, only the two end nodes are specified. For quadratic and cubic elements, the interior nodes are generated automatically. For quadratic elements, node 3 is placed in the center of the element. For cubic elements, nodes 3 and 4 are placed at the third points. If a physical node already exists at this location, then that node is used. If a node does not exist, then a new node is created. Therefore, be sure to specify a sufficient number of nodes in the nodes command to include generated nodes. The automatic generation of interior nodes is the main difference between this element and d1l234 (in addition to extensive code changes).

On input, created arrays are:

```
.d1l234_mp(m,4) -> modulus, area, dens, ksp
.d1l234_el(7,n) -> node1 - node4, material, print code, #nodes
.d1l234_len(n)  -> element length
.d1l234_st(n,8) -> coordinate and force for each gauss point
```

For stiffness calculation, exact integration is used.

For state calculation, coordinates and stresses are put in .d1l234_st.

For response calculation, global coordinate, displ., and force are calculated for local coordinates in d1l234_lc. Results are put in .d1l234_resp(*,3).

For state output, results in .d1l234_st are printed.

For response output, results in .d1l234_resp are printed.

See Also
pd1l234v2   nodes   pstate   presponse

pd1l234v2

Command Syntax
pd1l234v2
Print d1l234v2 element data

See Also
d1l234v2

d2l3to9

2-D, linear, 3 to 9 node, isoparametric element for 2-D elasticity. Implementation assumes element is in the X-Y plane.

Command Syntax

```
d2l3to9 m=? n=? [type=?]  [gauss=?]
m=mat#  e=emodulus  [nu=nu  t=thickness  [mass=mass]              &
    pat=pat  bx=distr_x  by=distr_y  [gaussf=gaussf]  (1 record/matl)
n=nel  nodes=node1,node2,...,node 9  mat=mat [print=print]        &
    [gauss=gauss]  [inc=inc1,inc2,inc3 gen=gen]                   &
    [inc_2d=inc1_2d,inc2_2d,inc3_2d  gen_2d=gen_2d  inc_el=inc_el]
```

m is the number of different materials
n is the number of elements
type is the element type

```
     1 -> plane stress (default)
     2 -> plane strain
     3 -> axisymmetric
```

gauss is the order of Gauss integration for stiffness and mass

```
     1 -> 1 x 1
     2 -> 2 x 2
     3 -> 3 x 3 (default)
     ...
    10 -> 10 x 10
```

mat# is the material number
emodulus is the modulus of elasticity
nu is the Poisson ratio
thickness is the element thickness
mass is the mass density (per unit volume)
pat is the load pattern number for the body forces
distr_x and distr_y specify distributions for body forces (per unit volume) in the X-Y coordinates. The distributions are defined by the body_frc2d command.
gaussf is the integration order for the body forces (default = 3).

nel is the element number
node1 thru node9 are node numbers (3 to 9 nodes)
mat is the material number for the element
print .ne. 0 -> element results not printed
gauss overrides the previously specified integration order
inc1, inc2, inc3 are node increments in a "linear sequence"
gen is the number of elements to generate in a sequence
inc1_2d, inc2_2d, inc3_2d are node increments between sequences
gen_2d is the number of linear sequences to generate
inc_el is the element increment between sequences

Nodes 1 to 4 are the corner nodes for quad elements and are specified counterclockwise. Nodes 5 to 8 are the midnodes on the edges (see sketch below), while node 9 is the center node. For triangular elements, only the first three nodes are to be specified.

A "linear sequence" of elements can be generated by specifying inc1, inc2, inc3, and gen. In a linear sequence, nodes 1, 2, and 5 are incremented by inc1; nodes 6, 8, and 9 are incremented by inc2; and nodes 3, 4, and 7 are incremented by inc3. gen is the number of elements to generate, so a sequence will have gen+1 elements. To generate a 2D patch of elements, multiple sequences can be specified; inc1_2d, inc2_2d, and inc3_2d are used to increment the node numbers from one sequence to the next. gen_2d is the number of additional sequences. The element numbers in two successive sequences differ by inc_el (default = numgen+1).

End input with a blank line.

On input, created arrays are:

```
  .d2l3to9_et(1)    -> 1,2,3 for plane stress, strain, or axisym.
  .d2l3to9_mp(m,8)  -> emodulus, Poisson ratio, thickness, mass,
                       load pattern, bx, by, gaussf
  .d2l3to9_el(12,n) -> node1 - node9, material #, print code, # gauss pts
```

This element forms a consistent mass matrix.

For state calculation, X-Y coordinates and stress at each Gauss point are put in .d2l3to9_st(n,6*gauss^2). The stresses are in the global coordinate system and are stored in order Sxx, Syy, Sxy, Szz for each point.

For response calculation, global coordinates and stresses are calculated for local coordinates in d2l3to9_lc(#pts,2). Results are put in .d2l3to9_resp(#pts*n,6). See the explanation for state calculation for the order of stresses. If instead global coordinates are given in d2l3to9_gc(#pts,2), then results for those points are put in .d2l3to9_resp(#pts,6). The vector .d2l3to9_index(#pts) maps the data points to the element in which it falls.

For state output, results in .d2l3to9_st are printed.

For response output, results in .d2l3to9_resp are printed.

For error estimation, the strain-energy option and the "user" option are supported. With strain-energy based error estimation, the "exact" stresses are expected in .d2l3to9_xst(n,6*#pts) in the same format as .d2l3to9_st; these values may be the result of some smoothing procedure. The finite element stresses are expected in .d2l3to9_st(n,6*#pts). The number of integration points used for the element error is based on the number of columns of these matrices. If the same integration points used for the stiffness calculation are used, .d2l3to9_st can be generated by the state command; otherwise, it can be generated by the response command and rearranged by the unwrap command. The X,Y coordinates and error of each integration point are put in .d2l3to9_err(n,3*#pts). The integrated element error and relative error (element error/global error) are put in .d2l3to9_elerr(n,2).

The "user" error estimation option functions similarly, except that the error function is expected in .d2l3to9_ruser(n,3*#pts), where #pts is the number of integration points and the first two columns contain the global coordinates. This option creates the same arrays as the strain-energy option. Because .d2l3to9_ruser can contain the values of any function, this option can be used for the numerical evaluation of an integral over a two-dimensional domain. For example, if .d2l3to9_ruser contains all ones, then the element and total volumes will be calculated.

```
 node4                 node7                 node3
   X---------------------X---------------------X
   |                                           |
   |                                           |
   |                                           |
   |                                           |
   |                                           |
   X node8               X                     X node6
   |                   node9                   |
   |                                           |
   |                                           |
   |                                           |
   |                                           |
   X---------------------X---------------------X
 node1                  node5                node2
```

See Also
  gauss_pts   pd2l3to9   presponse   pstate unwrap

pd2l3to9

Command Syntax

```
pd2l3to9
  Print d2l3to9 element data
```

See Also
  d2l3to9

d2ltri

2-D, linear, 3 to 6 node, isoparametric triangular element for 2-D elasticity.
  Implementation assumes element is in the X-Y plane.

Command Syntax

```
d2ltri m=? n=?  [type=?]  [intpt=?]  [tol=?]
m=mat#  e=emodulus  nu=nu  t=thickness  [mass=mass]              &
    pat=pat  bx=distr_x  by=distr_y  [intf=intf]    (1 record/matl)
n=nel  nodes=node1,node2,...,node 6  mat=mat  [print=print]       &
    [intpt=intpt]  [inc=inc1,inc2,inc3  gen=gen]                  &
    [inc_2d=inc1_2d,inc2_2d,inc3_2d  gen_2d=gen_2d  inc_el=inc_el]

  m is the number of different materials
  n is the number of elements
  type is the element type
       1 -> plane stress (default)
       2 -> plane strain
       3 -> axisymmetric
  intpt is the order of integration
       1 -> 1 pt. integration
       3 -> 3 pt. integration (default)
       4 -> 4 pt. integration
       7 -> 7 pt. integration
       9 -> 9 pt. integration
  tol is a tolerance on nodal coordinates (see below)

  mat# is the material number
```

emodulus is the modulus of elasticity
nu is the Poisson ratio
thickness is the element thickness
mass is the mass density (per unit volume)
pat is the load pattern number for the body forces
distr_x and distr_y specify distributions for body forces (per unit volume) in the X-Y coordinates. The distributions are defined by the body_frc2d command
intf is the integration order for the body forces and mass(default = 4)

nel is the element number
node1 thru node6 are node numbers (3 or 6 nodes)
mat is the material number for the element
print .ne. 0 -> element results not printed
intpt overrides the previously specified integration order
inc1, inc2, inc3 are node increments in a "linear sequence"
gen is the number of elements to generate in a sequence
inc1_2d, inc2_2d, inc3_2d are node increments between sequences
gen_2d is the number of linear sequences to generate
inc_el is the element increment between sequences

Nodes 1 to 3 are the corner nodes for the elements and are specified counterclockwise. Nodes 4 to 6 are the midnodes on the edges (see sketch below). If a negative value is input for a midside node, then the coordinates of the node are calculated midway between the corresponding vertex nodes. If a node does not exist at that location (within tolerance of tol), a node will be generated with a node number one greater than the previous maximum defined node number. In this case, the restraint codes for the node of absolute value of the number specified will be used for the new node (e.g., if -10 is specified, then restraint codes for node 10 will be used). Normally, it will be convenient to use the negative of one of the vertex nodes.

When specifying intf to calculate the mass matrix, one should be aware that with the 6-node element and 3-pt integration, the integration points are at the midside nodes. This leads to a diagonal mass matrix with zeroes for the vertex nodes. An intf of 1, 4, or higher will avoid this.

A "linear sequence" of elements can be generated by specifying inc1, inc2, inc3, and gen. In a linear sequence, nodes 1, 2, and 4 are incremented by inc1; nodes 5 and 6 are incremented by inc2; and node 3 is incremented by inc3. gen is the number of elements to generate, so a sequence will have gen+1 elements. Multiple sequences can be specified; inc1_2d, inc2_2d, and inc3_2d are used to increment the node numbers from one sequence to the next. gen_2d is the number of additional sequences. The element numbers in two successive sequences differ by inc_el (default = numgen+1). Note: Node increments are added to positive node numbers and subtracted from negative node numbers.

End input with a blank line.

```
On input, created arrays are:
.d2ltri_et(1) -> 1, 2, 3 for plane stress, strain, or axisym.
.d2ltri_mp(m,8) -> emodulus, Poisson ratio, thickness, mass,       &
                     load pattern, bx, by, intf
.d2ltri_el(n,9) -> node1 - node6, material #, print code, #integ. pts
```

This element forms a consistent mass matrix.

For state calculation, X,Y coordinates and stress (Sx, Sy, Sxy, Sz) at each integration point are put in .d2ltri_st(n,6*#pts).

For response calculation, X,Y coordinates and stresses are calculated for area coordinates (3 values/point) in d2ltri_lc(#pts,3). Results are put in .d2ltri_resp(n*#pts,6).

For state output, results in .d2ltri_st are printed.

For response output, results in .d2ltri_resp are printed.

For error estimation, the strain-energy option and the "user" option are supported. With strain-energy based error estimation, the "exact" stresses are expected in .d2ltri_xst(n,6*#pts), i.e., in the same form as .d2ltri_st; these values may be the result of some smoothing procedure. The finite element stresses are expected in .d2ltri_st(n,6*#pts). The number of integration points used for the element error is based on the number of columns of these matrices. If the same integration points used for the stiffness calculation are used, .d2ltri_st can be generated by the state command; otherwise, it can be generated by the response command and rearranged by the unwrap command. The X,Y coordinates and error of each integration point are put in .d2ltri_err(n,3*#pts). The integrated element error and relative error (element error/global error) are put in .d2ltri_elerr(n,2).

The "user" error estimation option functions similarly, except that the error function is expected in .d2ltri_ruser(n,3*#pts), where #pts is the number of integration points and the first two columns contain the global coordinates. This option creates the same arrays as the strain-energy option. Because .d2ltri_ruser can contain the values of any function, this option can be used for the numerical evaluation of an integral over a two-dimensional domain. For example, if .d2ltri_ruser contains all ones, then the element and total volumes will be calculated.

```
                         node3
                           X
                          | |
                         |   |
                        |     |
                       |       |
                      |         |
              node6  X          X node5
                    |             |
                   |               |
                  |                 |
                 |                   |
                |                     |
               X-----------X-----------X
             node1       node4       node2
```

See Also

pd2ltri   presponse   pstate   tri_intpts   unwrap

pd2ltri

```
    Command Syntax
      pd2ltri
        Print d2ltri element data

    SeeAlso
      d2ltri
```

iFEM2D

```
    2-D, 3 to 9 node, isoparametric element for 2-D nonlinear iFEM.
      Implementation assumes element is in the X-Y plane.

    Command Syntax
      iFEM2D n=? [gauss=?,?]
      n=nel nodes=node1,node2,...,node 9  print=print  inc=inc1,inc2,inc3   &
           gen=gen  inc_2d=inc1_2d,inc2_2d,inc3_2d  gen_2d=gen_2d           &
           inc_el=inc_el

        n is the number of elements
        gauss is the order of Gauss integration for stiffness and "loads"
             n,m -> n x m in the xi and eta directions (default = 4x4)

        nel is the element number
        node1 thru node9 are node numbers (3 to 9 nodes)
        print .ne. 0 -> element results not printed
        inc1, inc2, inc3 are node increments in a "linear sequence"
        gen is the number of elements to generate in a sequence
        inc1_2d, inc2_2d, inc3_2d are node increments between sequences
        gen_2d is the number of linear sequences to generate
        inc_el is the element increment between sequences

        Nodes 1 to 4 are the corner nodes for quad elements and are specified
        counterclockwise. Nodes 5 to 8 are the midnodes on the edges (see
        sketch below), while node 9 is the center node. For triangular
        elements, only the first three nodes are to be specified. Note: this
        element is meant to be used with 6 nodes, with quadratic interpolation
        in the longitudinal direction and linear interpolation in the
        transverse direction.

        A "linear sequence" of elements can be generated by specifying inc1,
        inc2, inc3, and gen. In a linear sequence, nodes 1, 2, and 5 are
        incremented by inc1; nodes 6, 8, and 9 are incremented by inc2; and
        nodes 3, 4, and 7 are incremented by inc3. gen is the number of
        elements to generate, so a sequence will have gen+1 elements. To
        generate a 2D patch of elements, multiple sequences can be specified;
        inc1_2d, inc2_2d, and inc3_2d are used to increment the node numbers
        from one sequence to the next. gen_2d is the number of additional
        sequences. The element numbers in two successive sequences differ by
        inc_el (default = numgen+1).

      End input with a blank line.

      On input, created arrays are:
        .ifem2d_int(2)   -> # gauss pts in xi and eta directions
        .ifem2d_el(10,n) -> node1 - node9, print code

      For state calculation, X-Y coordinates, X-Y displacements, and strain at
```

each Gauss point are put in .ifem2d_st(n,7*gauss_xi*gauss_eta). The strains are in the global coordinate system and are stored in order Exx, Eyy, Exy for each point.

For response calculation, global coordinates and strains are calculated for local coordinates in ifem2d_lc(#pts,2). Results are put in .ifem2d_resp(#pts*n,7). See the explanation for state calculation for the data and order of strains. If instead global coordinates are given in ifem2d_gc(#pts,2), then results for those points are put in .ifem2d_resp(#pts,7). The vector .ifem2d_index(#pts) maps the data points to the element in which it falls.

For state output, results in .ifem2d_st are printed.

For response output, results in .ifem2d_resp are printed.

No error estimation feature is supported.

See Paczkowski, K., Riggs, H.R., 2007, "An inverse finite element strategy to recover full-field, large displacements from strain measurements," Proc., 26th International Offshore Mechanics and Arctic Engineering Conference, paper OMAE2007-29730 for details on this element.

```
node4                node7                node3
  X---------------------X---------------------X
  |                                           |
  |                                           |
  |                                           |
  |                                           |
  |                                           |
  X node8               X                     X node6
  |                   node9                   |
  |                                           |
  |                                           |
  |                                           |
  |                                           |
  X---------------------X---------------------X
node1                  node5                node2
```

See Also

gauss_pts   piFEM2D   presponse   pstate

piFEM2D

Command Syntax

piFEM2D

Print iFEM2D element data

See Also

iFEM2D

interface

Quadrilateral (and triangular) interface element

Command Syntax (option 1)

```
interface  n=?
n=nel  nodes=node1,node2,node3,node4  [print=print]  [face=face]     &
       [gen=gen inc=inc1,inc2]                                       &
       [gen_2d=gen_2d  inc_2d=inc1_2d,inc2_2d  inc_el=inc_el]
```

n is the maximum element number specified

nel is the element number
node1 thru node4 are node numbers
mat is the material number for the element
print .ne. 0 -> element results not printed
inc1, inc2 are node increments in a "linear sequence"
gen is the number of elements to generate in a sequence
inc1_2d, inc2_2d are node increments between sequences
gen_2d is the number of linear sequences to generate
inc_el is the element increment between sequences
face specifies the face on which the pressure acts

```
     =  0 -> no pressure acts ("dry" element) - default
     = -1 -> pressure acts on negative face (-z)
     =  1 -> pressure acts on positive face (+z)
```

Command Syntax (option 2)

```
interface  -subdivide  range=?,?  nxm=?,?
```

Option 2 subdivides previously defined elements
- range specifies a range of element ID numbers; all elements in the range are divided
- nxm specifies how many elements to subdivide each element into. For example, nxm=2,3 would subdivide each element into 6 elements; 2 in the 1-2 direction and 3 in the 1-4 direction.

Nodes 1 to 4 are the corner nodes for quadrilateral elements, specified in a counterclockwise order when looking at the positive face. For triangular elements, if node 4 is not specified it will be set equal to node 3.

A "linear sequence" of elements can be generated by specifying inc1, inc2, and gen. In a linear sequence, nodes 1 and 2 are incremented by inc1; nodes 3 and 4 are incremented by inc2. gen is the number of elements to generate, so a sequence will have gen+1 elements. Multiple sequences can be specified; inc1_2d and inc2_2d are used to increment the node numbers from one sequence to the next. gen_2d is the number of additional sequences. The element numbers in two successive sequences differ by inc_el (default = numgen+1).

The local x-axis (axis-1) is directed from node 1 to node 2. The local y-axis (axis-2) lies in the plane defined by nodes 1-2-4, is normal to the x-axis, and is directed "toward" node 4. The local z-axis (axis-3) follows from the right-hand-rule.

The normal pressures act in the local z-direction.

End input with a blank line.

On input, created arrays are:

```
.interface_el(8,n)  -> node1 - node4, print code, face, ID#
.interface_pressure -> 1 if -pressure specified, otherwise 0
```

The columns of .interface_el are based on the number of defined elements, not the maximum ID# (i.e., ID numbers don't have to be sequential).

This element does very little. It has no stiffness or mass. The following is an example of one possible use. For visualization purposes, it can be used to represent the surface of a rigid body with the nodes of this element kinematically constrained to the CG of the rigid body so that the displacements are kinematically constrained. The face parameter is included to allow one side to be distinguished from the other (for example, as 'wet').

pinterface

Command Syntax

```
pinterface
  Print interface element data
```

See Also

interface

isomin6

Linear, 6-node, 18 DOF triangular, isoparametric Mindlin plate element. Implementation assumes element is in the X-Y plane.

Command Syntax

```
isomin6  m=?  n=?  [q=q_vec]
m=mat# e=E1,E2,E3 g=G12,G23,G13 nu=nu12,nu23,nu13               &
       t=thickness mass=mass iso=iso (1 record/matl)
n=nel nodes=node1,node2,node3,node4,node5,node6 mat=mat print=print  &
      inc=inc1,inc2,inc3 gen=gen inc_2d=inc1_2d,inc2_2d,inc3_2d      &
      gen_2d=gen_2d inc_el=inc_el pat=pat [q=q1,q2,q3,q4,q5,q6]
```

m is the number of different materials
n is the number of elements
q_vec is the name of a vector in the database in which the ith element is the normal pressure at node i

mat# is the material number
E1, E2, E3 are the moduli of elasticity
G12, G23, G13 are the shear moduli
nu12, nu23, nu13 are the Poisson ratios
thickness is the element thickness
mass is the mass per unit volume
iso = 0 -> isotropic material (default)
    = 1 -> orthotropic material

nel is the element number
node1 thru node6 are node numbers
mat is the material number for the element
print .ne. 0 -> element results not printed
inc1 and inc2 are node increments in a "linear sequence"
gen is the number of elements to generate in a sequence
inc1, inc2 inc3 are node increments in a "linear sequence"
gen is the number of elements to generate in a sequence
inc1_2d, inc2_2d, inc3-2d are node increments between sequences
gen_2d is the number of linear sequences to generate
inc_el is the element increment between sequences

pat is the load pattern number for the normal pressures
q1,q2,q3,q4,q5,q6 are normal pressures for nodes (these values override those defined by q_vec, if any.)

Nodes 1 to 3 are the corner nodes and are specified counterclockwise. Nodes 4 to 6 are the midnodes.

A "linear sequence" of elements can be generated by specifying inc1, inc2, inc3, and gen. In a linear sequence, nodes 1, 2, and 4 are incremented by inc1; nodes 5, 6 are incremented by inc2; and nodes 3 is incremented by inc3. gen is the number of elements to generate, so a sequence will have gen+1 elements. To generate a 2D patch of elements, multiple sequences can be specified; inc1_2d, inc2_2d, and inc3_2d are used to increment the node numbers from one sequence to the next. gen_2d is the number of additional sequences. The element numbers in two successive sequences differ by inc_el (default = numgen+1).

End input with a blank line.

The element is restricted to lie in an x-y plane (z=constant). For an orthotropic material, the material parameters are specified in the global coordinate system.

The normal pressures act in the z-direction.

On input, created arrays are:

```
.isomin6_mp(m,14) -> Ei, Gij, nuij, thickness, mass, unused, unused, iso
.isomin6_el(8,n) -> node1 - node6, material #, print code
.isomin6_q(7,n) -> pat, q1, q2, q3, q4, q5, q6
```

If a nonzero mass density is specified, uniform (gravitational) body forces are applied if the vector gravity(4) has been defined. The 4 components of gravity are: load pattern number, gx, gy, and gz, where gi is the gravitational acceleration in the global i direction.

For state calculation, global coordinates and bending stress resultants are stored in .isomin6_stb(n, 4*8) in the order x, y, z, Mx, My, Mxy, Qx, and Qy. These values are calculated at the 4 Gauss points of the triangle.

The element does not calculate the equivalent nodal forces in equilibrium with its stress state, and therefore cannot be used in a nonlinear analysis.

The response option has not been implemented.

The error estimation option has not been implemented.

For state output, the stress resultants in .isomin6_stb are printed.

See Also

pisomin6 min3s min5s min6 pstate

pisomin6

Command Syntax

```
pmin6
```

Print isomin6 element data

See Also
isomin6

min3s

Mindlin 3-D triangular, linear shell element.
Membrane and shear relaxations are included in the implementation.

Command Syntax

```
min3s  m=?  n=?  [-pressure]  [-kg]  [-kf]
m=mat#  e=E1,E2,E3  g=G12,G23,G13  nu=nu12,nu23,nu13  t=thickness      &
       [tb=tbending]  [mass=mass]  [global=global]                     &
       [C_s=relxs]  [C_m=relxm]  [drill=drill_stiff]                   &
       [fdensity=fdensity]  (1 record/matl)
n=nel  mat=mat  nodes=node1,node2,...,node 6  [print=print]            &
      [face=face]  [gen=gen  inc=inc1,inc2,inc3]                       &
    [gen_2d=gen_2d  inc_2d=inc1_2d,inc2_2d,inc3_2d  inc_el=inc_el
```

m is the number of different materials
n is the number of elements
-pressure is a flag to apply nodal pressures in the array .nodal_pressure
-kg is a flag to include geometric stiffness
-kf is a flag to include hydrostatic stiffness (includes geometric stiffness)

mat# is the material number
E1, E2, E3 are the moduli of elasticity
G12, G23, G13 are the shear moduli
nu12, nu23, nu13 are the Poisson ratios
thickness is the element thickness (see below)
tbending is the element bending thickness (default=thickness)
mass is the mass per unit volume
global = 0 -> calc. stress resultants in local coordinates (default)
       = 1 -> calc. stress resultants in global coordinates
relxs is the shear relaxation factor (default = 0.5)
relxm is the membrane relaxation factor (default = 1.0)
drill_stiff is an artificial drilling stiffness (default = 1.e-5)
fdensity is the fluid weight density to calculate the hydrostatic stiffness
Note: Defaults are obtained for the above factors not by specifying 0.0, but by omitting the input for the specific factors or by specifiying a negative value.

nel is the element number
node1 thru node6 are node numbers
mat is the material number for the element
print .ne. 0 -> element results not printed
inc1, inc2, inc3 are node increments in a "linear sequence"
gen is the number of elements to generate in a sequence
inc1_2d, inc2_2d, inc3_2d are node increments between sequences
gen_2d is the number of linear sequences to generate
inc_el is the element increment between sequences
face specifies the face on which the pressure acts
     = 0 -> no pressure acts ("dry" element) - default
     = -1 -> pressure acts on negative face (-z)

= 1 -> pressure acts on positive face (+z)

Nodes 1 to 3 are the vertex nodes and are specified counterclockwise. Nodes 4 to 6 are the midnodes on the edges, which are used to define the initial curvature of the element, and do not have any degree of freedom. If the element is "flat", then nodes 4 to 6 may be specified as zero.

A "linear sequence" of elements can be generated by specifying inc1, inc2, inc3, and gen. In a linear sequence, nodes 1, 2, and 4 are incremented by inc1; nodes 5 and 6 are incremented by inc2; and node 3 is incremented by inc3. gen is the number of elements to generate, so a sequence will have gen+1 elements. Multiple sequences can be specified; inc1_2d, inc2_2d, and inc3_2d are used to increment the node numbers from one sequence to the next. gen_2d is the number of additional sequences. The element numbers in two successive sequences differ by inc_el (default = numgen+1).

The local x-axis (1) is directed from node 1 to node 2. The local y-axis (2) lies in the plane defined by the vertex nodes, is normal to the x-axis, and is directed "toward" node 3. The local z-axis (3) follows from the right-hand-rule.

The material parameters are specified in the local coordinate system. thickness is used for the membrane and shear stiffnesses, while tbending is used for the bending stiffness. If tbending is not input, it will be set equal to thickness. thickness is also used to determine the mass (from the mass density). The element has no drilling dof stiffness, and so an artificial stiffness is added to these dofs that is equal to drill_stiff times the minimum of the element diagonal stiffnesses.

The normal pressures act in the local z-direction.

End input with a blank line.

On input, created arrays are:

```
  .min3s_mp(m,17) -> Ei, Gij, nuij, thickness, mass, global, C_s, C_m,
                     tbending, drill_stiff, fdensity
  .min3s_el(9,n)  -> node1 - node6, material #, print code, face
  .min3s_pressure -> 1 if -pressure specified, otherwise 0
```

For stiffness calculation, the shear and membrane relaxation factors are stored in .min3s_rlx(2,n) for later use in calculating the element state.

If a nonzero mass density is specified, uniform (gravitational) body forces are applied if the vector gravity(4) has been defined. The 4 components of gravity are: load pattern number, gx, gy, and gz, where gi is the gravitational acceleration in the global i direction.

If hydrostatic stiffness is to be calculated, gravity is assumed to act in the negative global z direction. The hydrostatic stiffness is only calculated for "wet" elements, i.e., with face = +-1. The flag -pressure must be specified as well. The hydrostatic pressure is assumed to be in the first column of .nodal_pressure, as specified by the command nodal_pressure. In this case, the load pattern number should be 1 for this pressure. If multiple load patterns are used, they should all have the same pressures specified.

For state calculation, global coordinates and the stress resultants in the local coordinate system are put in .min3s_st(n,11) in the order x, y, z, Nx, Ny, Nxy, Mx, My, Mxy, Qx, and Qy. These values are calculated at the element centroid. If the value of global on the material card is specified to be 1, then the resultants in global coordinates are calculated instead. (This option gives correct results only if the element is in the global X-Y plane.) The element does not calculate the equivalent nodal forces in equilibrium with its stress state, and therefore cannot be used in a nonlinear analysis.

For response calculation, global coordinates and local stress resultants are calculated for local coordinates in min3s_lc(#pts,3). Results are put in .min3s_resp(#pts*n,11). See the explanation for state calculation for the order and for the option for resultants in global coordinates.

For state output, stress resultants in .min3s_st are printed.

For response output, coordinates and stress resultants in .min3s_resp are printed.

For error estimation, the strain-energy option and the "user" option are supported. With strain-energy based error estimation, the "exact" stress resultants are expected in .min3s_xst(n,11*#pts), i.e., in the same form as .min3s_st; these values may be the result of some smoothing procedure. The finite element resultants are expected in .min3s_st(n,11*#pts). The number of integration points used for the element error is based on the number of columns of these matrices. If one-point integration is used, .min3s_st can be generated by the state command. The X,Y,Z coordinates and error of each integration point are put in .min3s_err(n,4*#pts). The integrated element error and relative error (element error/global error) are put in .min3s_elerr(n,2).

The "user" error estimation option functions similarly, except that the error function is expected in .min3s_ruser(n,4*#pts), where #pts is the number of integration points and the first three columns contain the global coordinates. This option creates the same arrays as the strain-energy option. Because .min3s_ruser can contain the values of any function, this option can be used for the numerical evaluation of an integral over a two-dimensional domain. For example, if .min3s_ruser contains all ones, then the element and total volumes will be calculated.

For the theory of this element, see Tessler, A., "A C0 anisoparametric three-node shallow shell element," Computer Methods in Applied Mechanics and Engineering, v. 78, 1990, pp. 89-103.

The basic element has been provided courtesy of Dr. Alex Tessler, Computational Mechanics Branch, NASA Langley Research Center, Hampton, VA. For the hydrostatic stiffness formulation, see Huang, L.L. and Riggs, H.R., "The hdyrostatic stiffness of flexible floating structures for linear hydroelasticity," Marine Structures, v. 13,2000, pp. 91-106.

See Also

nodal_pressure   pmin3s   pstate

pmin3s

Command Syntax

pmin3s
Print min3s element data

See Also
min3s

min5s

Mindlin 3-D quadrilateral, linear shell element consisting of 4 min3s (triangular) elements. Membrane and shear relaxations are included in the implementation. There are three Command Syntax options.

```
         ---------- OPTION 1 ----------
min5s  m=?  n=?  [-pressure]  [-kg]  [-kf]
m=mat# e=E1,E2,E3 g=G12,G23,G13 nu=nu12,nu23,nu13                        &
       t=thickness [tb=tbending] [mass=mass] [local=local]               &
      [C_s=relxs]  [C_m=relxm]  [drill=drill_stiff]                      &
      [iso=iso]  [fdensity=fdensity]                      (1 record/matl)
n=nel nodes=node1,node2,...,node5 mat=mat [print=print]  [face=face]     &
    [gen=gen  inc=inc1,inc2,inc3]                                        &
    [gen_2d=gen_2d  inc_2d=inc1_2d,inc2_2d,inc3_2d  inc_el=inc_el]
```

m is the number of different materials
n is the maximum element number specified
-pressure is a flag to apply nodal pressures in the array .nodal_pressure
-kg is a flag to include geometric stiffness
-kf is a flag to include hydrostatic stiffness (includes geometric stiffness)

mat# is the material number
E1, E2, E3 are the moduli of elasticity
G12, G23, G13 are the shear moduli
nu12, nu23, nu13 are the Poisson ratios
thickness is the element thickness (see below)
tbending is the element bending thickness (default=thickness)
mass is the mass per unit volume

```
local = 0 -> calculate stress resultants in quad coordinates (default)
      = 1 -> calculate stress resultants in triangle coordinates
             Note: local must be 1  for -kg and -kf
```

relxs is the shear relaxation factor (default = 0.5)
relxm is the membrane relaxation factor (default = 1.0)

```
iso   = 0 -> isotropic material (default)
      = 1 -> orthotropic material
```

drill_stiff is the drilling dof stiffness factor (default=1.e-5)
fdensity is the fluid weight density to calculate the hydrostatic stiffness

Note 1: For an isotropic material, the values E1, G12, and nu12 are used.
Note 2: Defaults are obtained for the above factors not by specifying 0.0, but by omitting the input for the specific factors or by specifiying a negative value.

nel is the element number
node1 thru node5 are node numbers
mat is the material number for the element
print .ne. 0 -> element results not printed

inc1, inc2, inc3 are node increments in a "linear sequence"
gen is the number of elements to generate in a sequence
inc1_2d, inc2_2d, inc3_2d are node increments between sequences
gen_2d is the number of linear sequences to generate
inc_el is the element increment between sequences
face specifies the face on which the pressure acts

```
     =  0 -> no pressure acts ("dry" element) - default
     = -1 -> pressure acts on negative face (-z)
     =  1 -> pressure acts on positive face (+z)
```

---------- OPTION 2 ----------

Command Syntax (option 2)

```
  min5s  -subdivide  range=?,?  nxm=?,?
```

Option 2 subdivides previously defined elements
range specifies a range of element ID numbers; all elements in the range are divided
nxm specifies how many elements to subdivide each element into. For example, nxm=2,3 would subdivide each element into 6 elements; 2 in the 1-2 direction and 3 in the 1-4 direction.

---------- OPTION 3 ----------

Command Syntax (option 3)

```
  min5s  -cylinder  n=nel  mat=mat  [print=print]  [face=face]           &
          p1=x1,y1,z1  p2=x2,y2,z2  R=R1,R2  CxL=Cseg,Lseg               &
          [-ring_stiffeners LRseg=LRseg  ring_mat=ring_mat               &
            ring_node3=ring_node3]                                       &
          [-spine  spine_mat=spine_mat  spine_node3=spine_node3          &
            tension_first=tension_first  tension_last=tension_last]
```

Option 3 generates a cylindrical mesh (min5s must have been inititalized previously with option 1, even if 0 elements were defined)
nel is the first element number (of a sequential sequence)
mat, print and face have the same meaning as in option 1
p1 are the center coordinates of the cylinder start
p2 are the center coordinates of the cylinder end
R1,R2 are the radii at the start and end, respectively
Cseg are the number of elements around the circumference
Lseg are the number of elements along the length

If -ring_stiffeners is specified, generate circumferential beam stiffeners:
LRseg is the number of segments along the length separated by stiffneners
ring_mat is the beam material number (define by the beam3d command)
ring_node3 (see beam3d for definition of node3)

If ring stiffeners exist, Lseg must be an integer mulitple of LRseg. If ring_node3 is blank, the node3 direction is along the length of the cylinder on the surface

If -spine is specified, the cylinder is modeled by beam elements along the center of the cylinder. The nodes to the shell elements are constrained via rigid body constraints to the nodes of the spine. The mass and stiffness of the combined structure can be specified either by the material parameters on the shell elements or the beam elements. Be careful not to include the properties twice. Specifically, if the mass and/or stiffness properties are specified by the beam elements, then the shell elements should have zero mass and very small (but not zero) values for modulus and thickness (and vice versa). For this option,

spine_mat is the beam material number (defined by the beam3d command)
spine_node3 corresponds to node3 for the beam (for a vertical cylinder -1 is usually convenient)
tension_first and tension_last, see beam3d

Nodes and elements are numbered around the circumference and then down the length. The shell elements are defined such that the local x-axis is down the length of the cylinder, in the direction from p1 to p2, that is, the line from node1 to node2 is parallel to the axis and in the direction of p2. Nodes are numbered clockwise looking from the outside.

---------- ALL OPTIONS -------

The 5-node min5s element is formed by four 3-node triangular (min3s) elements. Nodes 1 to 4 define the quadrilateral and are specified counterclockwise. Node 5 is the "interior" node, which is common to the four triangles. The connectivity of the triangles in terms of the quadrilateral nodes is 1-2-5, 2-3-5, 3-4-5, and 4-1-5. Normally, node 5 is not specified, in which case it is located at the intersection of the diagonals (straight lines connecting nodes 1 and 3 and nodes 2 and 4). If a node does not exist at this location, one is created. The restraint conditions are the same as for node "1" of the element, if the restraints have already been defined. Note that this "cross-diagonal" pattern is the preferred meshing strategy. Although nodes 1 to 4 are not forced to be coplanar, the element is meant to be used as a flat shell element.

A "linear sequence" of elements can be generated by specifying inc1, inc2, inc3, and gen. In a linear sequence, nodes 1 and 2 are incremented by inc1; nodes 3 and 4 are incremented by inc2; and node 5, if specified, is incremented by inc3. gen is the number of elements to generate, so a sequence will have gen+1 elements. Multiple sequences can be specified; inc1_2d, inc2_2d, and inc3_2d are used to increment the node numbers from one sequence to the next. gen_2d is the number of additional sequences. The element numbers in two successive sequences differ by inc_el (default = numgen+1).

Each triangle in the quadrilateral has its own "triangle" coordinate system. For each triangle, defined by nodes 1-2-3, the x-axis (axis-1) is directed from node 1 to node 2. The local y-axis (axis-2) lies in the plane defined by the vertex nodes, is normal to the x-axis, and is directed "toward" node 3. The local z-axis (axis-3) follows from the right-hand-rule. The quad coordinate system is the same as the triangle coordinates for triangle 1.

The material parameters are specified in the quad coordinate system. thickness is used for the membrane and shear stiffnesses, while tbending is used for the bending stiffness. If tbending is not input, it will be set equal to thickness. thickness is also used to determine the mass (from the mass density).

The normal pressures act in the local z-direction.

End input with a blank line.

On input, created arrays are:

```
  .min5s_mp(m,18) -> Ei, Gij, nuij, thickness, mass, local, C_s, C_m,
                     iso, tbending, drill_stiff, fdensity
  .min5s_el(9,n)  -> node1 - node5, material #, print code, face, ID#
  .min5s_pressure -> 1 if -pressure specified, otherwise 0
```

The columns of .min5s_el are based on the number of defined elements, not the maximum ID# (i.e., ID numbers don't have to be sequential).

For stiffness calculation, the shear and membrane relaxation factors are stored in .min5s_rlx(8,n) for later use in calculating the element state.

If a nonzero mass density is specified, uniform (gravitational) body forces are applied if the vector gravity(4) has been defined. The 4 components of gravity are: load pattern number, gx, gy, and gz, where gi is the gravitational acceleration in the global i direction.

The hydrostatic stiffness is only calculated for "wet" elements, i.e., with face = 1. The flag -pressure must be specified as well. The hydrostatic pressure is assumed to be in the first column of .nodal_pressure, as specified by the command nodal_pressure. In this case, the load pattern number should be 1 for this pressure. If multiple load patterns are used, they should all have the same pressures specified.

For state calculation, global coordinates and the stress resultants in the quad coordinate system are put in .min5s_st(4*n,11) in the order x, y, z, Nx, Ny, Nxy, Mx, My, Mxy, Qx, and Qy. These values are calculated at the element centroid. If the value of local on the material card is specified to be 1, then the resultants in triangle coordinates are calculated instead. The element does not calculate the equivalent nodal forces in equilibrium with its stress state, and therefore cannot be used in a nonlinear analysis.

The optimal stress resultants are stored in .min5s_op(n,11) in the same order as for .min5s_st. For a parallelogram, which consists of 4 constant stress elements in a cross diagonal pattern, the optimal stress at the intersection of the diagonals is the simple average of the stresses in the 4 triangles. For general quadrilaterals, this is an approximation. It is also an approximation for the shears, Qx and Qy, because they vary linearly within each triangle.

For response calculation, global coordinates and stress resultants are calculated for local coordinates in min5s_lc(#pts,3). Results are put in .min5s_resp(#pts*4*n,11). See the explanation for state calculation for the order and for the option for resultants in triangle coordinates.

For state output, stress resultants in .min5s_st are printed.

For response output, coordinates and stress resultants in .min5s_resp are printed.

For error estimation, the strain-energy option and the "user" option are supported. With strain-energy based error estimation, the "exact" stress resultants are expected in .min5s_xst(4*n,11*#pts), i.e., in the same form as .min5s_st; these values may be the result of some smoothing procedure. The finite element resultants are expected in .min5s_st(4*n,11*#pts). The number of integration points used for the element error is based on the number of columns of these matrices. If one-point integration is used, .min5s_st can be generated by the state command. The X,Y,Z coordinates and error of each integration point are put in .min5s_err(4*n,4*#pts). The integrated element error, relative error (element error/global error), and error density (element error/element area) are put in .min5s_elerr(n,3).

The "user" error estimation option functions similarly, except that the error function is expected in .min5s_ruser(4*n,4*#pts), where #pts is the number of integration points and the first three columns contain the global coordinates. This option creates the same arrays as the strain-energy option. Because .min5s_ruser can contain the values of any function, this option can be used for the numerical evaluation of an integral over a two-dimensional domain. For example, if .min5s_ruser contains all ones, then the element and total volumes will be calculated.

Note: If nodes 3 and 4 are equal, then the element degenerates to the min3s element. In general it is better to use min3s elements for triangles, but this option is included for convenience. However, the optimal stresses are not calculated correctly for triangular elements, and the error estimation is not implemented for them.

For the theory of this element, see Tessler, A., "A C0 Anisoparametric Three-Node Shallow Shell Element," Computer Methods in Applied Mechanics and Engineering, v. 78, 1990, pp. 89-103. For the hydrostatic stiffness formulation, see Huang, L.L. and Riggs, H.R., "The hdyrostatic stiffness of flexible floating structures for linear hydroelasticity," Marine Structures, v. 13,2000, pp. 91-106.

See Also

min3s nodal_pressure pmin5s presponse pstate

pmin5s

Command Syntax

```
pmin5s
  Print min5s element data
```

See Also

min5s

min4t

Quadrilateral, linear, Mindlin shell element. Consists of 4 min3s (triangular) elements with the interior node kinematically constrained to the 4 vertex nodes. Shear relaxation is included.

Command Syntax

```
min4t  m=?  n=?  [q=q_vec]  [version=version#]  [constraint=constraint#]
m=mat#  e=E1,E2,E3  g=G12,G23,G13  nu=nu12,nu23,nu13
    t=thickness  [mass=mass]  [local=local]  [gamma=]  [C_s=C_s]     &
    [iso=iso]              (1 record/matl)
n=nel  nodes=node1,node2,node3,node4  mat=mat  [print=print]         &
    [pat=pat q=q1,q2,q3,q4]  [inc=inc1,inc2  gen=gen]
    [inc_2d=inc1_2d,inc2_2d  gen_2d=gen_2d inc_el=inc_el]
```

m is the number of different materials

n is the number of elements

q_vec is the name of a vector in the database for which the ith element is the normal pressure at node i

```
version# controls application of shear relaxation & constraints
version# = 1 -> relax each triangle, then apply constraints (default)
           2 -> apply the constraints on the 4 triangles
                and relax the quad (not implemented)
           3 -> apply the constraints on the 4 triangles and
                relax each triangle (in quad DOFs) separately
constraint# specifies the constraints for the 2 theta DOFs:
            = 1 -> use least squares (default)
              2 -> use 2 constraints only
              3 -> use interior constraints only
```

```
mat# is the material number
E1, E2, E3 are the moduli of elasticity
G12, G23, G13 are the shear moduli
nu12, nu23, nu13 are the Poisson ratios
thickness is the element thickness
mass is the mass per unit volume
local = 0 -> stress resultants in quad coordinates (default)
      = 1 -> stress resultants in triangle coordinates
gamma is the penalty parameter for the membrane drilling DOFs
      default value is G12 * 10^-4
C_s is the shear relaxation factor (default = 0.6)
iso  = 0 -> isotropic material
     = 1 -> orthotropic material.
```

nel is the element number
node1 thru node4 are node numbers
mat is the material number for the element
print .ne. 0 -> element results not printed
inc1 and inc2 are node increments in a "linear sequence"
gen is the number of elements to generate in a sequence
inc1_2d and inc2_2d are node increments between sequences
gen_2d is the number of linear sequences to generate
inc_el is the element increment between sequences
pat is the load pattern number for the normal pressures
q1,q2,q3,q4 are normal pressures for nodes (these values override those defined by q_vec, if any.)

The 4 node min4t element is formed by four 3 node triangular elements. Nodes 1 to 4 define the quadrilateral. "Node 5" is an internal "virtual" node and is common to the four triangles. The connectivity of the triangles in terms of the quadrilateral nodes is 1-2-5, 2-3-5, 3-4-5, and 4-1-5. Node 5 is located at the intersection of the diagonals (straight lines connecting nodes 1 and 3 and nodes 2 and 4). Although nodes 1 to 4 are not forced to be coplanar, the element is meant to be used as a flat shell element.

A "linear sequence" of elements can be generated by specifying inc1, inc2, and gen. In a linear sequence, nodes 1 and 2 are incremented by inc1; and nodes 3 and 4 are incremented by inc2. gen is the number of elements to generate; hence, a sequence will have gen+1 elements. Multiple sequences can be specified; inc1_2d and inc2_2d are used to increment the node numbers from one sequence to the next. gen_2d is the number of additional sequences. The element numbers in two successive sequences differ by inc_el (default = numgen+1).

For each triangle, the local x-axis (1) is directed from node 1 to node 2. The local y-axis (2) lies in the plane defined by the vertex nodes, is normal to the x-axis, and is directed "toward" node 3. The local z-axis (3) follows from the right-hand-rule. The "quad" coordinate system is the same as local coordinate system for triangle 1.

For an orthotropic material, the parameters are specified in the quad coordinate system.

The normal pressures act in the local z-direction.

End input with a blank line.

On input, created arrays are:
.min4t_mp(m,14) -> Ei, Gij, nuij, thickness, mass, local, C_s, iso
.min4t_el(6,n) -> node1 - node4, material #, print code
.min4t_node5(3,n) -> x, y, z coordinates of "node 5"
.min4t_q(5,n) -> pat, q1, q2, q3, q4
.min4t_ver(1) -> version#
.min4t_const(1) -> constraint#

During stiffness calculation, the shear relaxation factors are stored in .min4t_rlx(4*n) for later use in calculating the element state.

If a nonzero mass density is specified, uniform (gravitational) body forces are applied if the vector gravity(4) has been defined. The 4 components of gravity are: load pattern number, gx, gy, and gz, where gi is the gravitational acceleration in the global i direction.

For state calculation, global coordinates and bending stress resultants are stored in .min4t_stb(5*n,8) in the order x, y, z, Mx, My, Mxy, Qx, and Qy. These values are calculated at the triangle centroids. Each 5th row is reserved for "optimal"stresses, which are the average of the 4 triangle centroidal values and are located at the cross diagonal. If the value of local for the material is 1, then the bending resultants are calculated in the local triangle coordinate systems. The global coordinates and the membrane stress resultants at the 2x2 Gauss points are stored in .min4t_stm(n,24) in the order x, y, z, Nx, Ny, Nxy. The membrane resultants are always in quad coordinates.

The element does not calculate the equivalent nodal forces in equilibrium with its stress state, and therefore cannot be used in a nonlinear analysis.

The response option has not been implemented.

The error estimation option has not been implemented.

For state output, the stress resultants in .min4t_stb and .min4t_stm are printed.

For the theory of min3s, see Tessler, A., "A C0 Anisoparametric Three-Node Shallow Shell Element," Computer Methods in Applied Mechanics and Engineering, v. 78, 1990, pp. 89-103.

See Also

min3s min5s pmin4t pstate

pmin4t

Command Syntax

```
pmin4t
  Print min4t element data
```

See Also

min4t

min6

6-node, 18-DOF, linear, triangular Mindlin plate bending element.
Implementation assumes element is in the X-Y plane.

Command Syntax

```
min6  m=?  n=?  [q=q_vec]
m=mat#  e=E1,E2,E3  g=G12,G23,G13  nu=nu12,nu23,nu13               &
         t=thickness [mass=mass]  [local=local] [C_s=C_s]
         [iso=iso]     (1 record/matl)
n=nel  mat=mat  nodes=node1,node2,node3,node4,node5,node6           &
        [print=print] [inc=inc1,inc2,inc3  gen=gen]                 &
        [inc_2d=inc1_2d,inc2_2d,inc3_2d  gen_2d=gen_2d              &
        inc_el=inc_el]  [pat=pat q=q1,q2,q3,q4,q5,q6]
```

m is the number of different materials
n is the number of elements
q_vec is the name of a vector in the database for which the ith element is the normal pressure at node i

```
mat# is the material number
E1, E2, E3 are the moduli of elasticity
G12, G23, G13 are the shear moduli
nu12, nu23, nu13 are the Poisson ratios
thickness is the element thickness
mass is the mass per unit volume
local = 0 -> stress resultants in quad coordinates (default)
C_s is the shear relaxation factor (default = 0.0)
iso  = 0 -> isotropic material (default)
     = 1 -> orthotropic material
```

```
nel is the element number
node1 thru node6 are node numbers
mat is the material number for the element
print .ne. 0 -> element results not printed
inc1, inc2 inc3 are node increments in a "linear sequence"
gen is the number of elements to generate in a sequence
inc1_2d, inc2_2d, inc3-2d are node increments between sequences
gen_2d is the number of linear sequences to generate
inc_el is the element increment between sequences
pat is the load pattern number for the normal pressures
q1,q2,q3,q4,q5, q6 are normal pressures for nodes (these values
override those defined by q_vec, if any.)
```

Nodes 1 to 3 are the corner nodes and are specified 3 node triangular elements. Nodes 1 to 3 define the counterclockwise. Nodes 4 to 6 are the midnodes.

A "linear sequence" of elements can be generated by specifying inc1, inc2, inc3, and gen. In a linear sequence, nodes 1, 2, and 4 are incremented by inc1; nodes 5, 6 are incremented by inc2; and nodes 3 is incremented by inc3. gen is the number of elements to generate, so a sequence will have gen+1 elements. To generate a 2D patch of elements, multiple sequences can be specified; inc1_2d, inc2_2d, and inc3_2d are used to increment the node numbers from one sequence to the next. gen_2d is the number of additional sequences. The element numbers in two successive sequences differ by inc_el (default = numgen+1).

End input with a blank line.

The global coordinate system is the same as local coordinate system for triangle.

For an orthotropic material, the parameters are specified in the global coordinate system.

The normal pressures act in the local z-direction.

On input, created arrays are:
.min6_mp(m,14) -> Ei, Gij, nuij, thickness, mass, local, C_s, iso
.min6_el(6,n) -> node1 - node6, material #, print code
.min6_q(7,n) -> pat, q1, q2, q3, q4, q5, q6

During stiffness calculation, the shear relaxation factors are stored in .min6_rlx(4*n) for later use in calculating the element state.

If a nonzero mass density is specified, uniform (gravitational) body forces are applied if the vector gravity(4) has been defined. The 4 components of gravity are: load pattern number, gx, gy, and gz, where gi is the gravitational acceleration in the global i direction.

For state calculation, global coordinates and bending stress resultants are stored in .min6_stb(n, 4*8) in the order x, y, z, Mx, My, Mxy, Qx, and Qy. These values are calculated at the 4 Gauss points of the triangle.

The element does not calculate the equivalent nodal forces in equilibrium with its stress state, and therefore cannot be used in a nonlinear analysis.

For response calculation, global coordinates and bending stress resultants at the nodes are stored in .min6_respb(n,6*8) in the order x, y, z, Mx, My, Mxy, Qx, and Qy.

The error estimation option has not been implemented.

For state output, the stress resultants in .min6_stb are printed.

For response output, the stress resultants in .min6_respb are printed.

MIN6 is an anisoparametric Mindlin plate bending element with a cubic variation of transverse displacement and quadratic variation for rotational displacements.

See Also

min3s min5s min4t pmin6 presponse pstate

pmin6

Command Syntax

```
pmin6
  Print min6 element data
```

See Also

min6

nbeam2d

Large displacement, elastic 2D beam element

```
nbeam2d m=? n=? [nl=?]  [-xy]  [-xz]  [-yz]
m=matl  e=emodulus  g=gmodulus  [mbar=density]  [mxy=mx,my]  [mI=mI]  &
        [wbar=wbar]  a=area  i=moi  [as=as]  &
        [Cm=cmx,cmy] [Cd=cdx,cdy] (1 record/material)
n=nel  mat=mat  nodes=node1,node2  [print=print]   [gen=gen  inc=inc]  &
       [tension=tension]  [-xy]  [-xz]  [-yz]
```

m is the number of different materials
n is the number of elements
nl = 1 -> linear stiffness
   = 2 -> nonlinear and geometric stiffness (default)
-xy or -xz or -yz specifies the plane in global coordinates

matl is the material number
emodulus is the modulus of elasticity
gmodulus is the shear modulus
density or mx,my is the mass/unit length
mI is mass moment of inertia (per unit length) in local coordinates
wbar is the weight density (per unit length)
area is the cross sectional area
moi is the moment of inertia
as is the shear area (0 -> shear deformation is ignored)
Cm are the effective added mass coefficients (see below)
Cd are the effective drag coefficients (see below)

nel is the element number
node1 and node2 are the node numbers
mat is the material number for the element
print .ne. 0, element results not printed
inc is the node increment used for generation
gen is the number of elements to generate
tension is the initial tension (for stiffness calculation only)

End input with a blank line.

The 2-D large displacement, elastic beam element assumes small strains. Hence, the forces are calculated as for a linear beam element, except that first the rigid body rotation is removed from the displacements.

The element must lie in a plane parallel to the global X-Y (default), X-Z, or Y-Z planes. The default for all elements is on the nbeam2d record; this can be overwritten on a member basis on the member input record. The element local z-axis is in the global Z, -X, and Z directions, respectively.

On input, created arrays are:

```
  .nbeam2d_mp(m,13) -> emodulus, gmodulus, wbar, area, moi, as, mx,
                       my, mI, cmx, cmy, cd, cy
  .nbeam2d_el(n,4)  -> node1, node2, material, print code
  .nbeam2d_len(n)   -> element length
  .nbeam2d_st(n,4)  -> Axial Force, V, M at node1, M at node2
  .nbeam2d_dir(n)   -> 1 -> X-Y; 2 -> X-Z; 3 -> Y-Z
```

mx and my are the mass densities per unit length in local coordinates. If density is specified, then mx=my=density. This element computes a lumped mass matrix in local coordinates. However, when transformed to global coordinates, it will no longer be diagonal unless mx = my or the element is parallel to one of the global axes. If neither of these conditions is met, then a global diagonal mass matrix should not be used. Note that for large displacements, the mass matrix will also need to be reformulated if mx and my are not the same, and even if the element were initially parallel to a global axis, it will not in general remain parallel.

If the weight density is specified, it always acts in the -Z (global) direction. Therefore, if it is used the element should be in the X-Z or the Y-Z planes. The weight is always applied if it is specified.

cmx and cmy are the effective mass coefficients for a "Morison" treatement of fluid loading. These are effectively densities per unit length in local coordinates and will typically be equal to 1/2*rho*Cm*D, where rho is the fluid density, Cm is the actual mass coefficient, and D is the "diameter". The program will multiply these by L/2 for a lumped formulation, where L is the original length. Similarly, cdx and cdy are effective damping coefficients. Note: both the added mass and drag terms are on the right-hand-side only. The mass term is multiplied by the fluid acceleration to obtain a load, and the drag term is multiplied by abs(v-u)(v-u) to obtain a quadratic drag loading. The component of the added mass that is multiplied by the structure acceleration is included by specifying structure mass densities (mx,my) that include the added mass; the user is responsible for providing these modified mass densities.

For state calculation, element forces are put in .nbeam2d_st.

For response calculation, element does nothing.

For state output, results in .nbeam2d_st are printed.

For response output, no results are printed.

See Also

pnbeam2d   pstate   presponse

pnbeam2d

Command Syntax

```
pnbeam2d
  Print nbeam2d element data
```

See Also

nbeam2d

pbridge

Elastic, 3-D pontoon bridge element

Command Syntax

```
  pbridge s=? n=?  [maxpts=?]  [-kg]  gravity=?  rho=?  kvisc=?         &
          depth=?  [option=?]
  s=section e=emodulus g=gmodulus mbar=density w=weight a=area j=jsect  &
      iy=iyy iz=izz asy=asy asz=asz  road_width=? road_depth=? KC=? KG=? &
      [Cp=?] [Cptable=?] #pts=? [p1=?,? p2=?,? p3=?,?] (1 record/section)
  n=nel nodes=node1,node2 mat=mat print=print inc=inc gen=gen           &
        node3=node3 ndiv=ndiv tension=tension
```

s is the number of different module sections
n is the number of elements
maxpts is the maximum number of points to define the module cross sections (default=1)
-kg is a flag to include geometric stiffness
gravity is the acceleration of gravity
rho is the mass density of water
kvisc is the kinematic viscosity of the water
depth is the water depth
option determines the method to calculate drag
   = 0 -> viscous form drag only (default)
   = 1 -> viscous form drag + venturi lift

section is the section number
emodulus is the modulus of elasticity
gmodulus is the shear modulus
density is the mass/unit length
weight is the in-air weight/unit length
area is the cross sectional area
jsect is the torsional inertia
iyy,izz are moments of inertia in local coordinates
asy, asz are the shear areas in y and z, respectively (0 -> the corresponding shear deformation is ignored)
road_width is the total width of the roadway
road_depth is the total depth of the roadway
KC is the vertical distance from the keel to the elastic axis
KG is the vertical distance from the keel to the center of weight
Cp is the pressure drag coefficient
Cptable is the number of the table with Cp
#pts is the number of points to define the section geometry
pj is jth of #pts coordinate pairs in the form y,z (see below)

nel is the element number
node1 and node2 are the node numbers
mat is the section/material number for the element
print .ne. 0, element results not printed
inc is the node increment used for generation
gen is the number of elements to generate
node3 lies in the local x-z plane
tension is the inital tension (for geometric stiffness only)
ndiv is the number of divisions at which the internal forces are calculated in each element (default=4)

Either a constant drag coefficient, Cp, can be specified, or a table of Cp values, as a function of draft and trim, can be specified. In the latter case, the table number is specified here, and the table itself is specified with the cp_tables command.

For hydrostatic calculations, the cross section is assumed to be a polygon. To define the polygon only, local y-z axes are defined such that z is parallel to the global Z axis and z=0 corresponds to the bottom of the roadway. The cross section is assumed to be symmetric about z. The cross section consists of a rectangular middle section and two "bow" sections on either side. The rectangular middle section is defined by the road_width and road_depth. The "right" bow section, when the local y and z axes are directed to the right and up, respectively, is defined by the #pts pj. (#pts cannot be greater than maxpts.) The points are specified counterclockwise. It is not necessary to specify the two end points corresponding to the end of the roadway section, because these two points are generated automatically. The program reflects the cross section about the z-axis to generate the other half. Hence, for a given section, a total of 2*(#pts+2) points are used to define completely the polygon cross section.

The initial local (principal) axes of the cross section are defined as follows:

```
    The local x-axis is directed from node1 to node2
    The local y-axis = (x-axis) X (vector from node1 to node3)
    The local z-axis = (x-axis) X (y-axis)
```

If node3 is -1, -2, or -3, then the "vector to node3" is a unit vector in the direction of the negative X, Y, or Z global axes, respectively.

End input with a blank line.

On input, created arrays are:

```
  .pbridge_mp(s,17) -> emodulus, gmodulus, density, area, jsect,
                       iyy, izz, asy, asz, weight, KC, KG, Cp,
                       road_width, road_depth, #pts, Cptable
  .pbridge_sec(mpts,2,s) -> xi, yi for each section; mpts=2*(maxpts+2)
  .pbridge_el(6,n) -> node1, node2, section, print code, node3, ndiv
  .pbridge_len(n)  -> element length
  .pbridge_st(n,12)-> Axial Force, Vy, Vz, Torque, My, Mz at node1
                      Axial Force, Vy, Vz, Torque, My, Mz at node2
  .pbridge_kg      -> 0 or 1; w/o or w/ geometric stiffness
```

This element calculates a lumped mass matrix with zero rotational inertia.

For state calculation, element forces are put in .pbridge_st.

For response calculation, element does nothing.

For state output, results in .pbridge_st are printed, but using beam sign convention (torque at nodej is positive in the local x-axis).

For response output, results are printed for ndiv sections. If the unformatted write options on the presponse command are specified, the results are written to the file project_name.pbr. The data is written: element #, ndiv+1, and for each output section: axial force, Vy, Vz, torque, My, Mz.

This is a special-purpose 3-D beam element designed to model floating pontoon bridges. It assumes small strains and small rotations. However, it includes the geometric stiffness, hydrostatic stiffness/forces, and drag, and these quantities can change as a result of displacement. Hence, it incorporates some nonlinear behavior, and an iterative solution is usually required.

The element assumes:

1. The still-water-plane is at (global) z=0
2. Gravity acts in the negative Z direction
3. The initial Z-coordinates of the modules should be determined based on the weight and the buoyancy.

See Also

cp_tables ppbridge pstate presponse

ppbridge

Command Syntax

```
ppbridge
  Print pbridge element data
```

See Also

pbridge

ntruss

Two node large displacement truss element

Command Syntax

```
ntruss m=? n=? [nl=?]
m=matl e=emodulus a=area [mbar=density] [-no_compression]  (1 rec/matl)
n=nel  nodes=node1,node2  mat=mat  [print=print]              &
       [inc=inc]  [gen=gen]  [tension=tension]  [Lo=length]   &
       [ecc1=delta_x,delta_y,delta_z] [ecc2=delta_x,delta_y,delta_z]
```

```
m is the number of different materials
n is the number of elements
nl = 1 -> linear stiffness
   = 2 -> nonlinear and geometric stiffness (default)
```

matl is the material number
emodulus is the modulus of elasticity
area is the cross sectional area
density is the mass/unit length
-no_compression indicates tension-only material

nel is the element number
node1 and node2 are the node numbers
mat is the material number for the element
print .ne. 0, element results not printed
inc is the node increment used for generation
gen is the number of elements to generate
tension is the inital tension
length is the unstretched length
ecc1 are the offsets, in global coordinates, of the element start from node1

ecc2 are the offsets, in global coordinates, of the element end from node2

End input with a blank line.

The tension specified by tension= is only used to provide an initial stiffness to stabilize a slack initial configuration. The value is not used subsequently. The length specified by Lo= is the unstretched length (corresponding to zero force). The default unstretched length is the initial distance between the two ends. If Lo is specified, then any value specified for tension is not used; rather the initial tension is calculated.

If -no_compression is specified for a material, then the corresponding elements can only resist tension.

On input, created arrays are:

```
  .ntruss_mp(m,4)  -> modulus, area, density, compression code
  .ntruss_el(4,n)  -> node1, node2, material, print code
  .ntruss_len(n)   -> unstretched length
  .ntruss_st(n)    -> element force
  .ntruss_ecc(n)   -> eccentricity code
  .ntruss_ecc2(6,n)-> eccentricities (delta_x,delta_y,delta_z)
```

The element calculates a diagonal mass matrix only.
For state calculation, element forces are put in .ntruss_st.
For response calculation, element does nothing.
For state output, the axial forces in .ntruss_st are printed.

For response output, the element state is printed. If the unformatted write options on the presponse command are specified, the results are written to the file project_name.ntr. The data is written: element #, axial force.

constraints associated with an offset of the element end with the corresponding node are based on linear kinematics. Specifically, for nonzero offsets, the element end is 'slaved' to the node, and the displacements at the element end are determined based on rigid body, linear kinematics.

See Also

pntruss   pstate   presponse

pntruss

Command Syntax

pntruss

Print ntruss element data

See Also

ntruss

smth1c

One-dimensional, cubic smoothing element

Implementation assumes element is parallel to X-axis

Command Syntax

```
smth1c n=?
n=nel  nodes=node1,node2  mat=mat  [print=print]  [inc=incr  gen=gen]
```

n = number of elements

nel is the element number
node1 and node2 are the node numbers
mat is not used for this element
print .ne. 0, element results not printed
inc is the node increment used for generation
gen is the number of elements to generate.

End input with a blank line.

On input, created arrays are:

.smth1c_el(n,4) -> node1, node2, material, print code
.smth1c_ls(n,10) -> length, x-coord, stress, stress gradient for local coordinates -1, 0, 1

For stiffness calculation, data points are expected in array smth1c_in(#pts,2) -> x-coord, stress. The vector .smth1c_indx1(#pts) is created that matches data points with elements.

For state calculation, results are put in .smth1c_ls.

For response calculation, position, stress and stress gradient are calculated for local coordinates (-1 to 1) in smth1c_lc, if it exists, or for global coordinates in smth1c_gc. Results are put in .smth1c_resp. If .smth1c_gc exists, vector smth1c_indx2 is created with element number corresponding to each data point.

For response output, results in .smth1c_ls are printed.

For state output, results in .smth1c_resp are printed.

See Also
psmth1c   pstate   presponse

psmth1c

Command Syntax

```
psmth1c
```

Print smth1c element data

See Also
smth1c

smth1l

One-dimensional, 2 to 4 node, discrete least squares smoothing element. Implementation assumes element is parallel to the X-axis.

Command Syntax

```
smth1l n=?
n=nel  nodes=node1,node2,...,node4  mat=mat  [print=print]  &
    [inc=inc gen=gen]
    n is the number of elements

    nel is the element number
    node1 thru node4 are node numbers (2,3 or 4 nodes)
    mat is the material number for the element
    print .ne. 0 -> element results not printed
    inc is node increment for generation
    gen is the number of elements to generate

  End input with a blank line.

  On input, created arrays are:
    .smth1l_el(n,7) -> node1 - node4, material, print code, #nodes
    .smth1l_len(n)  -> element length

  For stiffness calculation, coordinates and variables to be smoothed are
  expected in array .smth1l_in.

  For state calculation, coordinate and stresses are put in .smth1l_st (NOT
  IMPLEMENTED).

  For response calculation, position, stress and stress gradient are
  calculated for local coordinates (-1 to 1) in smth1l_lc, if it exists, or
  for global coordinates in smth1q_gc. Results are put in
  .smth1l_resp(*,3).

  For state output, results in .smth1l_st are printed.

  For response output, results in .smth1l_resp are printed.

See Also
  psmth1l   pstate   presponse
```

psmth1l

```
Command Syntax
  psmth1l
    Print smth1l element data
See Also
  smth1l
```

smth1q

```
One-dimensional, quadratic smoothing element
    Implementation assumes element is parallel to the X-axis

Command Syntax
  smth1q m=? n=?
  m=mat  l=lambda  norm=norm  pen_norm=pen_norm  (1 record/material)
  n=nel  nodes=node1,node2  mat=mat  print=prin  [inc=inc  gen=gen]

    m = number of different "materials" (lamda values)
    n = number of elements

    mat is the material number
    lambda is the penalty parameter
    norm specifies normalization of the error term
       =0 -> no normalization
```

```
         =1 -> normalize by # data points in the element
         >1 -> normalize by the value specified
      pen_norm specifies normalization of the penalty term
         =0 -> no normalization
         >1 -> normalize by the value specified

      nel is the element number
      node1 and node2 are the node numbers
      mat is the material number for the element
      print .ne. 0, element results not printed
      incr is the node increment used for generation
      gen is the number of elements to generate.

    End input with a blank line.

    Arrays created on input:
      .smth1q_mp(m,3) -> lamdba, norm, and pen_norm
      .smth1q_el(n,4) -> node1, node2, material, print code
      .smth1q_ls(n,8) -> length, and X-coord, stress for local coordinates
                         -1, 0, 1. Last entry is difference between
                          derivative and slope.

    For stiffness calculation, data points are expected in array
    smth1q_in(#pts,2) -> X-coord, stress. The vector .smth1q_indx1(#pts) is
    created that matches data points with elements.

    For state calculation, results are put in .smth1q_ls.

    For response calculation, position, stress, and stress gradient are
    calculated for local coordinates (-1 to 1) in smth1q_lc, if it exists, or
    for global coordinates in smth1q_gc. If smth1q_gc exists, vector
    .smth1q_indx2 is created with element number corresponding to each point.
    Results are put in .smth1q_resp.

    For state output, results in .smth1q_ls are printed.

    For response output, results in .smth1q_resp are printed.

  See Also
    psmth1q    pstate    presponse
```

psmth1q

```
  Command Syntax
    psmth1q
      Print smth1q element data

  See Also
    smth1q
```

smth2l

```
  2-D, 3 to 9 node, discrete least squares smoothing element.
    Implementation assumes element is in the X-Y plane.

  Command Syntax
    smth2l n=? [-value]
    n=nel  nodes=node1,node2,...,node 9  [print=print]  [gauss=gauss]    &
```

```
[inc=inc1,inc2,inc3  gen=gen]                                    &
    [inc_2d=inc1_2d,inc2_2d,inc3_2d  gen_2d=gen_2d  inc_el=inc_el]
```

```
  n is the number of elements
  -value is a response code, as explained below

  nel is the element number
  node1 thru node9 are node numbers (3 to 9 nodes)
  mat is useless for this element
  print .ne. 0 -> element results not printed
  gauss is useless for this element
  inc1, inc2, inc3 are node increments in a "linear sequence"
  gen is the number of elements to generate in a sequence
  inc1_2d, inc2_2d, inc3_2d are node increments between sequences
  gen_2d is the number of linear sequences to generate
  inc_el is the element increment between sequences
```

This data line is identical with that of d2l3to9 command.

Nodes 1 to 4 are the corner nodes for quad elements and are specified counterclockwise. Nodes 5 to 8 are the midnodes on the edges (see sketch below), while node 9 is the center node. For triangular elements, only the first three nodes are to be specified.

Element generation is done by specifying inc1, inc2 & inc3. Nodes 1, 2, and 5 are incremented by inc1. Nodes 6, 8, and 9 are incremented by inc2. Nodes 3, 4, and 7 are incremented by inc3.

End input with a blank line.

On input, created arrays are:

```
  .smth2l_el(10,n) -> node1 - node9, print
  .smth2l_rtype    -> 0 if -value is not specified; 1 if specified
```

For stiffness calculation, data points are expected in smth2l_in(#pts,numvar+2) -> X, Y coordinates, numvar variables. Each variable (column of values) is smoothed separately, except they must be defined at the same X,Y coordinates. The vector .smth2l_indx(#pts) is created with the element number each X,Y pair falls in. In this description, it is assumed that the variables are stresses, although they may be anything.

For state calculation, X-Y coordinate and stresses at 9 nodal points (at local coordinates xi=-1, 0, +1 and eta=-1, 0, 1) are put in .smth2l_st(node#,numvar+2).

The response calculation depends on the value of .smth2l_rtype. If it is 0, X,Y coordinates, stress, and stress derivatives are calculated for local coordinates specified in smth2l_lc(*,2), if it exists, or for global coordinates in smth2l_gc(*,2). Results are put in smth2q_resp(*,5). If smth2l_gc exists, smth2l_index(*) is created with element number corresponding to each data point. This is done only for the first variable in smth2l_in. If _rtype is 1, then the smoothed values for all variables in smth2l_in are put in .smth2l_resp(*,numvar+2) but derivative information is not calculated.

For state output, results in .smth2l_st are printed.

For response output, results in .smth2l_resp are printed.

```
         node4               node7                 node3
           X---------------------X---------------------X
           |                                           |
           |                                           |
           |                                           |
           |                                           |
           |                                           |
           X node8               X                     X node6
           |                   node9                   |
           |                                           |
           |                                           |
           |                                           |
           |                                           |
           X---------------------X---------------------X
         node1                  node5                node2
```

See Also
psmth2l pstate presponse d2l3to9

psmth2l

Command Syntax

```
psmth2l
  Print smth2l element data
```

See Also
smth2l

smth2q

2-D, quadratic, triangular smoothing element. Implementation assumes that element is in the XY plane.

Command Syntax

```
  smth2q m=? n=?  [-value]  [-quad]
  m=mat#  lambda=lambda  option=option  norm=norm                 &
      pen_norm=pen_norm  tol=tol  lambda2=lambda2
      alpha=?  eps=?  Ao=?  gamma=?
  n=nel  nodes=node1,node2,node3,node4  mat=mat  [print=print]     &
      [inc=inc1,inc2 gen=gen]
[inc_2d=inc1_2d,inc2_2d  gen_2d=gen_2d  inc_el=inc_el]
```

m is the number of different materials
n is the number of elements
-value is a response code, as explained below
-quad indicates quadrilateral rather than element numbers are specified; see note below.

mat# is the material number
lambda is the first penalty parameter
option for the penalty stiffness term
      = 1 -> use discrete edge constraints (default)
      = 2 -> use integrated form
norm specifies normalization of the error term

= 0 -> no normalization
= 1 -> normalize by # data points in the element
> 1 -> normalize by the value specified
pen_norm specifies normalization of the penalty term
= 0 -> no normalization
> 1 -> normalize by the value specified
tol is a tolerance to map global to local coordinates
lambda2 is the penalty parameter on the curvature term
alpha, eps, Ao, and gamma define discrete weights, as explained below.

nel is the element number
node1 thru node4 are node numbers
mat is the material number for the element
print .ne. 0 -> element results not printed
inc1, inc2 are node increments in a "linear sequence"
gen is the number of elements to generate in a sequence
inc1_2d and inc2_2d are node increments between sequences
gen_2d is the number of linear sequences to generate
inc_el is the element increment between sequences

Lambda = 100 is suggested for this element.

If -quad is specified, all "element" numbers (n, nel, inc_el) refer to a quadrilateral number instead. Quad "i" is discretized into elements 4*i-1 to 4*i. This option is given so that the element input data for d2l3to9 can be used directly to develop a smoothing mesh.

The values alpha (default=0), eps (default=0), Ao (default=0), and gamma (default=2) are used to define the weight w_q on the discrete square error term for the qth data point. If Ao > 0, then

```
                exp(-alpha * A_bar^gamma)
          w_q = ---------------------
                   A_bar^gamma + eps
```

where A_bar is the area of the smoothing element for point q divided by Ao. This form of w_q weights data points associated with smaller elements more than data points associated with larger elements. Ao should be specified to be a "nominal" element area. If the element area = Ao, w_q = 1. If Ao is less than or equal to 0, w_q = 1.

Three Nodes Specified

Nodes 1 to 3 are the vertex nodes for the element and are specified counterclockwise.

A "linear sequence" of elements can be generated by specifying inc1, inc2, and gen. In a linear sequence, nodes 1 and 2 are incremented by inc1, and node 3 is incremented by inc2. gen is the number of elements to generate, so a sequence will have gen+1 elements. Multiple sequences can be generated: inc1_2d and inc2_2d are used to increment the node numbers from one sequence to the next. gen_2d is the number of additional sequences. The element numbers in two successive sequences differ by inc_el (default = numgen+1).

Four Nodes Specified

The nodes are the four corner nodes of a general quadrilateral and are specified counterclockwise. The quad is meshed with 4 triangular elements in a cross-diagonal pattern (recommended mesh for this element).The coordinates of the fifth (internal) node are calculated. If a node exists at these coordinates, it is used for node 5, otherwise a new node is created. This option then creates 4 triangular elements (nodes 1-2-5, 2-3-5, 3-4-5, and 4-1-5) numbered from nel to nel+3.

A "linear sequence" of elements can be generated by specifying inc1, inc2, and gen. In a linear sequence, nodes 1 and 2 are incremented by inc1, and nodes 3 and 4 are incremented by inc2. gen is the number of quads to generate, so a sequence will have 4*(gen+1) triangular elements. Multiple sequences can be generated: inc1_2d and inc2_2d are used to increment the node numbers from one sequence to the next. gen_2d is the number of additional sequences. The element numbers in two successive sequences differ by inc_el (default = 4*(numgen+1)).

End input with a blank line.

On input, created arrays are:

```
  .smth2q_mp(m,6) -> lambda, option, norm, pen_norm, tol (def = 1.e-8),
                     lambda2
  .smth2q_el(5,n) -> node1 - node3, material #, print code
  .smth2q_rtype   -> 0 if -value is not specified; 1 if specified
```

For stiffness calculation, data points are expected in smth2q_in(#pts,numvar+2) -> X, Y coordinates, numvar variables. Each variable (column of values) is smoothed separately, except they must be defined at the same X,Y coordinates. The vector .smth2q_indx(#pts) is created with the element number each X,Y pair falls in. In this description, it is assumed that the variables are stresses, although they may be anything.

For state calculation, X,Y coordinates, stress, stress derivatives and the slopes at the element centroids are put in .smth2q_st(n,7). This is done only for the first variable (column 3) in smth2q_in.

The response calculation depends on the value of .smth2q_rtype. If it is 0, X,Y coordinates, stress, stress derivatives and the slopes are calculated for local (area) coordinates specified in smth2q_lc(*,3), if it exists, or for global coordinates in smth2q_gc(*,2). Results are put in smth2q_resp(*,7). If smth2q_gc exists, smth2q_indx2(*) is created with element number corresponding to each data point. This is done only for the first variable in smth2q_in. If _rtype is 1, then the smoothed values for all variables in smth2q_in are put in .smth2q_resp(*,numvar+2) but derivative information is not calculated.

For state output, results in .smth2q_st are printed.

For response output, results in .smth2q_resp are printed.

This element is used to smooth stresses over a two-dimensional field based on discrete least squares. The triangular element has 3 DOFs/node: 1 stress and 2 "slopes". The stress and slopes are interpolated independently; the penalty parameter lambda is used to enforce compatibility between the slopes and the derivatives of the stress. For details, see Tessler, A., Riggs, H.R., and Macy, S.C., "Application of a Variational Method for Computing Smooth Stresses, Stress Gradients, and

Error Estimation in Finite Element Analysis," MAFELAP VIII, Brunel University, England, April 27-30, 1993.

See Also

psmth2q state response

psmth2q

Command Syntax

```
psmth2
  Print smth2q element data
```

See Also

smth2q

smthspr

2-D, 3 to 9 node SPR smoothing element.

Implementation assumes element is in the X-Y plane.

Command Syntax

```
smthspr n=? [-value]  [vtex_num=?]  r=1,x,y,x^2,xy,y^2,x^2y,xy^2,x^2y^2
n=nel  nodes=node1,node2,...,node9  [print=print]      &
    [gauss=gauss]  [inc=inc1,inc2,inc3 gen=gen]        &
    [inc_2d=inc1_2d,inc2_2d,inc3_2d  gen_2d=gen_2d  inc_el=inc_el]
```

n is the number of elements
-value is a response code (options not implemented)
vtex_num is the minimum # elements connected to a vertex node (default=3)
r= identifies which terms in the polynomial to include:
  0 -> do not include the term
  1 -> include the term

nel is the element number
node1 thru node9 are node numbers (3 to 9 nodes)
mat is unused (see below)
print .ne. 0 -> element results not printed
gauss is unused (see below)
inc1, inc2, inc3 are node increments in a "linear sequence"
gen is the number of elements to generate in a sequence
inc1_2d, inc2_2d, inc3_2d are node increments between sequences
gen_2d is the number of linear sequences to generate
inc_el is the element increment between sequences

mat and gauss are included here so that the input is identical to the input for element d2l3to9.

Nodes 1 to 4 are the corner nodes for quad elements and are specified counterclockwise. Nodes 5 to 8 are the midnodes on the edges (see sketch below), while node 9 is the center node. For triangular elements, only the first three nodes are to be specified.

Element generation is done by specifying inc1, inc2 & inc3. Nodes 1, 2, and 5 are incremented by inc1. Nodes 6, 8, and 9 are incremented by inc2. Nodes 3, 4, and 7 are incremented by inc3.

End input with a blank line.

On input, created arrays are:

```
  .smthspr_el(n,10) -> node1 - node9, print code
  .smthspr_rtype    -> set to 1 at present
  .bcid(10,#nodes)  -> see command bcid for explanation
```

For stiffness calculation, data points are expected in smthspr_in(#pts,numvar+2) -> X, Y coordinates, numvar variables. Each variable (column of values) is smoothed separately, except they must be defined at the same X,Y coordinates. The vector .smthspr_indx(#pts) is created with the element number each X,Y pair falls in. In this description, it is assumed that the variables are stresses, although they may be anything.

For state calculation, X-Y coordinate and stresses at each nodal point are put in .smthspr_st(#nodes,numvar+2). The stresses are in the global coordinate system and stresses are stored in the same order as in smthspr_in. Nodal stresses are copied to .disp(#nodes,numvar).

For response calculation, global coordinates and stresses are calculated for local coordinates in smthspr_lc(#pts,2). Nodal stresses are expected in .disp that are calculated in state. Hence, state must be called before response. Results are put in .smthspr_resp(#pts*n,numvar+2). See the explanation for state calculation for the order of stresses. If instead global coordinates are given in smthspr_gc(#pts,2), then results for those points are put in .smthspr_resp(#pts,numvar+2). The vector .smthspr_indx2(#pts) maps the data points to the element in which it falls.

For state output, results in .smthspr_st are printed.

For response output, results in .smthspr_resp are printed.

```
   node4                 node7                 node3
     X---------------------X---------------------X
     |                                           |
     |                                           |
     |                                           |
     |                                           |
     |                                           |
     X node8               X                     X node6
     |                   node9                   |
     |                                           |
     |                                           |
     |                                           |
     |                                           |
     X---------------------X---------------------X
   node1                  node5                node2
```

For the theory see Zienkiewicz, O.C. and Zhu, J.Z., "The Superconvergent Patch Recovery and A Posteriori Error Estimates. Part 1: The Recovery Technique," Intl. J. Num. Meth. Engrg., v. 33, 1992, pp. 1331-1364.

See Also
  psmthspr   presponse   pstate

psmthspr

```
    Command Syntax
      psmthspr
        Print smthspr element data

    See Also
      smthspr
```

spring

```
    Nonlinear, elastic spring element

              ---------- OPTION 1 ----------
    Command Syntax
      spring m=? n=?
      m=mat  e=Ko,Fyt,Fyc,Kpt,Kpc      (1 record/material)
      n=nel  type=type  nodes=node1,node2  mat=mat  [print=print]  l=lx,ly,lz

        m is the number of different materials
        n is the number of elements

        mat is the material number
        Ko is the initial spring stiffness
        Fyt is the tensile yield strength
        Fyc is the compressive yield strength (typically negative)
        Kpt is the post-yield stiffness in tension
        Kpc is the post-yield stiffness in compression

      End input with a blank line.

              ---------- OPTION 2 ----------
    Command Syntax
      spring m=? n=?
      m=mat  fv=d1,f1,d2,f2,etc        (1 record/material)
      n=nel  type=type  nodes=node1,node2  mat=mat  [print=print]  l=lx,ly,lz

        m is the number of different materials
        n is the number of elements

        mat is the material number
        fv is a sequence of deformation,force (d,f) defining the
        deformation-force curve for the spring. The d1, d2, etc must be in
        ascending order. A minimum of 2 points must be specified, and a maximum
        of 10 points may be specified.

      End input with a blank line.

              -------- BOTH OPTIONS --------
        nel is the element number
        type is the specifier for spring type
             =  1 -> translational spring
             =  2 -> rotational spring
        node1 and node2 are the node numbers
        mat is the material number
        print .ne. 0 -> element printout suppressed
        lx, ly, lz are direction cosines that specify the element orientation
```

Option 1 has a bilinear force-deformation relation (bilinear for each of the positive and negative deformation regimes. Option 2 is multilinear, with 10 datapoints for the whole spectrum (positive and negative spectrum). Option 1 is kept for backward compatibility. For option 2, if the calculated deformation is beyond the range specified, extrapolation is used.

If the spring is connected to "ground", leave node2 blank. If node2 is specified, the spring can be used to resist relative translation or rotation of nodes, whether they are separated or co-located.

For a translational spring, node2 is ignored (it is taken as "ground"). For a rotational spring, both node1 and node2 are used, and hence can be used to resist relative rotation of two otherwise identical nodes.

The direction cosines lx, ly, lz are with respect to the global coordinate system, and define the element orientation in the following sense. A grounded translational spring in the positive x-direction has a direction vector of 1, 0, 0. A grounded rotational spring with direction vector 1, 0, 0 means it resists rotation about the x-axis, with the right-hand-rule used to determine positive rotation. If the direction vector is not one of the global axes, the deformation of the element is calculated by taking the dot product of the displacement vector and the direction vector. If the spring connects two nodes, the direction is defined by the order of the nodes and the direction vector, not by the vector between the two nodes (which may be zero). Specifically, the relevant displacements are defined by the direction vector, and then the deformation is obtained by subtracting the displacement of node2 from node1. Note that this may be opposite from what might normally be expected. In both cases it can be imagined that node1 is at the head of the direction vector and node2, or ground, is at the tail.

On input, created arrays are:

```
  .spring_mp(m,5) -> Ko, Fyt, Fyc, Kpt, Kpc
  or
  .spring_mp(m,21) -> d1,f1...d10,f10, number of actual points
  .spring_el(n,5) -> spring type, node1, node2, material, print
  .spring_dr(n,3) -> lx, ly, lz
  .spring_st(n,2) -> element force and deformation
```

For state calculation, element forces are put in .spring_st.

For response calculation, element does nothing.

For state output, results in .spring_st are printed.

For response output, no results are printed.

See Also

pspring   pstate   presponse

pspring

Command Syntax

pspring

Print spring element data

```
    See Also
      spring
```

stiff2n

```
    Linear elastic 2-node stiffness element

    Command Syntax
      stiff2n n=?
      n=nel  nodes=node1,node2  k=stiff__name  [print=?]

        n is the number of elements

        nel is the element number
        node1 and node2 are the node numbers
        stiff_name is the name of the pre-defined 6x6 stiffness matrix
        print .ne. 0 -> element printout suppressed

      End input with a blank line.

      This element allows a 6x6 stiffness matrix to be used to connect two
      nodes. It may find most usefulness in connecting two nodes that are at
      identical locations. The "deformation" of the element is defined as the 6
      displacements of node2 minus the 6 displacements of node1. The element
      forces are then obtained by multiplying the deformation by the stiffness
      matrix. One use of the element is to specify a diagonal matrix, each
      diagonal with the spring stiffness connecting the two nodes.

      On input, created arrays are:
        .stiff2n_el(3,n) -> node1, node2, print code
        .stiff2n_k(32,n) -> names of the matrices (each up to 31 characters)
        .stiff2n_st(6,n) -> element force

      For state calculation, element forces are put in .stiff2n_st.

      For response calculation, element does nothing.

      For state output, results in .stiff2n_st are printed.

      For response output, no results are printed.

    See Also
      pstiff2n   pstate   presponse
```

pstiff2n

```
    Command Syntax
      pstiff2n
        Print 2-node stiffness element data

    See Also
      stiff2n
```

truss

```
    Two node linear truss element

    Command Syntax
```

```
truss m=? n=?  [-kg]
m=matl e=emodulus a=area [mbar=density]  (1 rec/matl)
n=nel  nodes=node1,node2  mat=mat  [print=print]              &
       [inc=inc]  [gen=gen]  [tension=tension]  [Lo=length]   &
       [ecc1=delta_x,delta_y,delta_z] [ecc2=delta_x,delta_y,delta_z]
```

m is the number of different materials
n is the number of elements
-kg is a flag to include geometric stiffness

matl is the material number
emodulus is the modulus of elasticity
area is the cross sectional area
density is the mass/unit length

nel is the element number
node1 and node2 are the node numbers
mat is the material number for the element
print .ne. 0, element results not printed
inc is the node increment used for generation
gen is the number of elements to generate
tension is the inital tension
length is the unstretched length
ecc1 are the offsets, in global coordinates, of the element start from node1
ecc2 are the offsets, in global coordinates, of the element end from node2

End input with a blank line.

This element is identical to the ntruss element with option nl=1 (linear stiffness), except this element also uses linear kinematics to determine the axial force. The tension specified by tension= is only used for a geometric stiffness matrix. The value is not used subsequently. The length specified by Lo= is the unstretched length (corresponding to zero force). The default unstretched length is the initial distance between the two ends. If Lo is specified, then any value specified for tension is not used; rather the initial tension is calculated.

On input, created arrays are:

```
 .truss_mp(m,4)  -> modulus, area, density, unused
 .truss_el(4,n)  -> node1, node2, material, print code
.truss_len(n)    -> unstretched length
.truss_st(n)     -> element force
.truss_ecc(n)    -> eccentricity code
.truss_ecc2(6,n)-> eccentricities (delta_x,delta_y,delta_z)
```

The element calculates a diagonal mass matrix only.
For state calculation, element forces are put in .truss_st.
For response calculation, element does nothing.
For state output, the axial forces in .truss_st are printed.

For response output, the element state is printed. If the unformatted write options on the presponse command are specified, the results are written to the file project_name.tr. The data is written: element #, axial force.

See Also

ntruss   ptruss   pstate   presponse

ptruss

Command Syntax

```
ptruss
  Print truss element data
```

See Also

truss

2.8. Miscellaneous Commands

fortran_kind

```
    Command Syntax
      fortran_kind
        Prints the number of bytes for standard data types (Absoft compiler).
```

gauss_int

```
    Command Syntax
      gauss_int arg1 arg2 arg3  n=?
        arg1 = function values at gauss points - (n*nel,1) matrix
        arg2 = element lengths - (nel,1) matrix
        arg3 = integration result/element - (nel,1) matrix
        n  = number of integration points (1 to 10)
```

This function numerically evaluates the integral of an arbitrary function over a one-dimensional domain. For two-dimensional domains, see the finite element d2l3to9, which can be used as well to integrate an arbitrary function as a special case of "error estimation".

See Also

```
      d2l3to9   fem_error
```

gauss_pts

```
    Command Syntax
      gauss_pts arg1  [arg2]  [n=?,?,?]
```

The array arg1 is created with the natural coordinates corresponding to n1 x n2 x n3 Gauss quadrature. The dimension of the space, and the number of columns of arg1, is specified by the number of parameters specified by n. That is, if only n1 is given, then arg1(n1) will be created; if n1 and n2 are specified, then arg1(n1*n2,2) will be created; etc. The maximum value of the n's is 10. If arg2 is given, the weights will be stored in it.

See Also

```
      gauss_int
```

poly

```
    Command Syntax
      poly  arg  d=?  [n=?]  [option=?]  c=?,?,?,...
```

Evaluate a polynomial defined by the coefficients specified by c=. arg is an (npts,q) array, where npts is the number of points at which to evaluate the function. If option=0 (default), then the function only will be evaluated; if option=1, then the function and the first derivatives will be evaluated.

One-Dimensional Polynomial (d=1):

An arbitrary degree 1-D polynomial can be defined. The number of coefficients is specified by n= (degree = n-1). The coefficients are specified in the order 1, x, x^2, etc. The first column of arg contains the X-coordinates at which to evaluate the function. The function values will be placed in the second column. If requested, the first derivatives will be put in the third column.

Two-Dimensional Polynomial (d=2):

A maximum fifth degree polynomial (21 coefficients) can be specified. The coefficients are given in the order of Pascal's triangle, i.e., 1, x, y, x^2, xy, y^2, etc. The X,Y coordinates are expected in the first two columns of arg. The function values will be put in the third column. If requested, the first partial derivatives wrt X and Y will be put in the 4th and 5th columns of arg, respectively.

In either case, if the coefficients have been defined previously (e.g., by the input command) use the form c=~array_name, where array_name is the array with the coefficients.

tri_intpts

Command Syntax

```
tri_intpts arg1  [arg2]  n=?
```

The array arg1(n,3) is created with the area coordinates corresponding to the n integration points for a triangular domain. If arg2 is given, the weights are put in it. The number of points may be 1, 3, 4, 7, 9, or 12.

See Also

gauss_pts

userf

Command Syntax

```
userf  f=?
```

User-defined functions. f specifies which function. The arguments and parameters depend on the function. Help is only provided if the developer of the function provides a help routine. This option should only be used for prototyping and routines of short term use. Often, help is not available for such routines.