Manual of the software package *maiprogs*
Version 3.7.5

Matthias Maischak*
Brunel University, Uxbridge, UK
Institut für Angewandte Mathematik, Universität Hannover

December 24, 2015

0 Overview

This manual covers the User Interface of the program system *maiprogs* for the solution of FEM and BEM problems in 1d,2d and 3d. The user interface is represented by a specialised Batch Control Language (BCL) used to express actions, definitions and to automate parameter studies.

*email: matthias.maischak@brunel.ac.uk

Contents

1 Introduction

The system of programs *maiprogs* consists currently of the programs *maifem1*, *maicoup2*, *maicoup3* and *maigraf*, which are actively developed, and the special purpose programs *maibem3*, *maifem3*, *maibem2*, *maifem2*, *maicont2* written in Fortran 95, which are only kept for comptability reasons. Additional programs are in preparation. The programs *maibem3*,*maifem3* and *maicoup3* are the implementations of the 3D-BEM on polyhedral boundaries, 3D-FEM on polyhedral domains and the coupling of 3D-BEM and 3D-FEM, respectively, The programs *maibem2*, *maifem2* and *maicoup2* are the implementations of the 2D-BEM on polygons, 2D-FEM on domains with polygonal boundaries and the coupling of 2D-BEM and 2D-FEM, respectively, whereas *maicont2* is the implementation for unilateral contact problems on polygonal boundaries. The program *maigraf* presents the results and meshes of *maibem3* on surfaces, *maifem3*, *maicoup3*, *maifem2*, *maibem2*, *maicoup2*, *maicont2*, *maifem3* and *maifem1*.

maifem1, *maicoup2* and *maicoup3* are now general purpose programs for 1d-, 2d- and 3d-problems.

Any new developments should be based on *maifem1*, *maicoup2* and *maicoup3*.

maibem3	3D-BEM on polyhedral boundaries
maifem3	3D-FEM on polyhedral domains
maicoup3	Coupling of 3D-BEM and 3D-FEM for polyhedral domains
maibem2	2D-BEM on polygonal boundaries
maifem2	2D-FEM in polygonal domains
maicoup2	Coupling of 2D-BEM and 2D-FEM for polygonal domains
maicont2	Signorini contact problems for polygonal domains
maifem1	1D-FEM
maigraf	Presentation and Postprocessing of data generated by the previous programs

All nine programs use the same user-interface, i.e. the same batch control language. They also share many source code files. This system of programs is intended to simplify numerical tests of the Galerkin BEM and FEM for integral equations of the first kind using various iterative solvers.

1.1 Installation

The system of programs *maiprogs* comes as a gzip'd tar-file *maiprogs.tgz*. Move it to a separate folder, usually called `maiprogs`, and install it by

```
gzip -c -d maiprogs.tgz|tar -xvf -
```

After the initial installation all later updates can be simply downloaded from the central maiprogs-server by

```
make update
```

The update comes as a *maiprogs.tgz*-file, which can be unpacked and installed in the folder `maiprogs` just by

```
make install
```

Now all programs can be compiled by the command `make`. In case of a System V make you have to use `make sysv` due to the restrictions of this make command. The use of GNU-make (make, gmake) or Sun-make is recommended. If you want only to compile one program you can give for example the command `make bem2` to compile all source files of *maibem2* and link them. In general you can compile all programs separately by `make name` by ommiting the preceeding 'mai'.

There are the folders and files

```
ads/ bench/ config doku/ examples/ fo11/ fo12/ fo22/ fo23/ fo33/ fo2c/ fo3c/
foc/ focuda/ fog/ fox/ lib/ makefile uni/
```

doku/ contains this file and some other files documenting the analytically computed integrals. The examples which are presented in [5] are located in bone/ (in a separate archive).

fo11/ contains the source files of *maifem1*, fo12/ contains the source files of *maibem2*, fo22/ the source files of *maifem2*, fo23/ the source files of *maibem3*, fo33/ the source files of *maifem3*, fo2c/ the source files of *maicoup2*, fo3c/ the source files of *maicoup3*, foc/ the source files of *maicont2*, focuda/ the shared CUDA-source files, fog/ the source files of *maigraf*, fox/ the source files which are shared by all or the most programs and lib/ contains the libraries with the subroutines computing the Galerkin elements. uni/ contains programs which are optimized for uniform meshes on simple geometries. bench/ contains a simple benchmark program (started by `./run`) which helps to estimate the performance of a system suitable for *maiprogs*. 'makefile' is the topmost Makefile whereas every folder contains its own Makefile which is called by the topmost Makefile. 'config' is the configuration script. ads/ contains source files to adapt the system to different libraries. ads/ also contains the files 'cshrc', 'kshrc' and 'bashrc' which contain alias definitions for calling the programs of the maiprogs family using the csh, tcsh, ksh or bash shell.

The integrals which are used to compute the Galerkin elements are proven in [1, 2, 3]. A description of the interface of the libraries and other useful subroutines can be found in the technical manual to *maiprogs* [4].

1.2 Configuration

'config' is a shell script which determines the operating system, the used FORTRAN compiler and which libraries are available. Using this informations 'config' sets the Compiler options and link paths which are necessary to compile the whole program system. Using the shell variable MYLIBPATH you can extend the search path used by 'config'. It decides also whether it is necessary to add some code to replace the NAG-library which is primarily used (this code is located in the directory ads/). In general 'config' tries to take advantage of any optimized available library, especially BLAS and LAPACK.

'config' generates the file 'ads/adsys.f90', which adapts *maiprogs* to time measuring and reading of command line arguments in use by compiler and operating system. Compiler flags, link paths, libraries, resp. replacement codes will be written to the files 'makefile.mai' and 'makefile.gen'.

The behavior of 'config' can be influenced by several shell variables (see below) and the command line options

-help, -h	print a help text
-debug,-d,-g	set the compiler debug option
-range, -C	check for index ranges
-version, -v	print the version information
-test, -t=...	ignore library ... =nag,blas,lapack,arpack,sperf, imsl,dxml,mkl,pgplot, X11,opengl
-compiler,-c=...	use compiler ...
-opt, -o=...	Optimization level: no,low,full; full is default
-openmp,-omp=...	Use Openmp: no,yes (default)
-mpi =...	Use MPI: yes,no (default)
-cuda =...	Use CUDA code: yes,no (default)
-make	Use old (existing) configuration
-mp =...	Use Multiprecision package ARPREC: yes,no (default)
-qp =...	Use quadruple precision: yes,no (default)

Permanent defaults for 'config' can be written into the file .configrc in the maiprogs-directory. .configrc is read by config on start-up before command line options are read. Useful shell variables are

`FC`	(compiler, default: empty),
`DEBUG`	(default: no),
`RANGECHECK`	(default: no),
`OLEVEL`	(Optimizations, default: full),
`OPENMP`	(use OpenMP, default: yes),
`MPI`	(use MPI, default: no),
`CUDA`	(use CUDA, default: no)
`MPMODE`	(compile with arprec, default: no),
`QPMODE`	(compile with quad precision, default: no),
`MYLIBPATH`	(local library search path, default: empty).

1.3 *maiprogs* start-up sequence

On start-up *maiprogs* will try to find the file 'mairc' in the directories $MAIPROGSHOME/ads (if the environment variable $MAIPROGSHOME is set), the 'installation directory'/ads and $HOME/maiprogs/ads (in this sequence). 'mairc' contains bcl-code to load pre-defined problem definitions from the file 'problems', pre-defined geometry definitions from the file 'geometries' and pre-defined error estimator definitions from the file 'estimators'. User-defined problems, geometries and error estimators can be added to the files 'myproblems', 'mygeometries' and 'myestimators'. This three files don't belong to maiprogs, i.e. an update will not overwrite them, but the start-up script 'mairc' has instructions to load them if they exist.

Additionally, 'mairc' will try to find a user-defined file 'myrc' and run it, if it exists. 'myrc' is not part of the *maiprogs* release and will not be overwritten by any update. This offers a way to add additional persistent local definitions without modifying the global files, which will be overwritten by updates. It is strongly recommended not to change the files 'mairc', 'problems', 'geometries' and 'estimators', but to put any modifications and extensions either into your own files loaded by 'myrc', or the recommended user-defined files 'myproblems', 'mygeometries' and 'myestimators'. Files which are loaded at a later stage will overwrite standard definitions with the same name.

Current version of 'mairc':

```
! maiprogs resource file, used for global configuration
! use myrc for personal configuration (loaded by this file)
```

```
getmode('debug',DEBUGRC); setmode('debug',0) ! switch off debug mode
setmode('blank',1) ! leading blanks are no comments

file=Locatefile('problems')
if ((file/=' ').and.(PROGNAME/='maigraf')); then
 load(file)
fi
file=Locatefile('estimators')
if ((file/=' ').and.(PROGNAME/='maigraf')); then
 load(file)
fi
file=Locatefile('geometries')
if (file/=' '); then
 load(file)
fi
file=Locatefile('myproblems')
if ((file/=' ').and.(PROGNAME/='maigraf')); then
 load(file)
fi
file=Locatefile('myestimators')
if ((file/=' ').and.(PROGNAME/='maigraf')); then
 load(file)
fi
file=Locatefile('mygeometries')
if (file/=' '); then
 load(file)
fi
file=Locatefile('myrc') ! find personalized configuration
if (file/=' '); then
 load(file)
fi

destroy(file)
setmode('debug',DEBUGRC); destroy(DEBUGRC) ! restate debug mode
end
```

2 The batch control language

2.1 The concept of this language

All programs are controlled by commands in a given file. The name of this file can be given as the first argument on the command line, otherwise the programs use a default name for accessing the file. The default names are *pltin* for *maibem3*, *recin* for *maibem2*, *femin* for *maifem2* and *maifem3*, *coup2in* for *maicoup2*, *coup3in* for *maicoup3*, *contin* for *maicont2* and *grain* for *maigraf*. The file is read line by line. Multiple commands on a line are separated by ';'. Lines can be continued by using an '&' as the last element on a line. If, optionally, the next line starts with blanks followed by '&', also the '&' will be removed.

All commands can have a list of arguments on the same line. Arguments can be integers, real numbers, complex numbers and strings ('...'). There are two styles for writing the arguments. In the first case the arguments are separated by blanks if not otherwise stated. If the command demands a single string the ' ' can be omitted. For most arguments there are

standard values given. The symbol '-' can be given at the position of an argument instead of a value if the standard value has to be used. If the last arguments are missing, the standard values are also used. In the second case, the argument list is enclosed in brackets (...) and the arguments are separated by ',' and the arguments can be named.

Variables consist of a sequence of letters and numbers followed, optionally, by an index enclosed in square brackets, e.g., 'VAR[23]'. Initialized variables can replace every number/string (integer, real, complex) in argument lists. Strings can also be constructed by concatenation using the symbol '//', for example the variable 'I' may have the value 12, then 'I= '//I//'.' gives the string 'I= 12.' . In string constructions as an argument by 'open', resp. the output of 'print' or 'write', numbers 'n' can be written in a field of length 'l' by the syntax 'n:l', i.e. in case of integers the leading space will be filled by blanks, whereas reals will be printed with the corresponding number of digits to fill the field of length 'l'.

Function names start with a captial letter followed by lower case letters.

Char()	Create string with ascii value as argument of Char()
Len()	Number of characters in argument
Len_trim()	Length of argument without trailing blanks
Trim()	Argument with out trailing blanks
Exp(), Log()	Exponential and logarithm
Sqrt()	Square root
Abs()	Absolut value
Int()	Integer part
Real(), Aimag()	Real part, imaginary part
Sin(), Cos(), Tan()	Trigonometric functions
Asin(), Acos(), Atan()	Inverse trigonometric functions

Expr denotes an arithmetical or logical expression. There are the arithmetical operators '$+, -, *, /, **$' and the logical operators '$\&\&, ||, =, <>, <=, <, >=, >$' and '.and., .or., .eq., .ne., .le., .lt., .ge., .gt.'. A logical false is represented by integer zero and a logical true by integer one. Any integer value unequal zero will be interpreted as a logical true in a logical expression.

Vector valued arguments are given by '(/..,...,.../)'.

2.2 Command line options

The following options are available.

-o file replaces the default for structured output by 'file'. Can be overridden by 'open(1)' (see below).

-o2 file replaces the default for unstructured output by 'file'. Can be overridden by 'open(2)' (see below).

-f file replaces the default for the input filename by 'file'. The '-f' can be omitted.

-d starts the program in debug mode. See 'debug' below.

-i starts the program in interactive mode. Now a leading '!!' will repeat the last command, whereas a leading '!str' will search for the last command beginning with 'str' which will be repeated.

-r starts the program in restricted mode, i.e. file creation and deletion is only allowed in subdirectories of the current working directory.

-v prints the name of the program and the version number of BCL.

2.3 Commands common to all programs

Historically BCL had a mode, where the simplest command was a line with a leading blank, which was interpreted as comments and ignored. This mode is now switched off by default.

The commands in common are implemented in the source code file 'bcl.f90' shared by all programs.

load(filename) executes the commands from the file given by 'filename' which can be a string expression. The old command '#in' is still supported for compatibility. The command 'load' can be nested at least two times.

open(chno) file opens a new output file with number 'chno'. Channel 1 is used by all commands which produce a structured output intented to be read by a program again. If there was a output file used before, the program appends an 'end' command and closes the file before the new file is opened. Channel 2 is used for unstructured output, which is not intended to be used by *maiprogs* again, such as output for LaTeX, Gnuplot or Xfig. The channel numbers 3 – 10 are available for own purposes.

The old commands '#out' and '#out2' for channel 1 resp. 2 and '#out(chno)' are still supported.

append(chno) file appends the output written to channel number 'chno' to the existing file 'file'.

write(chno) var/number/string does the same as print, but appends the output to channel 'chno'. Here channel 0 denotes the standard output. The old command 'write2' is still supported.

read(chno[,status-var]) var-list reads from channel 'chno' into the variables given by var-list, where the value of the variable 'status-var' indicates the end of file (unequal zero).

close(chno) closes the channel 'chno'.

end appends a 'end' command to the output file, if used, and returns back to the command execution from a previous input file. If there is no input file anymore the execution is stopped. The old commands '#e,quit' are still supported.

set var expression sets the variable 'var' to the value of 'expression', integer, real complex or string. The same action can be performed by 'var=expression' omitting the set. The expression can be any arithmetical/string expression.

print var/number/string prints the value of the variable, number or string to the standard output.

do var=start,end[,inc] The command 'do' is followed by a variable and integers for the start value, the end value and an increment. The commands between 'do' and 'continue' are repeated until the end value is reached by the loop parameter 'var', which can by accessed during the loop. This kind of loops can be nested, but loops have to be closed in the same input file. If no loop variable is given, the loop is an endless loop. If you are using the syntax with '=', then start, end and inc can be expressions.

cycle(expr) If expr is true, i.e. not zero, ignore all commands until the next continue.

exit(expr) If expr is true, i.e. not zero, exit from the do-continue loop.

if (expr); then If expr is true, i.e. not zero, execute the following commands, otherwise search for 'elif', 'else' or 'fi'. With 'elif (expr); then' begins an additional 'if'-conditional and the commands following 'else' will be executed if all conditionals are false.

inquire([file=]filename,var) If the file with name 'filename' exists set var to true, i.e. var=1, otherwise var=0.

inquire([var=]varname,var) If the variable with name 'varname' exists set var to true, i.e. var=1, otherwise var=0.

system(command-string,var) Execute 'command-string' in a shell, set var to return value. Not available in restricted mode.

destroy(var-list) Destroy the variable(s) var-list.

history prints the last 1000 executed commands of the first input file to standard output. This can be used for debugging.

setmode(mode,value) Currently, there are the following three modes: debug,time,blank debug activates the debug mode (1). time corresponds to #ti, i.e. whether the running time of commands is measured and displayed (1) or not (0). blank determines whether leading blanks are ignored (1) or start comments (0).

This command is an alternative to the #ti toggle and the explicit debug command.

getmode(mode,variable) saves the current value of mode into the named variable. Modes are: debug,time,blank.

debug All following input lines are copied to the standard output for debugging. Corresponds to command line option -d.

#id. prints the name and author of the program.

#ti switch on/off time measuring of time intensive commands if allowed by the operating system. This commands print their execution time in seconds to the standard output.

#time prints the system time to the standard output if allowed by '#ti'.

showvar prints all bcl-variables which are currently known. This command is replaced by show('variable').

showcmds prints all commands available in the program. This command is replaced by show('commands').

help command shows the help for command if available. The helpfile 'helpfile' will be searched in the directories $MAIPROGSHOME/ads (if $MAIPROGSHOME is set), the installation directory and $HOME/maiprogs/ads. help for your own commands can by written to the file 'myhelpfile' which will be searched in the same directories.

rate(array1,array2,array3,ind) computes the convergence rate of a sequence with the arguments in array1, the values in array2 and the convergence rates in array3, beginning with the index ind, if present, otherwise with the index 0.

getenv(env='..',var='..') reads the environment variable env and stores the value in the bcl-variable var (character). The environment variables USER, HOSTNAME, PWD, HOME and ARGV[0],ARGV[1] etc are always initialized.

openlog(file) opens the logfile. All messages using the logmsg-interface will also be written to the file.

closelog close the old logfile. All messages using the logmsg-interface are send only to the console.

In addition to the user defined variables, there exist also variables to access values computed by the system. Note that these values are only accessible after computation by an explicit command (see below). They can be found in the appendix.

2.4 Commands to steer parallelisation

All programs in the *maiprogs* family are OpenMP, MPI and Cuda aware. In fact the command 'parallel' is common to all programms and part of the bcl-group.

In general the use of OpenMP and MPI is determined by compile-options, and in case of MPI by the user's decision how many processes to start.

For matrix computation and postprocessing, e.g. computation of error indicators parallelisation is completely transparent and uses all resources available.

In case of the iterative solvers, the user can decide on the parallelisation strategy using 'defmpi'.

parallel(ht='..',omp_threads=..,report='..') initializes the number of omp-threads on all mpi-processes. If ht='yes', the maximum number of logical threads is distributed between all mpi-processes on the same machine. Otherwise only the physical cores are shared between mpi-processes on the same machine. Without omp_threads the maximum number of physical threads is used, otherwise omp_threads sets the upper limit of threads available per process.

If report='yes' (default: yes), then a summary of mpi-processes, cores and cuda capable graphic cards is printed, followed by a list of host computers with indiviudal capabilities, i.e. cpu frequency, cores, attached graphic cards. Finally a process table is printed, detailing the resource allocation per process.

This command should always be used if you have more than one mpi-process per machine.

defmpi(data=...,mode=...) sets the data model (data), and sets the communication model (mode), including several auxiliary data structures in dmp as part of the problem description. Has to be done before solve.

data	-1: uniform slice based (default)
	0: mesh based etc, user depended
mode	-1: full broadcast using mpi_broadcast (default)
	-2: full broadcast using partial broadcast
	0: partial broadcast with user defined routine
	1: partial broadcast with mpi_alltoallv

2.5 Commands for computation

problem(otype='..',igltyp=..,nickname='..',pnum=..,bde=..) Defines the problem type consisting of operator and formulation. Only the operator and the igltyp/nickname are of importance, because igltyp/nickname determines which kind of pre-defined problem formulation will be chosen and otype decides, for which operator. otype can take the values 'Laplace', 'Lame', 'Helmholtz', 'Maxwell', 'Convection', 'Maxwell (scattering)',

'Stokes', 'Bilaplace'. pnum denotes the number of the problem, which can be refered later to switch between several problems, see switch_problem. bde can be used if the program can deal with 2d as well as 3d problems. The problem definitions can be found in the file ads/problems, which will be loaded automatically. You can extend the problem definitions, see 'defproblem' and put your own definitions into ads/myproblems to keep them permanently.

This command replaces '#pro'.

switch_problem(pnum=...) activates the problem with number pnum.

addbcdat(fullname='..',nickname='..',i=..,r=..,c='..') creates entries in the global bcdat data structure.

setprob(spline='..',mat='..',dat='..',otyp='..') writes the definitions for spline, mat and dat together with the operator type otyp to the current problem (actp).

setmat(matrix='..',type='..' [,operator='..')] creates an entry in the matrix database for the given matrix name. The type can have the values 'dense,sparse,sparse-row,sparse-col,sparse-sym'. If operator is not given, the operator name is assumed to be equal to the matrix name.

matrix(quadrature='..',gqna=..,gqnb=..,ijn=..,sigma=..,mu=..,nonlin=...,arg='..') The execution of this command will create the internal block matrix representation, allocate the memory for the sparse and/or dense blocks and finally will compute the Galerkin matrices. For this purpose the string 'matstr', set by setproblem, will be analyzed. If arg is given, only the list of matrices in arg=.. will be created and stored in the matrix database. The block matrix of the problem will not be changed.

The given parameters influence the computation of the boundary element method Galerkin matrices.

quadrature	= 'analytic' (default), 'semi-analytic', 'numeric'
gqna	= number of outer quadrature nodes for far field
gqnb	= number of inner quadrature nodes for far field
ijn	= number of layers for geometrically graded quadrature (default: 6)
sigma	= refinement for geometrically graded quadrature (default: 0.17)
mu	= growth factor of quadrature nodes (default: 0.5)
nonlin	= 'update', 'no' (default) determines whether fem and bem matrices or only fem-matrices have to be recomputed (–¿ newtonupdate)

The organization and construction of the block structure will be done in the subroutine setmatrix, the actual computation of the matrices will take place in the subroutines matcomp2c,matcomp3c. But you can add your own subroutine for you own additional matrices.

rhs(gql=.. ,ltyp=...,lmode=...,lmodev=...,nonlin=...) computes the right hand side of the system. gql gives the number of Gaussian quadrature points, ltyp the number of the given boundary condition (jumps on the interface or boundary data), lmode the way of handling integral potential evaluations (0: use the mesh for integration, 3: use the geometry description for integration (faster)), ltypv the number of the given volume data function. nonlin gives the nonlinearity function (0: linear, 1: Hencky vonmises) For this purpose the string 'datstr', set by setproblem, will be analyzed.

'rhs' is supposed to replace all variants of the 'lft' command.

mat mtyp gqna gqnb starts the computation of the Galerkin matrix. If mtyp=0 we use analytical formulas for the computation. If mtyp=1 we do the outer integration numerical and the inner integration analytical. If mtyp=2 we do both integrations numerical. The integer 'gqna' gives the number of nodes for a Gaussian quadrature rule for the

outer integration and the integer 'gqnb' for the inner integration. This command is replaced by matrix(..).

mat0 (only 3D-BEM) starts the computation of the Galerkin matrix. Before any computation the mesh is analyzed. In case of only piecewise constant test- and ansatz-functions optimized analytical formulas are used, which are about ten times faster than the general analytical formulas for the hp-version.

lft gql ltyp lmode ltypv starts the computation of the right hand side. Here 'ltyp' describes the jumps of the boundary data and 'ltypv' describes the data of the interior FEM problem. The meaning of the other parameters remains unchanged.

lft gql ltyp lmode starts the computation of the right hand side. The integer 'gql' gives the number of nodes of the numerical integration needed for arbitrary functions. The integer 'ltyp' gives the index number of the function used as boundary data. The integer 'lmode' describes which kind of integration will be used for the evaluation of the potential functions involved in the direct method. 'lmode=0' means the mesh of the Galerkin method will be used for the evaluation, whereas 'lmode=3' means the coarsest polygon/polyhedron describing the geometry will be used for the numerical integration (much faster with the same precision). If 'lmode=-1' no potential functions will be evaluated. All available functions are defined in the source code modules named 'funct?.f90' which can be extended.

lft gq rhs mode rhsv nonlin computes the right hand side of the system. gq gives the number of Gaussian quadrature points, rhs the number of the given boundary condition (jumps on the interface or boundary data), mode the way of handling integral potential evaluations (0: use the mesh for integration, 3: use the geometry description for integration (faster)), rhsv the number of the given volume data function. nonlin gives the nonlinearity function (0: linear, 1: Hencky vonmises) For this purpose the string 'datstr', set by setproblem, will be analyzed.

#setc n int real sets the parameter set 'n' of the quadrature engine designed for numerical quadrature of singular integral kernels. Details are given in [4].

#getc n shows the parameter set 'n' of the quadrature engine.

#showc shows all parameters of the quadrature engine in a table.

x>x2 saves the current solution vector. Replaced by 'eval'.

x2>x restores the solution vector. Replaced by 'eval'

#pgs. writes the polygonal Signorini boundary.

#pgs ngs reads the polygonal Signorini boundary.

check checks whether the solution respects the Signorini conditions.

#pro otyp rptyp ctyp igltyp defines the investigated problem. The integer 'otyp' defines the given operator and the integer 'rptyp' defines which boundary conditions have to be used.

otyp	equation
0	Laplace equation
1	Lamé equation
2	Helmholtz equation

rptyp	boundary condition
0	Dirichlet
1	Neumann
2	mixed
3	Signorini

The string 'ctyp' states which kind of functions have to be used as test and trial space.

If ctyp=PC we use piecewise defined polynomials, if ctyp=C0 we use continuous piecewise defined polynomials.

If we have Neumann or mixed boundary conditions ctyp is automatically set to 'C0'.

The integer 'igltyp' defines which type of variational formulation for the direct method is used.

igltyp	variational formulation
0	first kind integral equation
1	second kind integral equation
2	Burton-Miller formulation (Helmholtz)

In case of the coupling programs the integer 'igltyp' has the values

igltyp	formulation
0	FEM-BEM coupling, indefinite
1	FEM-BEM coupling, Poincaré-Steklov
2	FEM-BEM coupling, Poincaré-Steklov with contact
3	mixed FEM-BEM coupling, indefinite
4	mixed FEM-BEM coupling with Schur complement
5	mixed FEM-BEM coupling with Signorini contact
6	mixed FEM with Neumann-boundary (Lagrange-Multiplier)

Replaced by 'problem'.

#pro. prints the command '#pro' followed by current values of otyp, rptyp, ctyps and igltyp to the output file.

#pol ptyp defines which kind of polynomials have to be used. Replaced by spline definition via 'setsp' or '#taf.'.

#coll eps activates ε-collocation.

#gal activates the Galerkin method (standard).

#gal. prints the chosen method.

#mortar mortyp sets or resets the use of mortar elements.

2.6 Input/output of data

#cx. 'spline' writes the coefficients of spline 'spline' to the standard output file. The data format used can represent the coefficients for all kinds of base functions. The written data begins with the '#cx' command and can be read by all programs (depending on the dimension).

#cx ng 'spline' dim ptyp ktyp vtypc is followed by the coefficients of spline 'spline'. When this command appears in a bcl-file (formerly written by a bcl-program) the coefficients are stored in the internal data format. The corresponding spline can then be processed further as usual.

#cnod. 'spline' writes the nodal coefficients of spline 'spline' to the standard output file. This command saves a lot of file spaces compared with '#cx.' but can be used only for splines of lowest degree and ptyp=3, i.e. hat functions.

#cnod nnodes 'spline' dim ptyp ktyp vtypc is followed by the nodal values of spline 'spline'. When this command appears in a bcl-file (formerly written by a bcl-program) the nodal values are stored in the internal data format. The corresponding spline can then be processed further as usual.

#c i dim1 reads the components of the solution vector on element 'i'.

#l i dim1 reads the components of the right hand side on element 'i', replaced by #lx.

#c. dim1 writes the components of the solution vector as commands to the output file.

#l. spline writes the components of the right hand side belonging to test-spline 'spline' to the output.

#tno computes a matrix times solution vector product. The result is stored internally and can be accessed by #tl.

#tl. spline writes the right hand side and the result of the matrix-vector product (#tno) to the output file. This command is used for debugging.

#co. writes the solution vector in geometric order to the output file.

2.7 Working with matrices

#smtp smtyp determines the storage scheme of the Galerkin matrix.

smtyp	
0	symmetric part
1	full matrix
2	sparse matrix (only FEM)

symch matrix performs a symmetry test of the given matrix, which has to be computed before by an explicit matrix command. The symmetry test consists of computing $\max|a_{ij} - a_{ji}|$ and $\max|a_{ij} + a_{ji}|$. If the matrix is complex, the numbers will be computed for real and imaginary part separatly. The two or four numbers will be used to classify the matrix in symmetric, hermitean, anti-symmetric, anti-hermitean, vanishing. Otherwise for general real or complex matrices only the numbers $\max|a_{ij} - a_{ji}|$ and $\max|a_{ij} + a_{ji}|$ will be printed.

#fk. 'Matrix-name' 'mode' prints the Galerkin matrix according to the internal storage scheme as command to the output file. Default mode is 'dense', i.e. independend of the internal storage scheme, the matrix is printed in form of a dense matrix. If mode='thin' then only non-zero entries will be printed. If mode='sparse' and the matrix is of type sparse-row or sparse-col, then all internal entries will be printed, even zero entries. In the last two modes, the output of entries will be ended by printing an entry with negative row and col numbers.

#fm. matrix-name prints the Galerkin matrix in dense format as command to the output file.

#fk n0 n1 ktyp 'name' 'mode' reads the Galerkin matrix from the input file. Here 'n0,n1' means the degrees of freedom in case of a full matrix or the number of mesh elements in case of a sparse matrix.

#fks. prints the full Galerkin matrix as command to the output file.

#fk- i reads the Galerkin matrix with 'i' elements and computes the difference to the stored Galerkin matrix.

#sign. prints the sign of the Galerkin matrix elements.

#ew. typ quiet computes the eigenvalues and condition numbers of components of the coupling matrix

$$\begin{pmatrix} A & B & 0 \\ B & C+W & -I+K' \\ 0 & -I+K & -V \end{pmatrix}$$

Here A denotes the part of the FEM-matrix belonging to interior nodes. The string 'typ' can obtain the following values: 'A', 'V', 'W', 'C', 'C+W', 'ABC', 'K-I', 'S' ($S = W+(-I+K')V^{-1}(-I+K)$) for denoting the corresponding component of the coupling matrix.

#ew. matrix computes and prints the eigenvalues of the given matrix, if this matrix is in the matrix database and was computed before by an explicit 'matrix' command.

2.8 Splines

#taf ctyp vtypc vtypf ktyp ptyp 'spline' bmode0 bmode1 'mesh' reads the spline description of spline 'spline'.

ctyp	test and trial space
PC	piecewise continuous defined polynomials
C0	continuous space
C1	C1-continuous space
Hrot	H(rot)-conforming
Hdiv	H(div)-conforming

vtypc= number of vector coefficients in the internal spline representation, vtypf= number of vector coefficients of the spline, ktyp = 1 (real), 2 (complex), ptyp = Type of base function

ptyp	basis functions
0	monomials
1	Legendre polynomials
2	Tschebyscheff polynomials
3	antiderivatives of Legendre polynomials
4	orthonormal Legendre polynomials
5	Lagrange polynomials (uniform nodes)
6	Lagrange polynomials (arbitrary nodes)
10	Raviart-Thomas elements
11	Brezzi-Douglas-Marini (BDM) polynomials
12	Brezzi-Douglas-Fortin-Marini (BDFM) polynomials
13	Nedelec elements (ND)
14	Traces of Nedelec elements (TND)
15	PEERS elements
16	TRT polynomials (traces of RT)
20	Serendipity elements
21	Enriched Serendipity elements

'mesh' is the name of the corresponding mesh. If the spline name does not exists an entry in the database will be created.

#taf. 'spline' writes the spline description of spline 'spline' to the standard output file.

setsp(spline,bmode0,bmode1,ptyp,ctyp,vtypc,vtypf,ktyp[,mesh [,freebd] [,nofreebd] [,Dirichlet] [,Dirichlet_n] [,Dirichlet_t] [,Neumann] [,Neumann_n] [,Neumann_t] [,Signorini] [,Transmission] [,dntyp])] sets the spline information. bmode has to be an integer BCL-array. freebd, nofreebd, Dirichlet, Dirichlet_n, Dirichlet_t, Neumann, Neumann_n, Neumann_t, Signorini, Transmission can be BCL-Arrays. This command generalizes #taf

clear(spline='..') sets all coefficients of spline to zero.

eval('spline=expression') evaluates a spline expression and assigns the value to the given spline name. E.g. u=2*p+u, where u and p are splines belonging to the same or similar spline-class (differences are only allowed in terms of boundary conditions). Factors can also be BCL variables. Using 'Rhs(u)' etc we can also access the components of the right hand side of our global linear equation. The argument of 'Rhs(...)' is the name of the corresponding test spline space. Rows and Columns of Matrices can be accessed by 'Row(Matrixname,Row number)' and 'Col(Matrixname,Column number)' on the right hand side and for assigmnent. The counting of column and row numbers starts with zero.

nlupdate(spline='du',full='u',factor=1.0) computes u=u+factor*du, du can have additional homogenous boundary data.

extend(sp10='u',sp1='u_bd',sp2='u_ex') adds two splines belonging to the same mesh and the same polynomial degrees or one spline sp10 and its trace spline sp1 with the result in sp2.

defspline(spline='...') creates the splines in the given list and allocates local memory for the coefficients. The global solution vector will not be changed. The command #taf can be used to create the necessary entry in the spline database before.

defmatrix(matrix='..',type='..',operator='..',n=..,n0=..,n1=..,ktyp=..,smtp='..') creates the entry in the matrix database (sparse or dense) with the name given by matrix='..' and sets the operator. The type can have the values 'dense,sparse,sparse-row,sparse-col,sparse-sym'. In case of dense matrices (type='dense') we can also allocated the necessary memory, if the matrix will not be computed by matrix(..). If n¿=0 is given a square matrix is defined, if n0¿=0 and n1¿=0 a general matrix is defined. If the matrix is square we can choose the packed storage format for symmetric matrices by smtp='packed'. ktyp=1 is for real numbers, ktyp=2 for complex numbers.

genspl(...) creates internal spline structures and traces of meshes. For this purpose the string 'splinestr', set by setproblem, will be analyzed.

defproblem(operator='..',nickname='..',igltyp=..,fullname='..',dim=..) creates an entry in the problem database, which can be accessed later by the problem(..) command. dim is the dimension of the problem (2d or 3d). The individual problem definition is given by the following setprob,setsp and defmatrix commands. A final 'enddefproblem' command finishes the problem definition. The default problem definitions in the file ads/problems will be read during program start. You are strongly encouraged to write your own file with problem definitions which can be loaded by your bcl script or put your own definitions into the file ads/myproblems. The environment variable HOME with your home directory is available.

enddefproblem finishes the last problem definition (beginning with defproblem) and switches back to the standard behaviour, i.e. the following setprob,setsp and defmatrix commands change the actual problem and not the problem database.

showprobs prints all problem definitions to the standard output. This command is replaced by show('probs').

2.8.1 Geometry and meshes

#p. prints the whole mesh information to the output file (*maibem2*).

#p ng reads the number of elements given by the integer 'ng' from the input file. Usually this command and the following list are produced as output by '#p.' (*maibem2*).

#px. 'spline' Writes the data of mesh and polynomial degrees of spline 'spline' to the standard output file. The data are written in a format suitable for conforming meshes consisting of parallelepipedes.

#px ng bmode0 bmode1 'spline' followed by ng data lines.

Reads the mesh data for the spline 'spline' together with polynomial degrees, the mesh has ng elements and is a dim0 manifold in a dim1-dimensional space.

#pnod. 'spline' Writes the data of mesh and polynomial degrees of spline 'spline' to the standard output file. The data are written in a format suitable for conforming meshes consisting of the nodes used and the nodes and polynomial degrees per element.

#pnod ng nnodes dim0 dim1 'spline' followed by ng+nnodes data lines.

Reads the mesh data for the spline 'spline' together with polynomial degrees, the mesh has ng elements, the number of nodes is nnodes and the mesh is a dim0 manifold in a dim1-dimensional space. In the first 'ng' lines are the node numbers given followed by the polynomial degrees. The node numbers are starting with 0. In 2d the first 4 numbers describe the node numbers, and the next two numbers the polynomial degree. If an element has less than 4 nodes, the remaining nodes have to be set to -1. In 3d the first 8 numbers give the node numbers of the element, and the next three numbers the polynomial degree in x-,y- and z-direction. Hexahedrals have 8 nodes, prisms (wedges) have 6 nodes, pyramids have 5 nodes and tetrahedrals have 4 nodes. If an element has less than 8 nodes, the remaining nodes will be set to -1.

The lines are then followed by 'nnodes' lines containing the coordinates of the nodes, i.e. either x-,y-components in 2d or x-,y-,z-components in 3d.

Example: Two triangles with polynomial degrees 3 and 4.

```
#pnod 2 4 2 2 'u'
  0 1 3 -1 3 3
  1 3 2 -1 4 4
 -1. -1.
  1. -1.
  1.  1.
 -1.  1.
```

#pelem. 'spline' writes the data of mesh and polynomial degrees of spline 'spline' to the standard output file. The data are written in a format allowing hanging nodes.

#pelem ng nnodes nelem dim0 dim1 'spline' followed by nnodes+nelem data lines.

Reads the mesh data for the spline 'spline' together with polynomial degrees, the mesh has ng elements, the number of nodes is nnodes and there are nelem mesh elements in the tree structure of this non-conforming mesh. The mesh is a dim0 manifold in a dim1-dimensional space.

annotbd(annotation='..',rcbal=..) reads the rcval corresponding to the given annotation name (in the external mesh description, ensight or patran format).

import(format='..',file='..',spline='..',p=..,scale=..,gm='..',bmode=(/.,./)) reads the mesh-definition from file (and possibly given values) and creates a spline. Formats are patran,ensight,cfx,gmsh. p indicates a polynomial degree for the spline definition. The mesh can be scaled by the factor 'scale'. If gm is given then we are importing a geometry definition, not a mesh. bmode indicates the dimension of the geometry.

This command replaces readpatran, readensight, readcfx etc.

export(format='...',file='...',spline='...') writes the mesh information (and possible data) in spline to the file, using the given format. Formats are cfx,gmsh.

This command replaces writecfx etc.

readpatran(file='...',name='...',p=...) reads a patran neutral file format mesh description from the file. The corresponding spline and mesh have the given name and the polynomial degree p.

readensight(file='...',name='...',p=...,scale=...) reads a mesh description in ensight file format from the given file. The corresponding spline and mesh have the given name and the polynomial degree p. The mesh can be scaled by the factor 'scale'.

writecfx(file='filename',spline='E:u') writes the solution in cfx-format to a file. Included are the nodal data for all splines given in spline.

readcfx(file='...',spline='...') reads spline coefficients given in cfx-format from a file. In spline several splines separated by ':' can be given.

rebound(mesh='..',rcold=..,rc=..) changes all occurrences of rcold to the value rc. This is used for changing boundary conditions.

shiftgeom(geom='..',off(0:2)) shifts the geometry definition of geom='..' using the off-vector.

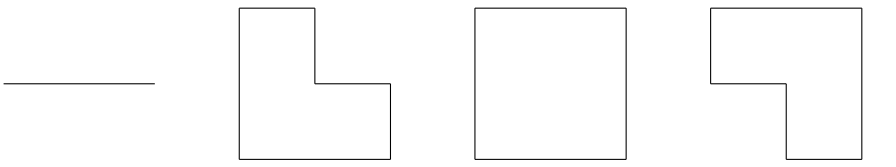
shiftmesh(mesh='..',off(0:2)) shifts the definition of mesh='..' using the off-vector.

updatemesh(mesh='..',update='..') uses the vector-valued update-spline to shift the mesh='..'. If mesh='..' is missing the mesh of update='..' will be shifted. updatemesh works currently on the same mesh and on a copy of a mesh. The case with the same geometry but non-matching meshes has still to be implemented.

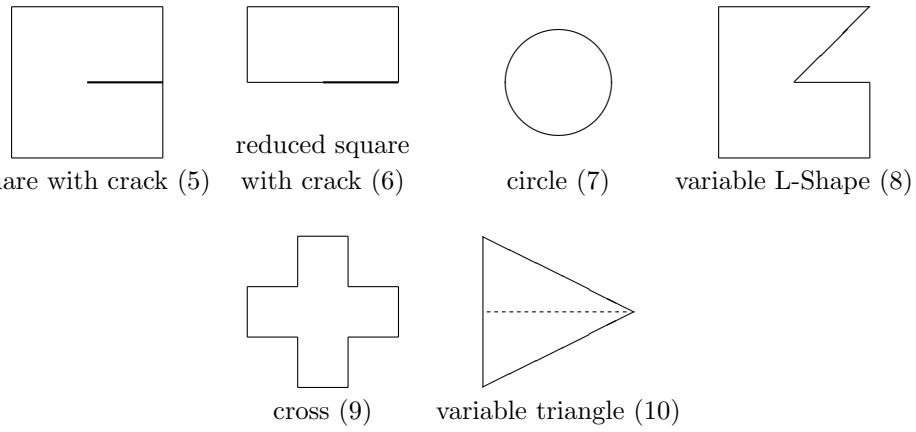
geometry(gtype='..',styp=..,dim0=..,dim1=..,dim2=..,gm=' ') initializes the geometry with the name gtype='..'. The geometry description in the database is called gm=' ' (default: ug). Additionally there will be also created a description of the boundary which is named by appending a '_bd' to the name of the geometry (default: ug_bd). This command replaces '#gm'. The set of pre-defined geometries can be found in the file 'geometries'.

#gm gtyp styp dim0 dim1 dim2 dim3 defines the geometry where the mesh is constructed. In 2D-BEM the following values of the integer gtyp are possible

gtyp	geometry
0	interval [-1,1]
1	L-Shape
2	square
4	upper L-Shape
5	square with a crack
6	reduced square plate with a crack
7	circle
8	variable L-Shape
9	variable Cross (not finished yet)

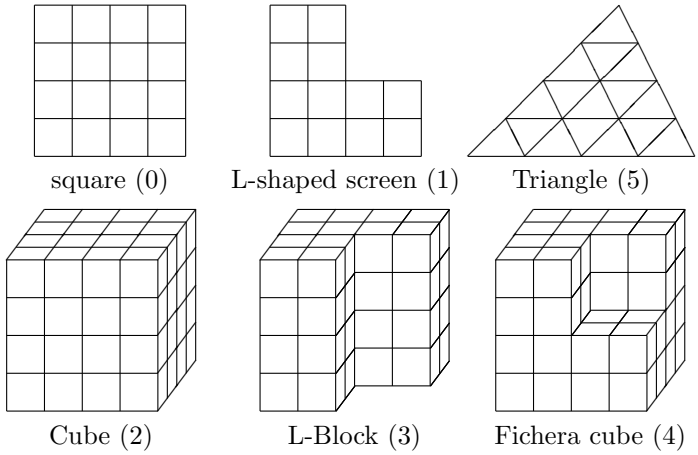


interval (0) L-Shape (1) square (2) upper L-Shape (4)



In 3D-BEM the following values of gtyp are possible

gtyp	geometry
0	Squared screen
1	L-Shaped screen
2	cube
3	L-block
4	Fichera-cube
5	Triangle



The integer styp defines the distribution of singularities of the domain which is important for non-uniform meshes. The real numbers dim0, dim1 and dim2 define the lengths of the domain in x-, y- and z-direction (in the 3D case, otherwise we have only dim0 and dim1), the standard values are 2.0 in each direction. In the case of the variable L-Shape dim2 holds the angle of the opening ($180° \leq$ dim2 $\leq 360°$). This command is outdated and replaced by 'geometry'.

#gm. prints the current values of gtyp, styp, dim0, dim1 (and dim2, dim3) as a command to the output file.

mesh(type='..',n=..,p=..,beta=..,sigma=..,mu=..,elements=..,mode='..',spline=' ',gm=' ',genspl='
The argument type can have the values 'uniform','graded','geometrical', 'Kgraded','global'. In case of 'global', there will be used only the argument 'n' to set the number of global basis function. A further mesh generation will not be done. In case of 'uniform' the

usual geometry description will used to generate a quasi-uniform mesh. In the remaining cases a geometry description, containing singularity information will be used. mode can be 'conform','conforming' or 'hanging'. spline='..' gives the name of spline (and mesh) which has to be initialized (default: u), and gm=' ' denotes the underlying geometry (default: ug). genspl='no' means, that the genspl command will not be executed. This allows to perform additional mesh commands for independently generated meshes. This command replaces the commands '#rc','#gr','#gr2','#go'.

rotmesh(type='..',n=..,p=..,elements=..,mode='..',spline=',mesh=' ',genspl=' ')
The argument type can only have the value 'uniform'. The 2d-mesh mesh will be rotated in n steps to create a 3d-object in spline. elements can have the values hexahedrals, in this case quadrilaterals in mesh will be used to generate hexahedrals in spline and triangles in mesh will give prisms in spline, if the value of elements is tetrahedrals all elements in mesh will create tetrahedrals. In the last case n should be a multiple of 2. mode can be 'conform','conforming' or 'hanging'. spline='..' gives the name of spline (and mesh) which has to be initialized (default: u), and gm=' ' denotes the underlying geometry (default: ug). genspl='no' means, that the genspl command will not be executed. This allows to perform additional mesh commands for independently generated meshes.

#pxg. 'geom' writes the generalized geometry(G) definition 'geom' to the output file. See #pxg for the definition of the data format.

#pxg n bmode0 bmode1 'geometry' reads the generalized geometry(G) definition from the input file. n is the number of macro elements (and lines) in this definition. If bmode0=bmode1 we have a volume mesh. Then every line consists of typ nr node_1 node_2 ... node_nr dn If bmode0¡bmode1 we have a surface mesh. Then every line consists of typ nr node_1 node_2 ... node_nr nx dn

nr is always the number of nodes needed for the description of the element. node_1 node_2 etc are the coordinates of nodes, nx is the normal direction, giving the surface element an orientation. dn is reserved for future use, set it to zero. typ is the type of the macro element. 0 always means the polygonal/polyhedral element. If bmode0=1, then typ=0, nr=2 means a line segment. If bmode0=2, then typ=0, nr=3 gives a triangle, nr=4 gives a quadrilateral element. If bmode0=3, then typ=0, nr=4 gives a tetrahedral element, nr=5 gives a pyramidal element (the first four nodes give the base of the pyramid), nr=6 gives a prismatic element (the first three nodes define the bottom), nr=8 gives a hexahedral element (the first four nodes define the bottom).

typ=1 often defines arcs, circles, cylinders, spheres. E.g. bmode0=1, bmode1=2, typ=1, nr=3 defines an arc segment, described by three nodes located at the beginning, the middle and the end of the segment. bmode0=2, bmode1=2, typ=1, nr=4 defines a circle segment, the first three nodes define the arc segment, whereas the last node defines a corner. typ=1,nr=5 defines a quadrilateral element with one curved (arc) boundary segment, the first three nodes define the arc segment, whereas the two nodes define corners. Effectively nodes 1,3,4,5 define the corners of the curved quadrilateral element, whereas node 2 is used to define the curvature. typ=1,nr=6 defines a quadrilateral element with two curved (arc) boundary segments, opposite to each other. Nodes 1,3,4,6 define the corners of the curved quadrilateral element. Node 2 defines the curvature of the first arc segment and node 5 defines the curvature of the second arc segment.

bmode0=2,bmode1=3 typ=1,nr=4 defines a triangle with one curved (arc) boundary segment, the first three nodes define the arc segment, whereas the last node defines a corner. typ=2,nr=4 defined a curved triangular element (sphere-shell). The first three nodes define the corners of the curved triangle, the last node is a node in the curved triangle which allows to determine the center of the sphere. typ=2,nr=6 defines

a curved quadrilateral element (cylinder-shell). The first three nodes define the first arc segment, the last three nodes the second arg segment.

bmode0=3,bmode1=3 typ=1,nr=8 defines a cylinder with one curved surface. The first 4 nodes define one end of the cylinder, the next 4 nodes the other end. typ=1,nr=10 defines again a cylinder, this time based on a quadrilateral element. typ=1,nr=12 defines a hollow cylinder shell (not working yet). The first 6 nodes define one end of the cylinder, the second 6 nodes the other end. From each set of 6 nodes the first 3 nodes describe the outer arc and the other ones the second arc (cf. bmode=(/2,2/),typ=1,nr=6).

typ=2,nr=4 defines a sphere. The first node is the center, the last three nodes define the surface segment. typ=2,nr=8 defines a spherical segment with two curved surfaces. The first three nodes define the corners of the first curved triangle, the fourth node is an additional node on the surface. The next three nodes define the corners of the second curved triangle, the last node is an additional node on the second surface.

#pxs. 'geom' writes the generalized geometry(S) definition 'geom' to the output file.

#pxs n bmode0 bmode1 'geometry' reads the generalized geometry(S) definition from the input file.

#pg. dim1 dim2 writes the mesh describing the geometry of the boundary or domain as a command to the output file. dim1 and dim2 describe more detailed which geometry mesh has to be used if there is more than one avaliable, as in command '#bmode'.

#pg ngg dim1 dim2 reads the mesh with 'ngg' elements describing the geometry of the boundary from the input file, following this command.

#ps. dim1 dim2 writes the mesh describing the location of the singularities as a command to the output file.

#ps ngs dim1 dim2 reads the mesh with 'ngs' elements describing the singularities on the boundary from the input file

#pbd. writes the mesh describing the vanishing boundary elements as a command to the output file.

#pbd ngbd dim1 dim2 reads the mesh describing the elimination of degrees of freedom on the boundary or anywhere else from the input file.

markcircle(spline='..',radius=...) tries to find a boundary line with the given radius in the named spline/mesh and sets it to the number -2.

meshbreak(mesh='..',number=..) breaks the elemnts denoted by #pbr if there was no break before (directly after a #pbr command), otherwise break the elements given by number next to the last break.

#g. writes the geometry as a command to the output file (see '#gm.') followed by the description of the mesh, if structured by the generating command otherwise as explicit mesh description (see '#p.', '#px.').

#pn. writes the node numbers of each (geometry-) element to the output file.

#pn ng reads the node numbers of each (geometry-) element.

#nod. writes the coordinates of each node to the output file.

#nod nnodes reads the coordinates of each node.

2.8.2 Defining global coefficients

#lm lambda mu lambda mu reads and sets the Lamé coefficients λ and μ for the inner domain (first) and the exterior domain (second pair).

#lm. writes the Lamé coefficients λ and μ for the inner domain (first) and the exterior domain (second pair) to output as a command.

#ke k-module e-module k-module e-module reads the K-module and the E-module and computes the Lamé-coefficients λ and μ for the inner domain (first) and the exterior domain (second pair).

#ke. writes the K-module and the E-module as computed by the Lamé-coefficients for the inner domain (first) and the exterior domain (second pair) to output as a command.

#ep e-module poisson e-module poisson reads E-module (Youngs module) and Poisson number and computes the Lamé-coefficients λ and μ for the inner domain (first) and the exterior domain (second pair).

#ep. Writes E-Module (Youngs module) and Poisson number to output for the inner domain (first) and the exterior domain (second pair) to output as a command.

#maxw. write the information for mu, sigma and omega for a Maxwell problem to the output.

#maxw mu-mode sigma-mode omegai omegar reads the mode number for the mu-function, the mode number for the sigma function and the imaginary part and the real part of omega (due to some old scripts).

#kw wavenumber1 wavenumber2 terms reads the real wavenumber for the interior domain (first) and the wavenumber for the exterior domains, followed by the number of terms in a series expansion, if the Galerkin matrices are computed by performing a series expansion of the fundamental solution.

Replaced by #kwc for complex wavenumbers.

#kw. writes the real wavenumbers for the interior and exterior domain to the output.

Replaced by #kwc. for complex wavenumbers

#kwc wavenumber1r wavenumber1i wavenumber2r wavenumber2i terms reads the complex wavenumber for the interior domain (first) and the wavenumber for the exterior domains, followed by the number of terms in a series expansion, if the Galerkin matrices are computed by performing a series expansion of the fundamental solution.

#kwc. writes the complex wavenumbers for the interior and exterior domain to the output.

#bm bmreal bmimg sets the complex valued coupling parameter of the Burton-Miller formulation.

#bm. prints the complex valued coupling parameter of the Burton-Miller formulation.

2.8.3 Accessing internal data structures

#hrc. 'spline' writes the element connection information as command to the standard output file.

#hro. writes the internal table of element offsets as command to the output file.

#rci. 'spline' writes the local to global mapping information (including weights) to the standard output file. Using this information together with '#cx' etc. a second program can reconstruct the original internal representation. Otherwise only the values are identical for the splines but the internal representation can differ.

#rci ncom bmode0 bmode1 'spline' dof is followed by ncom data lines. Reads the internal local to global mapping information for spline 'spline' from the data file.

#rco. writes the table of reordering for block preconditioners to the output file.

#pmi. np writes the internal coefficients of the base polynomial with degree 'np' as a command to the output file.

#gam gam1 gam2 gam3 gam4 gam5 reads the internal coefficients $\gamma_i, i = 1, \ldots, 5$ from the input file (Lamé).

#gam. prints the internal coefficients $\gamma_i, i = 1, \ldots, 5$ (Lamé).

show(what='..') writes the requested internal information to standard output.

what	
'matrix'	writes the internal Block-Galerkin matrix with its subblocks.
'var','variables'	writes all actual known bcl-variables.
'history'	writes the sequence of executed commands.
'cmds','commands'	writes all commands which are implemented in the current program together with the information whether help is available.
'memory'	writes the memory used for all splines, meshes, geometries and matrices stored in the internal databases. This does not cover the memory for locally defined structures.
'sparse'	writes the description of all sparse matrices.
'dense'	write the description of all dense matrices.
'splines'	writes the description of all splines.
'geometries'	writes all geometries.
'annotbd'	writes all annotations.
'problem'	writes the current problem description.
'probs'	writes all problem definitions to the standard output.
'splstr'	writes the current spline string 'splstr'.
'all'	writes all available information in the sequence given above.

qsp(var,action,spline) queries the spline database for the named 'spline' and retrives the data named in 'action', which are written to the variable 'var'. Possible 'actions' are: dof,ng,ktyp,dim,xdim dim is the dimension of the splineclass, xdim is the dimension of the spline itself. (Not every name is the name of a splineclass). If the answer is not known qsp will return -1.

matrixshow shows the structure of the block matrix. This command is replaced by show('matrix').

meshquality(mesh='..') gives information of the quality of the mesh, e.g. largest element length/smallest length of neighbor elements etc.

boundbox(mesh='..') determines the boundary box for all boundary parts (distinguished by the rc-values).

checkcont(spline='..',trace='..') If only 'spline' is given, checkcont checks for continuity of the spline description across faces and edges, depending on the ctyp (PC,C0,Hdiv,Hrot,...). In a testvector a coefficient of a local basis function of an element will be set to one. Then the neighbor elements will be checked, whether the basis function belongs to an edge or face and on this edge or face the spline will be evaluated on both elements. If 'spline' and 'trace' are given checkcont checks whether 'trace' and trace of 'spline' give the same values.

2.9 Fast iterative solvers

The following solvers often use the same parameters. Therefore the parameters in common are defined first.

eps	real	the last relative change of the solution vector in the Euclidean norm or the last relative error in the energy norm used in the stopping criterion
mtop	integer	number of levels, starting with 0 denoting the coarsest mesh
omega	real	damping factor of the smoother
mdc	integer	preconditioning
	0	no preconditioning of the multigrid algorithm
	1	diagonal preconditioning
	2	preconditioning by the difference operator of second order for the weakly singular kernel
	4	diagonal preconditioning in the Haar basis
mnum	integer	maximal number of iterations which has to be performed until the algorithm stops
hpmodus	integer	mesh generation starting with the finest mesh
	0	structured h-version
	1	structured p-version
	2	structured hp-version
ddmodus	integer	mode of domain decomposition
	-1	one-dimensional subspaces
	0	block spaces without a global coarse grid space
	1	block spaces with full coarse grid space
	2	coarse grid space with one-dimensional subspaces
	3	overlap a la Pavarino
stp	integer	parameter for the mesh generation. If we have a p-version, an integer stp$<$ 0 means the polynomial degree has decreased by stp at each level, otherwise if we have stp$>$ 0 the polynomial degree has to be divided by stp at each level. If we have a h-version, the element length of each element has to be multiplicated by stp at each level.
grflag	integer	whether an inverse Grammian matrix has to be used at the levels
	0	the algorithm has to be performed without an inverse Grammian
	1	the inverse Grammian has to be used
	2	an approximate inverse Grammian has to be used (not complete yet)
abrflag	integer	which stopping criterion has to be used.
	0	the iterations are controlled by the error in the energy norm. In this case a solution for comparison has to be calculated before.
	1	the algorithm is controlled by the relative change of the solution vector measured in the Euclidean norm
	2	the algorithm is controlled by the norm of the residual
mit	integer	the iterative solver which has to be preconditioned.
	0	CG
	1	GMRES
	2	HMCR
	-3	determine the condition number
	-6	Gaussian solver
	-7	determine the contraction rate (MSM)
	-13	condition number using Arnoldi iterations

The following parameters are used only for the multigrid algorithm

nu1	integer	number of presmoothing steps
nu2	integer	number of postsmoothing steps
mu	integer	number of multilevel steps at each level
	1	V-cycle
	2	W-cycle
	0	denotes the variable V-cycle
mds	integer	kind of smoother.
	0	damped Jacobi
	1	damped Gauss-Seidel (SOR)
	2	Richardson scheme

#mgen mgenabled enables (mgenabled=1, default) resp. disables (mgenabled=0) the allocation of memory for the multilevel schemes (mlev, mgcg, masv2 ...).

2.9.1 Iterative Solvers

solve(eps=..,mdi='..',mdc='..',mit='..',abrflag=..,quiet=..,restrt=..) is the interface to the new solver for preconditioned block systems. It replaces rlgs, mcg, mbpx, ddsolv, asmcg, msmcg, varacg, ahcg, etc. eps is the stopping threshold, mdi the initial value (x=0,unchanged,x=1,x=l/diag) mdc the elementary preconditioner (no,diag,auto or full list), mit the solver (CG,GMRES,MINRES or HMCR,BICGSTAB,CGNE,CGNR, EW for eigenvalues,GAUSS) abrflag the stopping criterion (1: relative change), quiet the amount of output (-1: nothing, 0: success, 1: eigenvalue estimate), restrt the number of restart vectors for GMRES.

mdc can contain the list of preconditioners applied to the individual splines. There are two modes implemented. In the diag-block mode, the preconditioners act only on the block diagonal, and we only have to give a list of the preconditioners, e.g. mdc='Pu:PN', or mdc='2*Pu:-1*PN'. Here 'Pu' is the name of the first preconditioner, defined by defprec and 'PN' is the name of the second preconditioner. In the general mode we have to name explicitly the names, on which the preconditioners have to act. If we have for example a matrix structure like $\begin{matrix} a_{11} * N1.V.N1 & a_{12} * N1.V.N2 \\ a_{21} * N2.V.N1 & a_{22} * N2.V.N2 \end{matrix}$, where N1=Spline(N):N2=Spline(N), and we have a preconditioner for V, e.g. called PN, then we can write the preconditioner for our system in the following way $b_{11} * N1.PN.N1 : b_{12} * N1.PN.N2 : b_{21} * N2.PN.N1 : b_{22} * N2.PN.N2$ where $B = (b_{ij}) = A^{-1}$, $A = (a_{ij})$.

This command will replace all specialized block-preconditioners.

For the declaration of preconditioners see the bcl-command defprec.

defprec(mode='..',spline='..',name='..',mat='..', additional parameters) defines the preconditioners.
mode gives the type of preconditioner:

ID	Identity, i.e. no preconditioner, no additional parameters
MG	Multigrid parameters: nu1,nu2,mu,mtop,omega,mdc,mds,hpmodus,stp,expo
BPX	BPX parameters: mtop,hpmodus,stp,mdc
HIERA	Hierarchical preconditioner parameters: mtop,hpmodus,stp
MAS	Multilevel additive Schwarz parameters: mtop,hpmodus,stp,mdc Similar to BPX but divides by diagonal of restricted Galerkin matrices and solves exact on coarsest level. mdc=4 uses transformation to Haar-basis for Single Layer Potential
INV	Inverse (optimized, internally by multigrid or LR decomposition)
INVCG	Inverse, by solving an auxillary problem with CG parameters: mit (number of solver, currently only 0 for CG)
DD	Domain decomposition (with/without coarse grid) still under construction parameters: domains,overlap,cflag,qtyp
ASM	HP Domain decomposition (with/without coarse grid) under construction parameters: hpmodus,ddmodus,domains,qtyp,optmode,overlap hpmodus 0: h-version 1: p-version ddmodus -1: diagonal (1 dim-blocks) (h+p-version) ddmodus 0: without overlap (h-version) 1: with full overlap (h-version) 0: p-blocks without coarse grid (p-version) 1: p-blocks with full coarse grid (p-version) 2: hp-version with h-block and diag p-blocks 3: Overlap a la Pavarino (p-version)
HH	Hhp domain decomposition (with/without coarse grid) parameters: cmesh,cmat,csmode,solver,overlap,domains csmode : Construction of coarse grid mesh values: no,cmesh,auto (using domains),box(using domains) cmesh : name of given coarse grid mesh cmat : name of given coarse grid matrix solver : name of solver for local subproblems overlap: Overlap in construction of fine grid spaces domains: number of domains (in 1d) for construction of coarse grid (and fine grids)

spline is the name of the splineclass, which is the argument of the preconditioner.

name is the name of this special preconditioner block, together with its parameters, which is used later in the list of 'solve'.

mat is the name which is used in constructing the preconditioner, e.g. for MG or INVCG. It can be any initialized matrix, or I+Matrixname. I is the mass matrix.

rlgs eps mdi mdc mit abrflag quiet restrt solves the linear system. eps is the stopping threshold, mdi the initial value (-1: x=0, 0: unchanged, 1: x=1, 2: x(i)=l(i)/a(i,i)), mdc the elementary preconditioner (0: without, 1: diagonal), mit the solver (0: CG, 1: GMRES, 2: MINRES, 3: BICGSTAB, -3: eigenvalue determination, -6: Gauss), abrflag the stopping criterion (1: relative change), quiet the amount of output (-1: nothing, 0: success, 1: eigenvalue estimate), restrt the number of restart vectors for GMRES.

mlev eps nu1 nu2 mu mtop omega mdc mds mnum hpmodus stp grflag abrflag expo quiet solves the Galerkin system using the Multigrid algorithm.

asm eps mnum hpmodus ddmodus domains abrflag theta quiet solves the Galerkin system by the Additive Schwarz algorithm with damping parameter theta using exact solution at each subspace.

msm eps mnum hpmodus ddmodus domains abrflag theta quiet solves the Galerkin system by the Multiplicative Schwarz algorithm with damping parameter theta using exact solution at each subspace.

lgs eps mdi mdc bn omega quiet restrt solves the Galerkin system. The real number 'eps' controls the stopping criterion. The integer 'mdi' determines the start value for iterative solvers.

mdi	start value
-1	all components of the start vector are equal to 0
0	the start value is unchanged by lgs, i.e. it is given by an other command
1	all components of the start vector are equal to 1
2	the start vector is the right hand side divided by the diagonal of the system matrix
3	the start vector is the solution of the linear system defined by the right hand side and the tridiagonal part of the system matrix

The integer 'mdc' determines which solver has to be used.

mdc	solver
0	CG-scheme without preconditioning
1	CG-scheme with diagonal preconditioning
2	CG-scheme with tridiagonal preconditioning
3	CG-scheme with block preconditioning determined by the integer 'bn'
4	CG-scheme with block-row preconditioning determined by the integer 'bn'
10	GMRES-scheme without preconditioning
11	GMRES-scheme with diagonal preconditioning
-1	exact solution by Cholesky decomposition
-2	damped Jacobi iteration determined by the damping factor omega
-3	Gauss-Seidel iteration
-4	SOR-iteration determined by omega
-5	Richardson iteration determined by omega
-6	exact solution by a LDL-decomposition

The integer 'bn' determines which part of the system matrix has to be used for the block preconditioners. The real number 'omega' is the damping or relaxation factor used by some iterative solvers. The integer 'quiet' gives the type of informative output. If quiet=0 the CG-schemes compute also an approximation for the condition number by the Lanczos-method if a CG-scheme is used. The integer restrt gives the number of iterations when the GMRES-scheme has to be restarted.

2.9.2 Preconditioned Schemes

mcg eps nu1 nu2 mu mtop omega mdc mds mnum hpmodus stp grflag mit quiet solves the Galerkin system using a Krylov-method preconditioned by the Multigrid algorithm.

mblcg eps nu1 nu2 mu mtop omega mdc mds mnum hpmodus stp grflag solves the Galerkin system using the CG-scheme with multigrid as block preconditioner on the lowest polynomial degrees and diagonal preconditioning on the higher degrees.

mbpx eps mtop mnum hpmodus stp mdc abrflag mit quiet solves the Galerkin system by using the BPX-Algorithm as preconditioner for the Krylov-schemes.

mblpx eps mtop mnum hpmodus stp solves the Galerkin system by the CG-scheme using the BPX-algorithm as preconditioner on the lowest polynomial degrees and diagonal preconditioning on the higher degrees.

masv2 eps mtop mnum hpmodus stp mdc abrflag mit quiet solves the Galerkin system using a Krylov-scheme preconditioned by the Multilevel Additive Schwarz algorithm with nested subspaces and exact solution at the lowest level.

asmcg eps mnum hpmodus ddmodus domains abrflag mit quiet qtyp solves the Galerkin system using an iterative solver (Krylov-scheme) preconditioned by the 2-level Additive Schwarz algorithm and exact solution at each subspace.

msmcg eps mnum hpmodus ddmodus domains abrflag mit quiet qtyp solves the Galerkin system using an iterative solver preconditioned by the 2-level Multiplicative Schwarz algorithm and exact solution at each subspace.

varacg eps mnum hpmodus domains overlap abrflag mit quiet restrt qtyp solves the Galerkin system using an iterative solver preconditioned by the 2-level Additive Schwarz algorithm and exact solution at each subspace, where a Hh subspace decomposition is used.

ahcg eps mtop mnum hpmodus stp abrflag mit quiet solves the Galerkin system using an iterative solver preconditioned by the multilevel additive Schwarz method given by the hierarchical base.

ddsolv(eps,mit,mdi,mnum,quiet,domains,overlap) solves the block system either with a global block preconditioner if domains=0, or via a domain decomposition preconditioner with overlap. eps : tolerance value mit : global solver mdi : initial value (mdi=0: unchanged, mdi=-1: x=0, mdi=1:x=1,mdi=2:x=b/diag(A)) mnum : number of global iterations quiet : amount of output domains : number of domains (cubic) overlap : amount of overlap (0.0 .. 1.0)

mcg eps nu1 nu2 mu mtop omega mdc mds mnum hpmodus stp grflag mit quiet modi abrflag solves the coupled system. 'mdc=0' means 3-block-preconditioners and 'mdc=1' 2-block-preconditioners. 'modi' consists of 3 digits. The first digit determines the preconditioner for the FEM part, the second digit the preconditioner for the Dirichlet-data part and the third digit the preconditioner for the Neumann-data part (i.e. the single layer potential).

Digit	preconditioner
0	no
1	multigrid
2	BPX
3	diagonal
4	hierarchical
5	exact inversion

uzawa eps1 eps2 rho mit performs the Uzawa algorithm for the mixed FEM-BEM coupling with Signorini contact. eps1 gives the convergence threshold for global convergence, eps2 gives the convergence threshold for the inner convergence, rho stires the Uzwawa algorithm and mit determines the solver used to solve the inner system.

2.10 Analyzing the Galerkin solution

#hno. exhnorm prints the energy norm and the square of the energy norm to the standard output and as command to the output file. If the exact energy norm 'exhnorm' is given, also the error in energy norm will be computed.

#rno. prints the norm of the residual $\|Ax - b\|_2$ and the relative residual $\|Ax - b\|_2/\|b\|_2$ for the actual linear system.

#hno energy reads the energy norm from the input file. This is meanly used for the collection of energy norms and the postprocessing done by *maigraf*, which allows the computation of convergence rates.

#no. notyp spline computes and prints the norm of the spline 'spline'. 'notyp' is a string with the following values:

notyp	norm
L2	L^2-norm
H1	H^1-norm
H10	H^1_0-norm
Hdiv	H(div) norm
Hdiv0	H(div) semi-norm
Hcurl	H(curl) norm
Hcurl0	H(curl) semi-norm
E	Energy norm

The result will be printed and stored in the variable 'NORM'.

norm(var,norm,spline1,spline2) computes the norm of spline1, or, if present, of spline1-spline2 and writes the result in the BCL variable var. The type of norm can be the Euklid norm, any matrix induced norm (given by a matrix name) or any of the standard norms 'L2','H1', etc., see #no.

value(spline='..',x=...,var='..',mode='..') computes the value of the given spline in the point x and stores the result in the given variable var. The result depends on mode, i.e. for mode=value (default) simply the value of the spline will be evaluated, for mode=grad the gradient of the spline will be computed.

#mean. spline computes and prints the integral mean of the spline 'spline'. The result will be stored in the variable 'MEAN'.

#err. gq rhs norm quiet 'spline' 'name' computes the error of 'spline' if the exact solution is known. gq gives the number of Gaussian quadrature points, rhs(=ltyp) the number of the solution function, norm the norm (L2,H1,H10,Hcurl,Hdiv etc), quiet the amount of output and spline the spline under consideration. 'name' is the name of the exact solution, if the name of the exact solution can not be guessed by the name of the spline.

approx aptyp rhs spline name approximates a given function with number rhs with the spline 'spline'. If aptyp=0, interpolation will be used. If aptyp=1, L2-projection will be used. If name is given, it is the name of the function which has to be approximated.

#dif. gql ltyp notyp computes and prints error of the Galerkin solution compared with the exact solution (if known). The integer 'gql' gives the number of nodes of a Gaussian quadrature rule. 'ltyp' gives the index number of the exact solution used for comparison. 'notyp' gives the index number of the norm which has to be used. notyp=0 denotes the L^2 norm.

exl gql ltyp computes the exact solution in the internal form.

eri. gql ltyp notyp ax ay ex ey nx ny num computes and prints the error in the interior on an interval in the norm 'notyp'.

val. gql ltyp ax ay ex ey nx ny num computes and prints the value of the solution on an interior interval.

valp. gqn ltyp x y nx ny computes the value of the numerical solution at the point (x,y) inside the domain and also the exact solution.

vall. ltyp x y nx ny computes the value of the exact solution at the point (x,y) inside the domain.

erin. gql ltyp notyp ax ay ex ey nx ny num computes and prints the error of the normal derivative of the solution in the interior on an interval in the norm 'notyp'.

valn. gql ltyp ax ay ex ey nx ny num computes and prints the value of the normal derivative of the solution on an interior interval

ewx. eps notyp computes and prints the maximal eigenvalue of the system matrix using the iterative von Mises scheme. The precision of the computed maximal eigenvalue is determined by the real number 'eps'. The integer number 'notyp' determines the norm in which the eigenvalue is computed. If notyp=-1 the euclidian norm is used, if notyp=0 the L^2-norm is used.

ewn. eps notyp computes and prints the minimal eigenvalue of the system matrix using the iterative von Mises scheme. The precision of the computed minimal eigenvalue is determined by the real number 'eps'. The integer number 'notyp' determines the norm in which the eigenvalue is computed. If notyp=-1 the euclidian norm is used, if notyp=0 the L^2-norm is used.

tlib eps tmode ijn quiet starts the test of the implemented integral libraries for the Galerkin elements and potentials. The comparison of the different methods implemented in the libraries is determined by the relative error given by the real number 'eps'. The integer number 'tmode' determines which kind of tests have to be applied.

tmode	action
-1	perform all tests
0	compare the analytical computation with outer numerical integration
1	compare analytical computation with full numerical integration
2	compare outer numerical integration with full numerical integration
3	test for adjointness
4	test for consistency (internal symmetry)
5	test the potentials

'ijn' gives the number of levels of the graded quadrature scheme used for the numerical integrations. The remaining parameters can be controlled by '#setc'.

tlib0 eps tmode gqna gqnb tests the optimized integral libraries for low polynomial degrees.

2.11 Adaptive refinement

adap(theta= .. , admode= .., gq=..,tlocfs=..,sresno=..,mode=' ',pxy=..,gamma='..) replaces (will replace) resh,reshxy,resp etc. mode='h','xy','p'

resh theta admode gqr tlocfs sresno computes the refinement table for a adaptive h-version. The real number 'theta' decides about the h-refinement. The integer 'admode' gives the use of the refinement parameter 'theta', i.e.

'admode'	strategy
0	refine if $\eta_i \geq \theta \eta_{\max}$
1	refine if $\eta_i \geq \theta \frac{1}{N} \sum_{j=1}^{N} \eta_j$
2	refine if $\eta_i \geq \eta_{\theta N}$ $(\eta_1 \leq \eta_2 \ldots \leq \eta_N)$

The integer 'gqr' gives the number of nodes of a Gaussian quadrature rule for the calculation of local residuals. The integer 'tlocfs' gives the index number of the error indicator. The local error indicator is weighted by a power of h given by the real number 'sresno'. Available error indicators are

'tlocfs'	error indicator
0	Residual
2	Gradient of the Residual
3	Hierarchical decomposition of the Residual

resxy pxy gqxy tlocfs sresno computes the refinement direction in the case of 3D-BEM after applying 'resh' for a adaptive h-version with direction control. The real number 'pxy' decides about direction. The integer 'gqxy' gives the number of nodes of a Gaussian quadrature rule for the calculation of local residuals. The integer 'tlocfs' gives the index number of the error indicator. The local error indicator is weighted by a power of h given by the real number 'sresno'.

resp gamma gqp tlocfs sresno computes the refinement table for a adaptive p-version. The real number 'gamma' decides between p-refinement or h-refinement. The integer 'gqp' gives the number of nodes of a Gaussian quadrature rule for the calculation of local residuals. The integer 'tlocfs' gives the index number of the error indicator. The local error indicator is weighted by a power of h or p given by the real number 'sresno'.

#res ng bmode0 bmode1 'spline' followed by ng-lines reads the indicators for spline.

#res. 'spline' writes the error indicators for 'spline' computed by 'resh' etc as command to the output file.

refine(type='..',spline='...',genspl='...') if type = 'all' then all elements will be refined, otherwise the information in the ref-array (set by resh) is used for the refinement of the given spline or spline-list (entries separated by ':') (default: u). If genspl = 'no' the genspl command will not be executed.

ref performs the new mesh for a given refinement table, replaced by 'refine'.

#ref. 'spline' writes the refinement information to the output file.

#ref ng bmode0 bmode1 'spline' followed by ng-lines reads the refinement information for 'spline' from the input file.

2.12 Postprocessing

#cnin cntyp smode x y initializes the computation of stress intensity factors. The integer 'cntyp=1,2,...' gives the number of the singularity function, resp. the number of the eigenvalue which has to be determined. The integer 'smode=0,1' gives the number of the singularity function if there is more than one for a given eigenvalue. The coordinates 'x,y' give the position of the singularity function. It has to be a node of the mesh. If 'x,y' are not given '0,0' is assumed. Then all internal parameters which are necessary to compute the singularity function are computed automatically by determining the opening angle of the mesh in 'x,y' and inspecting the boundary conditions on both sides of 'x,y'.

#c1. gql ltyp ltypi computes the stress intensity factor. The integer 'gql' gives the number of Gaussian nodes to be used. The integer gives the index number of the boundary data (see 'lft') and the integer 'ltypi' gives the index number of the volume data.

#sch. x1 x2 (x3) y1 y2 (y3) computes the leading singular exponent on a straight line from y to x on geometric refined points.

bem2fem(u0='..', t0='..', f='..', fem='..') uses the representation formula to construct the solution in the domain, i.e. u0 and t0 represent the boundary data (given splines or functions (u0,t0)) and f represents the volume data (spline or function (f)). The solution in the domain is written to the spline given by fem.

2.13 Commands of *maigraf*

maigraf can read the output of *maibem3*, *maifem3*, *maicoup3*, *maifem2*, *maibem2*, *maicoup2* and *maicont2* and has also all their mesh generating commands. Special for *maigraf* are the commands for printing meshes and solutions.

#bmode bmode0 bmode1 sets the bmode(0) and bmode(1) values for which the primary output is generated, i.e.

bmode0	bmode1	
1	2	2D-BEM
2	2	2D-FEM
2	3	3D-BEM
3	3	3D-FEM

#pmode mode clmode ncols sets the mode of output. Currently five different programs can be used for the visualization of meshes and solutions. Note, in case of 'Xfig'-output the file name has to be given by '#out(2)' in advance (before the command '#pmode').

mode	format
0	LaTeX (default)
1	node list for GNUPLOT
2	Plotmtv (contour plot)
3	Postscript output by UNIRAS (if available)
4	Xfig-3.2

'clmode=1' prepares for colored output and 'ncols' gives the number of colors for plots.

graphic(option1=arg1,option2=arg2,. . .) sets the options for drawing.

option	argument
pencolor	black|red|blue|green. . . (default: black)
textcolor	black|red|blue|green. . . (default: black)
fontsize	points (default: 12)
scale	on|off (draw a scale or not, default: on)
title	on|off (draw a title or not, default: on)
paper	A4|Letter (default: A4)
range	on|off (determine range for scale, default: on, if off then ZMIN,ZMAX are used)

#scale scale-factor sets the scaling factor. This factors scales all geometric values.

#scale. writes the scaling factor.

#view x y z w1 w2 w3 sets the viewpoint (x,y,z) and the view direction (w1,w2,w3) in degrees for all 3d plots.

#bbox x1 y1 z1 x2 y2 z2 sets the bounding box for 2d/3d bodies.

#viewplane ll(0:2) lr(0:2) ur(0:2) ul(0:2) lower left, lower right, upper right, upper left sets the viewplane for 3d-bodies.

view(mode='..') sets the view mode. If mode='front' we see only the frontal part of a 3d structure. If mode='back' or 'rear' we see only the rear part of a 3d structure. If mode='all' we see the whole surface.

r>nodv is equivalent to view('front').

r>nodh is equivalent to view('rear').

#mesh x y dmode factor opt 'spline' writes the mesh of 'spline' in the format given by '#pmode' to location x y in dmode-form (2: plane, 3: perspective view) with additional scaling factor to the file given by 'open(2)'. opt=1 means small line segments will be connected to longer straight lines, this reduced storage usage, increases computing time.

#bdmesh x y dmode factor opt rcval 'spline' writes the boundary of the mesh of 'spline' in the format given by '#pmode' to location x y in dmode-form (2: plane, 3: perspective view) with additional scaling factor to the file given by 'open(2)'. opt=1 means small line segments will be connected to longer straight lines, this reduced storage usage, increases computing time. rcval denotes the boundary part.

#plot amode limit ox oy ndelta dmode factor colmode dmeshmode writes a surface plot of the solution in the format given by '#pmode' to the file given by 'open(2)'. 'ndelta' gives the number of subdivisions of each element for plotting. The viewpoint and view angle given before are used for 3d plots. 'dmode=2' means the plot should be done in the plane, whereas 'dmode=3' means the plot will be performed as a 3D-plot even in the case of a screen or 2D-FEM mesh. 'amode' is a string describing the data which have to be plotted, i.e., the first entry is a spline name, e.g. 'u'

u	scalar function deformation plot if vector valued
u:re	real part of a complex function
u:im	imaginary part of a complex function
u:abs	absolute value of a complex function, resp. length of a vector
u:x	gives the x-component of a vector valued function
u:grd	gives the gradient

The suffices can be combined, e.g. 'u:grd:abs' for the absolute value of the gradient of a scalar function.

#bdplot amode limit ox oy ndelta dmode factor colmode dmeshmode bdrc(0:3) writes a surface plot of the 3d volume-solution in the format given by '#pmode'. The surface part used is determined by the rc-values in bdrc.

range(string='..') reads all the splines and formats given in string (separated by ',') and determines the extension in x and y as well as the minimum and maximum values.

plch generates the list of lines describing the mesh from a node list. This list of lines is optimized, i.e. double entries are eliminated and line segments have been connected if possible.

#pl2 amode limit ox oy writes the line list of a screen in the format given by '#pmode' to the file given by '#out2'. If we want to have output for Plotmtv or UNIRAS a contour plot of the solution is generated.

#pl2p spline writes the polynomial degrees of a mesh in the format given by '#pmode' to the file given by 'open(2)'.

#pl3 x y z wx wy wz ox oy writes the line list of a 3D-mesh in the format given by '#pmode' to the file given by '#out2'. 'x y z' are the coordinates of the view point and 'wx wy wz' the view angles, whereas 'ox oy' are the offset for the resulting plot.

#plrci dmode spline draws the rci-values of the given spline in 2d- or 3d-mode (dmode=2 or 3) to their effective location to the file given by 'open(2). In 3d dmode=0 means, that the rci-values are printed and dmode=1 draws the node numbers.

#plrn factor spline draws the normal directions of faces (or edges) of the given spline. in the format given by '#pmode' to the file given by 'open(2)'. With factor you can scale the length of the unit-length normal vectors.

#plor writes the orientation of the mesh elements of a 3D-boundary mesh resp. a 2d-FEM mesh in the format given by '#pmode' to the file given by '#out2'.

#plsg writes the location of singularities of a 3D-boundary mesh resp. a 2d-FEM mesh in the format given by '#pmode' to the file given by '#out2'.

#plpn factor spline draws the element numbers of the given spline. factor has no meaning in the moment.

#plref ox oy dmode pencolour writes the boundaries of new mesh elements (the refinement structure, ref-information) given by the '#ref' command in the Xfig-color given by pencolour to the file given by '#out2'.

#plrc factor spline draws the connections and element numbers of the given spline.

#drawscala x y dx dy draw the scale of colours in the given box.

#block x y dx dy r g b writes a colored block at the coordinates (x,y) with size (dx,dy) in the RGB colors r,g,b.

#text x y str writes the text string str to the coordinates (x,y).

#line x y dx dy draws a line from (x,y) to (x+dx,y+dy).

#arrow x y dx dy draws an arrow pointing from (x,y) to (x+dx,y+dy).

#line3 ax ay az bx by bz draw line from a to b which is project to the viewplane.

g>p use the geometry mesh as usual mesh.

s>p use the singularity mesh as usual mesh.

#eff. exhnorm reads the exact energy norm and computes the efficency of an adaptive algorithm using the error in the energy norm and the value of the error indicator.

ekap. exkap reads the exact capacity and computes the error and convergence rate for meshes which have been read by *maigraf* before. The results are written to the file given by '#out2' in a format readable by GNUPLOT.

eeno. exhnorm reads the exact energy norm and computes the error and convergence rate for meshes which have been read by *maigraf* before. The results are written to the file given by '#out2' in a format readable by GNUPLOT.

expe. rate extrapolates a sequence of energy norms using the convergence rate 'rate'.

#new sets the internal counters back to zero, so that a new sequence of norms can be read.

mout prints a screen mesh and the solution vector using UNIRAS if possible.

tout ox oy prints the 2D-BEM mesh in the format given by '#pmode' to the out2-file.

2.14 Commands of *maicont2*

maicont2 solves Signorini contact problems. Therefore it uses additional commands to deal with this special kind of boundary value problem.

schur mode niter computes the discrete Signorini-Operator by a Schur-complement. If mode=0 the original Galerkin matrices will be left intact and we need a lot of additional memory. If mode=1 the original Galerkin matrices will be partly destroyed and we will need less additional memory. If mode=2 the original Galerkin matrices will be partly destroyed and we will need no additional memory.

#cmode mode quiet sets the contact mode.

mode	contact mode
0	lower bound
1	upper bound
2	lower & upper bound

If quiet<0 no full text output will be generated.

#cmode. writes the current mode of the contact problem (lower bound, upper bound, two-sided obstacle).

lft gql ltyp load bdtyp starts the computation of the right hand side. The integer 'gql' gives the number of nodes of the numerical integration needed for arbitrary functions. The integer 'ltyp' gives the index number of the function used as boundary data. 'load' is a scaling factor for the boundary data. 'bdtyp' gives the index function for the contact data.

sor eps omega solves the contact problem using the SOR-scheme with projection. 'omega' is the damping parameter.

polyak eps mdc mit quiet solves the contact problem using the Polyak-algorithm with preconditioner 'mdc'. If mdc=0 no preconditioner is used. If mdc=1 the BPX preconditioner and if mdc=2 the hierarchical preconditioner is used.

mlev eps nu1 nu2 mu mtop omega mdc mds mnum hpmodus stp grflag abrflag expo quiet solves the contact problem using the Monotone Multigrid algorithm. Here mdc=0 denotes primitive restriction and mdc=1 denotes restriction a la Kornhuber.

check checks whether the solution respects the Signorini conditions.

checksig eps quiet checks whether the Signorini condition is satisfied with tolerance eps.

#obs. write the nodal based information of the contact problem to the output file. The format is: #obs dof bmode followed by 'coefficient number', 'contact condition' (0: Dirichlet, 1: Neumann (free), 2: Contact), 'lower bound', 'upper bound'.

#obs dof bmode followed by dof data lines reads the contact information from the input file.

2.15 Outdated commands

#rc n p triflag generates a uniform mesh on the given geometry with n elements per edge of an macro cell and polynomial degree p. In 2d triflag is 3 for triangles and 4 for a quadrilaterals. In 3d triflag is 4 for tetrahedral, 5 for pyramid, 6 for prism and 8 for hexahedral elements. The command is outdated and replaced by mesh(type='uniform',...).

#gr n p beta triflag generates an algebraically graded mesh on the given geometry with n elements per edge of a macro cell with grading parameter beta (1.0 ...4.0) and polynomial degree p. In 2d triflag is 3 for triangles and 4 for a quadrilaterals. In 3d triflag is 4 for tetrahedral, 5 for pyramid, 6 for prism and 8 for hexahedral elements. The command is outdated and replaced by mesh(type='graded',...).

#gr2 n p beta triflag generates a modified algebraically graded mesh on the given geometry with n elements per edge of a macro cell with grading parameter beta (1.0 ...4.0) and polynomial degree p. In 2d triflag is 3 for triangles and 4 for a quadrilaterals. In 3d triflag is 4 for tetrahedral, 5 for pyramid, 6 for prism and 8 for hexahedral elements. The command is outdated and replaced by mesh(type='Kgraded',...).

#go n n0 sigma mu triflag generates a geometrically graded mesh on the given geometry with n-n0 elements per edge of a macro cell with grading parameter sigma (typical 0.17) and polynomial degree rate mu. In 2d triflag is 3 for triangles and 4 for a quadrilaterals. In 3d triflag is 4 for tetrahedral, 5 for pyramid, 6 for prism and 8 for hexahedral elements. The command is outdated and replaced by mesh(type='geometrical',...).

3 The *unimesh*-family

The folder uni/ contains programs which are optimized for uniform meshes on simple geometries, i.e. slits, squares and cubes. Currently are available

name	operator	version	FEM/BEM
unimesh	Laplace (D)	h-version	3D-BEM
unilapn3	Laplace (N)	h-version	3D-BEM
unilap2	Laplace (D)	h-version	2D-BEM
unilapn2	Laplace (N)	h-version	2D-BEM
unihpn2	Laplace (N)	hp-version	2D-BEM
unifem1	Laplace	h-version	1D-FEM
unifem2	Laplace	h-version	2D-FEM
unifem3	Laplace	h-version	3D-FEM
uniflm3	Lamé	h-version	3D-FEM

Available is also the program *grdlapn2* as an extension of *unilapn2* to algebraically graded meshes.

In all or some programs are the following preconditioners/solvers available

Conjugate Gradient algorithm
Conjugate Gradient algorithm with diagonal preconditioning
CG with BPX-preconditioner
CG with multilevel additive Schwarz preconditioner
CG with 2-level additive Schwarz preconditioner
CG with variable-2-level (Hh) additive Schwarz preconditioner
CG with wavelet-additive Schwarz preconditioner
CG with hierarchical additive Schwarz preconditioner
CG with multigrid as preconditioner
Multigrid as iterative solver

In all cases the condition number of the preconditioned system is estimated by the Lanczos-algorithm.

A Global variables

The following global variables contain the results of some computations for further using by BCL-code.

Name	Meaning
DOF	Total number of degrees of freedom
DOF??	Degrees of freedom of spline ??
ENO	Energy norm
ENOR	Real Part of Squared Energy norm for complex formulations
ENOI	Imaginary Part of Squared Energy norm for complex formulations
ENOERR	Error in energy norm
JU, ERRJU	
CAP	Capacity
COND	Condition number
ERR, ERR[1], ERR[2]	Error estimation in some norm
ERREST	Total Error indicator
NORM	Norm of solution vector
ITER	Number of iterations
SEC	Execution time in seconds
LMIN	Minimal eigenvalue
LMAX	Maximal eigenvalue
EWPOS	Number of positive eigenvalues
EWNEG	Number of negative eigenvalues
EWZERO	Number of vanishing eigenvalues
EWNMIN	Minimal negative eigenvalue
EWNMAX	Maximal negative eigenvalue
EWPMIN	Minimal positive eigenvalue
EWPMAX	Maximal positive eigenvalue
STRESS	Stress coefficient
XMAX, XMIN, YMAX, YMIN	Extension of object in x-,y-direction
ZMAX, ZMIN	Smallest and largest value
ANTIP, CURLP, DGDT, DIFXI, DIVPF, GMPN, gmtg, GUMAX, JPHI, LAMS, LMTL, PHIXI, PMNPHI, PNWLAM, PTFREE, PTJUMP, PTLAMS, PTXIS, SGUMAX, SIGMAHT, THT, UMPHI, XIMTXI, ZETA	Parts of error indicators

COARSE CONTIT DELTAUF EMAX INTEN INTIT IREST ITER LSQNLC LSQNLCE LSQNLCR MAXCOARSE MAXCORR MAXCURR MAXEN MAXNORM MAXRHOBD MEAN MINCORR MINEN MXBLCK NABLA NEWTON NORM NORMP OREST

SPECTRALNORM STRESS TOTCURR UZIT VARPHI WIREBLCK

References

[1] M. Maischak, The analytical computation of the Galerkin elements for the Laplace, Lamé and Helmholtz equation in 2D-BEM. Preprint 95-15, DFG-Schwerpunkt Randelementmethoden.

[2] M. Maischak, The analytical computation of the Galerkin elements for the Laplace, Lamé and Helmholtz equation in 3D-BEM. Preprint 95-16, DFG-Schwerpunkt Randelementmethoden.

[3] M. Maischak, Galerkin elements of FEM in 2D and 3D. Preprint, Institut für Angewandte Mathematik, Universität Hannover.

[4] M. Maischak, Technical manual of the program system maiprogs. Preprint, Institut für Angewandte Mathematik, Universität Hannover.

[5] M. Maischak, B.O.N.E. – Book of Numerical Experiments. Preprint, Institut für Angewandte Mathematik, Universität Hannover.

Index