Documentation for rbMIT Software: I. Reduced Basis (RB) for Dummies

D.B.P. Huynh, N.C. Nguyen, A.T. Patera, G. Rozza

February 24, 2009

1 Directory Structure and "Problem Management"

You have already downloaded the `rbMIT_Part_II_and_IV_DATE` folder from our website (to your MATLAB® directory). It contains the documentation you are reading, the `rbMIT_Library` and `rbMIT_Aux` folders that contain the rbMIT software, a folder `rbUfiles` that contain `rbU_probname.m` files (see below), and the license agreement. All the RB activity will take place in the `rbMIT` directory. Before doing anything, first launch the function

```
>>addrbMITpath.m
```

in the `rbMIT` directory.

For each new problem you wish to create, choose a unique problem name `probname` and create a new (sub)folder named `probname` in the `rbMIT` directory. For example, for the problem named `probname = 'fin'`, the folder should be named `fin`. (This `fin` folder is in fact included in the `rbMIT` directory that you have downloaded.)

In the probname folder, create the "rbU".m file `rbU_probname.m`. Easiest is to cut and paste an `rbU_blah.m` file provided in the `rbUfiles` folder, change the name of the file to `rbU_probname.m`, and then modify appropriately according to Section 4. (For the problem named `probname = 'fin'` the `rbU_fin.m` file is included in the `rbUfiles` folder but also directly in the `fin` folder provided.) Once the rbU file is ready, you can do the Offline execution (Section 2) and Online execution (Section 3).

Note that there are really two names associated with a problem: the name of the problem, `probname`, and the name of the scalar output you wish to evaluate, `outputname`. In the course of Offline execution (see Section 2), the software will automatically generate a (sub)folder named `outputname` in the folder `probname` (within the directory `rbMIT`). You may on occasion need to access data in the `outputname` folder; however, make sure you are

always in the probname folder when you do the Offline and Online execution for the problem named probname.

We suggest that you first 'do' a few problems for which we provide rbU files: to begin do the probname = 'fin' problem with outputname = 'flux' directly from the fin folder you will find in the rbMIT directory that you have downloaded; then do a few problems from the documentation (for example, create the folder named tailpiano in the rbMIT directory, then copy the rbU_tailpiano.m file from the rbUfiles folder to the tailpiano folder). In each case, once you have the probname folder and rbU_probname.m file set up, proceed to Sections 2 and 3 below.

To create new rbU files for new problems, see Section 4.

2 Offline Execution

To run the problem named probname, put yourself in the probname folder (which in turn must be in the rbMIT directory, as explained above). Then at the MATLAB® command prompt >> type rbU_probname; i.e.,

```
>>rbU_probname
```

and sit back and wait.

Note in the rbU file you can specify plotdemo = 1 (lets you see what's going on, pausing often for you to hit a key and continue) or plotdemo = 0 (does everything secretively); in many of the rbU files provided in Section 4 we have set plotdemo = 1, so you will need to change this if you prefer the non-verbose form.

In the Offline stage there is a fair amount of computing, which for real problems could easily take not only minutes but hours. To facilitate fast and less painful learning, we have set the default value of femOPT.maxsize to 250 (related to the dimension of the underlying FE approximation); however, for actual "production" calculations, the user will undoubtedly wish to increase femOPT.maxsize in order to obtain higher FE resolution. The Offline stage will generate a bunch of .m and .mat files, the contents of which you, as a Dummy, are not obligated to understand.

3 Online Execution

3.1 Output Evaluation

The Online stage is your reward: for the problem named probname, you can evaluate the output named outputname extremely quickly for different val-

ues of the parameter `[mu1, mu2,...,muP]` (for P parameters) in the input parameter domain defined by

[mu1_min, mu1_max] × [mu2_min,mu2_max] × [muP_min,muP_max].

Note the user sets the parameter ranges in the rbU file: `mu_min = [mu1_min,...,muP_min]` and `mu_max = [mu1_max,...,muP_max]` (see Section 4.1.1(*iv*)).

Make sure you are positioned in the `probname` folder (not in the `outputname` subfolder) before continuing. In the Online execution the data are loaded as global variables in the workspace to improve efficiency during computation. If you want to delete these variables just do a `clear all`.

Now do

```
>>[sN, DeltaN] = Online_RB('probname',[mu1,mu2,...,muP],'outputname')
```

for example, for the problem named `probname = 'fin'` with output named `outputname = 'flux'`, do

```
>>[sN, DeltaN] = Online_RB('fin',[.1,3],'flux')
```

Note `[mu1,mu2,...,muP]` is the value of the parameter for which you wish to evaluate the output: this parameter value must be in the parameter domain as specified in the rbU file. Here `sN` is the RB prediction for the output, and `DeltaN` is a rigorous *a posteriori* error bound for the difference between the RB output prediction and finite element (FE) output prediction.

We emphasize that the RB approximation is an approximation to an FE approximation; the mesh for the latter is printed out if `plotdemo = 1`. You can change the FE resolution (either automatic adaptive, or a selected number of regular refinements) by setting the flag `femOPT.maxsize` (default = `250`) and `femOPT.refine` (default is `'adaptive'`, option is `'subdivision'`) in the rbU file, as described in Section 4. The better the FE approximation, the more expensive the Offline stage; however, the computational time for the Online stage is independent of the resolution of the FE approximation.

3.2 Visualization

The visualization is not as quick as the Online evaluation. In particular, the computational time for visualization is not independent of the resolution of the FE approximation; however, the visualization is still typically much faster than direct FE solution/rendering.

In general, you should remain exclusively in the `probname` folder during visualization. Also in this case the data are loaded as global variables to improve efficiency and avoid reloading many times. To clear the workspace just do a `clear all`.

To visualize the RB approximation to the FE field for the problem named `probname` for the parameter value `[mu1,mu2,...,muP]` (which must be in the parameter domain), just do

```
>>Vis_RB('probname',[mu1,mu2,...,muP])
```

which will present the geometry (which can be parameter-dependent), a rendering of the field, and a rigorous *a posteriori* bound for the error in the RB field relative to the FE field. (The error is an integral of the error^2 and error_in_the_derivative^2 over the domain.)

4 The rbU File

There are several components to the rbU file. We have already discussed `probname`, `outputname`, `plotdemo`, and `femOPT`. These overall descriptors must go at the beginning of the rbU file. For example, for our fin problem

```
probname = 'fin'
outputname = 'flux'
plotdemo = 1
femOPT.refine = 'adaptive'
```

(Note if the user does not set the `plotdemo` or refinement flag, the software will choose for the user. However, `probname` and `outputname` are required as the first lines of the rbU file.)

We now turn to the more technical inputs. We first consider the case of a scalar equation, then the case of a vector equation (linear elasticity).

4.1 Scalar Equation

There are two supported problem types for the scalar problems:

`probtype = 'TH'`: for normal case;

`probtype = 'TH2'`: for axisymmetric case.

The default value of probtype is `probtype = 'TH'`.

4.1.1 Straight Edges

We shall first describe the case with straight edges, using `probname = 'fin'` as our example (`probname = 'fin'`, `outputname = 'flux'`, `plotdemo = 0` or `1`, `femOPT.refine = 'adaptive'`).

(*i*) *Geometry*

There is a set of points

```
points = '[1/2,0; 1/2, mu2/2; 1/2, mu2; -1/2, mu2; -1/2,
mu2/2; -1/2, 0]';
```

note only the parts between the '`[` and `]`' will change from problem to problem. Note the points can be constants — better to use fractions rather than decimals in order to speed up the symbolic processing — or standard functions of `mu1,...,muP` expressed in the usual MATLAB® lexicon.

There is a set of edges

```
edge = [1,2;2,3;3,4;4,5;5,6;6,1];
```

(no quotes this time). Each pair of integers refers to indices of the points array: so the `1,2` pair in edge defines a straight line that connects `(1/2,0)` (the first point in the points array) and `(1/2,mu2/2)` (the second point in the points array); similarly, the `6,1` pair defines a straight line that connects `(-1/2,0)` (the sixth point in the points array) and `(1/2,0)` (the first point in the points array).

There are a number of `geometry{ }` cell arrays (note the { } brackets rather than `( )` parentheses). Each `geometry{ }` is a list of edges:

```
geometry{1} = [1,2,3,4,5,6];
```

in our example there is only one geometry array. The above statement defines a boundary: we start with edge 1, then edge 2 (which must share one endpoint with edge 1), then edge 3, ..., until we arrive at edge 6 (which must share an endpoint with edge 1 — forming a complete "cycle"). From this data we infer what we shall call `ProtoRegion{1}`: the interior (defined in the obvious way) of the boundary. (The user does not input the `ProtoRegions`; the `ProtoRegions` are deduced from the user-input geometry edge lists.)

Each `geometry{k}` will define a `ProtoRegion{k}`. We need different `ProtoRegions` (and hence different geometry) in order to be able to define different PDE coefficient/physical properties in different parts of the domain. The intersection of `ProtoRegion{k}` with `ProtoRegion{j}` can be either null, all of `ProtoRegion{k}` (in which case we say that `ProtoRegion{k}` is inside `ProtoRegion{j}`, or all of `ProtoRegion{j}` (in which case we say that `ProtoRegion{j}` in inside `ProtoRegion{k}`).

There is a `gflag` array,

```
gflag = [1];
```

if `gflag(k) = 1` then the `ProtoRegion{k}` corresponding to `geometry{k}` (here `k = 1` since for the fin there is only one geometry list) is part of the final domain (i.e., presence of "material"); if `gflag(k) = 0` then the `ProtoRegion{k}` corresponding to `geometry{k}` is a hole (i.e., absence of "material").

We now construct the final domain on which the PDE is defined. In particular, for each `ProtoRegion{k}` that is not a hole we subtract all `ProtoRegions` that are either (a) inside `ProtoRegion{k}` (as defined above), or (b) a hole (as indicated by `gflag`). The resulting `ProtoRegion{k}` — with nested `ProtoRegions` and holes removed — is now denoted `Region{k}`. (Note if `ProtoRegion{j}` is a hole, then it no longer plays any role in the rbU file: the `j` index is no longer relevant. Thus, if there are three geometry lists, and `gflag = [1,0,1]`, `Region{1}` (derived from `ProtoRegion{1}`), and `Region{3}` (derived from `ProtoRegion{3}`) define the final problem domain.)

(ii) *PDE Coefficients*

In each `Region{k}`, we solve

$$-\frac{\partial}{\partial x_i}\left(c\{k\}_{ij}\,\frac{\partial u}{\partial x_j}\right) + U_i\{k\}\,\frac{\partial u}{\partial x_i} + r\{k\}u = f\{k\} \text{ in } \texttt{Region\{k\}}$$

where `u` is the field variable, `x = (x_1,x_2)` is the spatial coordinate, `c{k}` is a 2×2 SPD tensor diffusivity, `U{k} = (U_1,U_2)` is a velocity, `r{k}` is a non-negative scalar, and `f{k}` is a scalar. All of these quantities can be polynomial functions of `x` and may depend on the parameter `[mu1,mu2,...,muP]`.

These quantities are input in the rbU file via the `kappa{k}` (for `Region{k}`) cell array. The `(1,1),(2,1),(2,1),(2,2)` entries of `kappa{k}` are `c{k}`; the `(3,1),(3,2)` entries of `kappa{k}` are `U{k}`; and the `(3,3)` entry of `kappa{k}` is `r{k}`. The entries for `kappa{k}` for `k` corresponding to "hole" `ProtoRegions` should be set to zero.

For the fin example,

```
kappa{1} = '[1 0 0; 0 1 0; 0 0 0]';
```

this means that in our single region (the entire domain), the equation is

$$-\nabla^2 u = f\{1\} \text{ in } \texttt{Region\{1\}}$$

corresponding to the simple Laplacian.

The f{k} are specified via the fareaload input in rbU. In particular, fareaload consists of (Region number k, f{k}) pairs; the user need only include those pairs for which f{k} is non-zero. For the fin problem, f{1} = 0, and hence no fareaload input is required. Values of f{k} may depends on the parameter [mu1,mu2,...,muP] and can be polynomial functions of x. For example, f{1} = mu1*x + mu2*x*y^2.

Remark. Please be aware that if Matlab is calling Mupad instead of Maple, for its higher versions, Mupad cannot convert expressions other than $*, /, +, -$, so if we declare points = '[1./mu1,0]' there will be an error ('[1.0/mu1,0]' is fine) and also do not use math expression containing $./, .*$ in the geometry declarations (points, load, kappa, etc).

Pleease try to avoid floating point number (real number) if you can, instead using fractional number (if we have 0.3, we better express it as 3/10). This helps to improve the speed in the computing procedures in the symbolic pre-processor.

(*iii*) *Boundary Conditions*

Continuity of the field and the flux (the normal component of c{k}_ij du/dx_j) is automatically imposed on all internal interfaces — defined as boundaries between Regions. We now turn to boundary conditions.

Dirichlet conditions are specified via the dirichlet input in rbU. In particular, dirichlet consists of (edge number, value of the field on edge number) pairs, where the value of the field on edge number must be a constant independent of parameter [mu1,...,muP] and x. Note the edge numbers must correspond to edges on the boundary of the domain. Note also that if there are no (inhomogeneous or homogeneous) Dirichlet boundary conditions then the dirichlet input may be absent from the rbU file.

For the fin problem,

```
dirichlet = '[6,1]';
```

which indicates that on edge 6 (which from the edge array and points list we see corresponds to the base of the fin) we impose Dirichlet conditions u = 1.

Turning now to natural boundary conditions, homogeneous Neumann (flux) boundary conditions require no action. For inhomogeneous Neumann conditions or Robin conditions, we impose the general boundary

conditions

$$n_i \, c\{k\}_{ij} \, \frac{\partial u}{\partial x_j} + g_1(u - g_2) = g_3 \ ,$$

where `n_i` is the unit normal, `g1` is the Robin coefficient (possibly zero), `g2` is the "sink" field value (possibly zero), and `g3` is the flux (possibly zero). The coefficients `g1`, `g2`, and `g3` may all be polynomial functions of `x` and/or involve the parameter `[mu1,...,muP]`.

The coefficients `g1`, `g2`, `g3` are specified via the `nload` input in the rbU file. In particular, `nload` consists of (edge number, value of `g1` on edge number, value of `g2` on edge number, value of `g3` on edge number) 4-tuples. (Again, if `g1` = `g2` = `g3` = `0` on an edge, this edge need not be listed in `nload`.) Note the edge numbers must correspond to edges on the boundary of the domain. Note also that if there are no inhomogeneous Neumann or Robin boundary conditions, then the `nload` input may be absent from the rbU file.

For the fin problem,

```
nload = '[1,mu1,0,0;2,mu1,0,0;4,mu1,0,0;5,mu1,0,0]'
```

which indicates that on edges 1, 2, 4, and 5 (which from the edge array and points list we see corresponds to the two sides of the fin) we impose Robin conditions with coefficient `g1 = mu1`. (Note on edge 3 we impose homogeneous Neumann conditions, which are automatic; thus edge 3 does not appear in either the `dirichlet` or `nload` rbU inputs.)

(*iv*) *The Parameter Domain*

The key input is the vector `mu_min = [mu1_min,mu2_min,...,muP_min]` and the vector `mu_max = [mu1_max,mu2_max,...,muP_max]`: the reduced basis is constructed for, and can be queried for, values of `mu1` between `mu1_min` and `mu1_max`, values of `mu2` between `mu2_min` and `mu2_max,...`, and values of `muP` between `muP_min, muP_max`. (Note that `P`, the number of parameters, is deduced from the input file by the length of the `mu_min` and `mu_max` vectors.)

For the fin problem,

```
mu_min = [0.02,2];
mu_max = [0.5,8];
```

which means that `0.02` $\leq$ `mu1` $\leq$ `0.5` (this is the Robin coefficient) and `2` $\leq$ `mu2` $\leq$ `8` (this is the length of the fin).

In addition, there are two more technical inputs related to the geometry transformations (`muref`) and *a posteriori* error bounds (`mu_bar`). For Dummies, it is perhaps easiest to set both `muref = mu_bar = 1/2(mu_min + mu_max)` for each component of the P-vector; in fact, `muref` and `mu_bar` can be different, and can be any parameter value in the parameter domain. For the fin problem,

```
muref = [0.1,4];
mu_bar = [0.1,4];
```

which is in fact the log-mean for first parameter (often a good choice for property and boundary condition parameters) and the arithmetic mean for the second parameter (often a good choice for geometric parameters).

(v) *Outputs*

For outputs we may consider the integral of the field over selected regions or over selected boundary edges. To specify an output over selected regions we set the `oareaload` input in rbU. In particular, `oareaload` consists of (Region number k, factor to be included in the integral over `Region{k}`) pairs; the user need only include those pairs for which the factor — the term in front of the integral — is non-zero. The factor may depend on the parameter and on `x` coordinates.

To specify an output over selected edges we set the `oload` input in rbU. In particular, `oload` consists of (edge number, factor included in the integral over edge number) pairs; the user need only include those pairs for which the factor — the term to be included in the integral — is non-zero. All edges must be boundary edges. The factor to be included in the integral may be a polynomial function of `x` and involve the parameter.

Finally, we may ask for the integral of the flux, `n_i c{k}_ij du/dx_j`, over selected boundary edges. To specify this output we set the `dLIFT` input in rbU. In particular, `dLIFT` consists of (edge number, prefactor to integral of flux over edge number) pairs; the user need only include those pairs for which the prefactor — the term in front of the integral — is non-zero. All edges must also appear in the `dirichlet` rbU input list (corresponding to edges on which we impose Dirichlet boundary conditions on `u`). The prefactor may depend on the parameter but not on `x`.

We note that an output must be one of the three options above; com-

binations are not allowed. (Multiple outputs are possible ... but not for Dummies.) For the fin problem, we set

```
dLIFT = '[6,1]';
```

which indicates that the output is the integral of the flux over edge 6 (the fin base) multiplied by prefactor 1.

(*vi*) *Axisymmetric Model*

The axisymmetric problem is specified by the `probtype` input (`probtype = 'TH2'`) in the rbU. The descriptions (geometry, coefficients, boundary conditions) for axisymmetric models are similar to that of normal models. However, there are some special requirements:

- The axis of symmetry is the `x_2` axis.
- The `x_1` and `x_2` directions represent the radial and axial direction, respectively.
- Negative `x_1`-coordinates of points are not permitted.

For axisymmetric model, outputs that are specified by `oareaload` and `nload/dLIFT` now correspond to integral over volumes (for `oareaload`) and surfaces (for `nload/dLIFT`) instead of regions or edges. Specifically, `oareaload` output corresponds to integral of the field over a 1-radian segment of the volume, and `nload/dLIFT` output corresponds to integral of the field over a 1-radian segment of the surface. To get the value integrated over the full volume/surface, the user must multiply the obtained results by `2*pi`.

(*vii*) "*Tailer*"

All rbU files end with the same tailer, which launches an irreversible sequence of events that will either successfully terminate upon completion of the Offline stage — or confuse MATLAB®, erase your hard disk, and crash your computer. The tailer, to be put verbatim at the end of every rbU file, is

```
%%%%%%% no user input required beyond this point
if exist('plotdemo')
plotdemo = 0;
end
save rbU ;
copyfile('../rbMIT_Aux/Step1_coer_noncompliant.m',...
...strcat(probname,'_','Step1_coer_noncompliant.m'))
```

```
eval(strcat(probname,'_','Step1_coer_noncompliant'))
%%%%%%%
```

Note that it is also often prudent to put a "clear" statement at the very beginning of your rbU files — but NOT at the end of your rbU file.

4.1.2 Curvy-Arc Edges

In the case in which the logical edges (point pairs) are connected by straight-lines, the geometry description above suffices. We consider here the case in which the point pairs are connected either by elliptical arcs or more general parametrized curvy arcs.

We first discuss the case of elliptical arcs. Each elliptical arc is a member of a family of elliptical arcs (this saves the user from needing to re-enter the same curve description for many geometrically similar edges). All edges that belong to a particular family are described by the common equation

$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = \begin{bmatrix} O_1 \\ O_2 \end{bmatrix} + \begin{bmatrix} \cos\phi & -\sin\phi \\ \sin\phi & \cos\phi \end{bmatrix} \begin{bmatrix} \rho_1 & 0 \\ 0 & \rho_2 \end{bmatrix} \begin{bmatrix} \cos t \\ \sin t \end{bmatrix}$$

where `t` is a parametrization of the curve. (This parametrization of course refers to "`t`" and not our parameter `[mu1,...,muP]`.) Here $[O_1, O_2]$ is the center, ϕ is an angle of rotation, and ρ_1 and ρ_2 are dilations: all of these quantities may depend on the parameter `[mu,...,muP]`. Each edge that belongs to this family is then defined by (different) "start" and "end" values for the parameter `t`. It is thus clear that these elliptical arcs correspond (appropriately enough) to segments of ellipses of prescribed location, orientation, and major/minor axes.

The user provides this information through the `curvedat` and `tarclist` inputs in the rbU file. Here (say, for two families)

```
curvedat = '[O1 for family 1, O2 for family 1, rho_1 for family 1,
rho_2 for family 1,
              phi for family 1, cos(t), sin(t); O1 for family 2, O2 for
family 2,
              rho_1 for family 2, rho_2 for family 2, phi for family 2,
cos(t), sin(t)]';
```

note that the last two entries for each family are always (for elliptical arcs) `cos(t)` and `sin(t)`, respectively. Then `tarclist` is a list of (edge number, family to which edge belongs, start value of `t`, end value of `t`) 4-tuples for each edge which is elliptical; only edges which are actually elliptical

need be included in the `tarclist` data. Note that the points list must be consistent with `curvedat` and `tarclist`: for the start and end values of `t`, the `curvedat/tarclist` description of an elliptical-arc edge must agree with the corresponding points entries. See rbU examples `rbU_elashole`, `rbU_elascoat`, `rbU_crack2`, `rbU_cantilever2`, and `rbU_elas4` for various examples.

We note that there are some restrictions on boundary conditions and outputs for elliptical-arc edges. We may impose (constant) homogeneous or inhomogeneous Dirichlet conditions and homogeneous Neumann (flux) conditions on any straight or elliptical-arc edge; however, we may only impose inhomogeneous Neumann conditions and Robin conditions only on straight or circular-arc edges (not general elliptical-arc edges). Similarly, we may consider "`oload`" outputs only for straight or circular-arc edges (not general elliptical-arc edges).

Finally, we consider the case of general "curvy"-arc edges. In fact, the treatment is identical to elliptical-arc edges, except that `cos(t)`, `sin(t)` is now replaced by general functions `h1(t)`, `h2(t)` selected by the user. See `rbU_tailpiano` as an example.

4.2 Vector Case

We shall describe only Problem type, PDE Coefficients, Boundary Conditions, and Outputs in this Section; see the Scalar case for Geometry (4.1.1*i*), Parameter Domain (4.1.1*iv*) and Tailer (4.1.1*vii*).

The problem type description variable — `probtype` — is required are must be specified for vector problems. Currently there are five supported problem types:

`probtype = 'LE1'`: for isotropic plane strain;

`probtype = 'LE2'`: for isotropic plane stress;

`probtype = 'LE3'`: for orthotropic plane strain;

`probtype = 'LE4'`: for orthotropic plane stress;

`probtype = 'LE5'`: for axisymmetric elasticity.

There is no default value for `probtype`. The user must specify the problem type, i.e., set the `probtype` variable to the correct value at the beginning of the rbU file (just after `probname`).

(*i*) *PDE Coefficients*

In each `Region{k}`, we solve the linear elasticity PDE

$$-\frac{\partial}{\partial x_j}\left(C_{ijkl}\{\mathrm{k}\}\,\frac{\partial u_k}{\partial x_l}\right)+r_i\{\mathrm{k}\}\delta_{ij}u_j = b_i\{\mathrm{k}\}, \quad i=1,2, \texttt{ in Region\{k\}}\,,$$

where `u` is the (displacement) field variable, `u = (u_1,u_2)`, `r_1`, `r_2`, are non-negative scalars, `b_1`, `b_2` are scalars, and `C_ijkl` is the effective elasticity tensor corresponding to the problem type.

The tensor `C_ijkl{k}` depends on the material properties as prescribed by `matprop{k}` in the rbU file. The order and descriptions of `matprop{k}` depends on the problem type and is given below:

- `probtype = 'LE1', 'LE2', and 'LE5': E, nu, r_1, r_2`
- `probtype = 'LE3': E_11, E_22, E_33, nu_12, nu_23, nu_31, G_12, r_1, r_2`
- `probtype = 'LE4': E_11, E_22, nu_12, G_12, r_1, r_2`

where `E` denotes the Young's modulus, `nu` denotes the Poisson ratio, and `G` denote the shear modulus. Unspecified `r_1` and `r_2` values will automatically set `r_1=0` and `r_2 = 0` for the particular region. For example, for

```
probtype = 'LE3';
matprop1 = '[1.0, 1.1, 1.2, 0.3, 0.2, 0.1, 1]';
```

corresponds to a orthotropic plane strain linear elasticity problem with `E_11 = 1.0, E_22 = 1.1, E_33 = 1.2, nu_12 = 0.3, nu_23 = 0.2, nu_31 = 0.1, G_12 = 1, r_1 = 0` and `r_2 = 0`. For `probtype = 'LE1'` and `probtype = 'LE2'`, the entries of `matprop{k}` may be polynomial functions of `x` and involve the parameter. However, for `probtype = 'LE3'` and `probtype = 'LE4'`, the entries of `matprop{k}` may depend on the parameter but not on `x`.

The `b_1` and `b_2` values are specified via the `fareaload` input. In particular, `fareaload` consists of (`Region number k`, `b_1`, `b_2`) 3-tuples; as in the Scalar case, the user need only to include the non-zero `b_1/b_2` pairs. Note `b_1` and `b_2` may be polynomial functions of `x` and involve the parameter.

(*ii*) *Boundary Conditions*

The Dirichlet conditions are specified via the `dirichlet` input. Dirichlet consists of (edge number, label, value of the field on edge number),

where label can be either "x" or "y" and specifies the component of the field to be constrained. For example,

```
dirichlet = '[1, x, 0; 1, y, 0; 2, x, 0]'
```

means that we impose u_1 = u_2 = 0 on the first edge and u_1 = 0 on the second edge. As in the Scalar case, the edge numbers must correspond to edges on the boundary of the domain, and the Dirichlet values must be constants.

Homogeneous Neumann boundary conditions require no input. For inhomogeous Neumann conditions, we adopt the input form from nload (edge, f_1, f_2), which corresponds to coefficients of

$$C_{ijkl}\,\frac{\partial u_k}{\partial x_l}\,\mathrm{n}_j \;=\; f_1\,\mathrm{e}_i, \qquad i = 1, 2\ ,$$
$$C_{ijkl}\,\frac{\partial u_k}{\partial x_l}\,\mathrm{n}_j \;=\; f_2\,\mathrm{n}_i, \qquad i = 1, 2\ ;$$

here n and e correspond to the normal and tangential vectors on the defined edge. We adopt the convection that the normal vector is the vector pointing outward the geometry, and the order of e and n follows the counter-clockwise fashion: n × e > 0. Here f_1 and f_2 correspond to the magnitude of the "load" in the normal and tangential directions of the edge, they can be polynomial functions of x and involve the parameter. For example,

```
nload = '[2, 1, 0; 3, 1, 0; 4, 1, 0]'
```

will specify f_1 = 1 (pressure) on edges 2, 3, and 4. Note we permit f_1 ≠ 0, f_2 ≠ 0 only on straight edges or circular-arc edges.

(*iii*) *Output*

For output we may consider the integral of components of the field over selected regions or over selected boundary edges. To specify an output over selected regions we set the oareaload input in rbU. In particular, oareaload consists of (Region number k, factor to be included in the integral of component u_1 over Region{k}, factor to be included in the integral of component u_2 over Region{k}) 3-tuples. As before, the user need only include those 3-tuples for which at least one of the factors is non-zero. These factors may be polynomial functions of x and involve the parameter.

To specify an output over selected edges we set the oload input in rbU. In particular, oload consists of (edge number, factor to be included

in the integral of the normal component of the solution field over the edge number, factor to be included in the integral of the tangential component of the solution field over the edge number) 3-tuples; the user needs only include those 3-tuples for which a factor — the term to be incorporated in the integral — is non-zero. All edges must be boundary edges. The factor may be a polynomial function of `x` and involve the parameter.

Finally, we may ask for the integral of the stress $C_{ijkl}(\partial u_k/\partial x_l)n_j$, over selected boundary edges. To specify this output we set the `dLIFT` input in rbU. In particular, `dLIFT` consists of (edge number, prefactor to integral of first component of stress over edge number, prefactor to integral of the second component of stress over edge number) 3-tuples; the user need only include those 3-tuples for which a prefactor — the term in front of the integral — is non-zero. All edges must also appear in the `dirichlet` rbU input list (corresponding to edges on which we impose Dirichlet boundary conditions on `u`). The prefactor may depend on the parameter but not on `x`.

(*iv*) *Axisymmetric Model*

The convention for the axisymmetric case (`probtype = 'LE5'`) is as the same as in the scalar case. There are some more requirements for this problem type: Homogeneous Neumann conditions on the `x_1`-coordinate axis are required, and `dLIFT` admits just one (axial) component; i.e., in the form `dLIFT = '[edge, 0, prefactor]'`.

We note that an output must be one of the three options above; combinations are not allowed. (Multiple outputs are possible ... but not for Dummies.)

5 rbU Files

Here is a list of rbU tutorial files provided as examples. These files also appear separately (for easy copying) in the rbU files folder.

rbU_fin.m

```
clear;

probname = 'fin'
```

```
% A thermal fin problem: illustrates inhomogeneous Dirichlet conditions,
% Robin conditions, and a flux output.
plotdemo = 1;
points = '[1/2,0; 1/2, mu2/2; 1/2, mu2; -1/2, mu2; -1/2, mu2/2; -1/2, 0]';
% mu2 is the length of the fin
edge = [1,2;2,3;3,4;4,5;5,6;6,1];
geometry{1} = [1,2,3,4,5,6];
gflag = [1];
kappa{1} = '[1 0 0; 0 1 0; 0 0 0]';
muref = [.1,4];
mu_min = [.02,2];
mu_max = [0.5,8];
mu_bar = [.1,4];
nload = '[1,mu1,0,0;2,mu1,0,0;4,mu1,0,0;5,mu1,0,0]';
% imposes kappa du/dn + c1*(u - c2) = c3; here c1 = mu1 is the Biot number
dirichlet = '[6,1]';
% unity temperature at the fin base

outputname = 'flux'
dLIFT = '[6,1]';
% requests the integrated flux (with prefactor '1') over edge 6 --- the fin base
% Note: in limit of mu1 (Biot) tends to zero, output should be
% sqrt(2*mu1)*tanh( mu2* sqrt(2*mu1) );
% for all mu1, this analytical expression should be an upper bound
% for the actual output.


%%%%%%%%%%%% no user input required beyond this point
if ~exist('plotdemo')
    plotdemo = 0;
end

save rbU ;
addpath('../rbMIT_Aux')
copyfile('../rbMIT_Aux/Step1_coer_noncompliant.m',...
    strcat(probname,'_','Step1_coer_noncompliant.m'))
eval(strcat(probname,'_','Step1_coer_noncompliant'))
```

rbU_TBiso.m

```
clear;

probname = 'TBiso'
```

```
%This problem describes a Thermal Block made up by 3x3 cells with  9
%(mu1-mu9) variable isotropic conductivity coefficients (geometry is fixed)

points = '[0,0;0.5,0;1.0,0;1.5,0;0,0.5;0.5,0.5;1.0,0.5;1.5,0.5;0,1.0;...
0.5,1.0;1.0,1.0;1.5,1.0;0,1.5;0.5,1.5;1.0,1.5;1.5,1.5]';
edge = [1,2;2,3;3,4;5,6;6,7;7,8;9,10;10,11;11,12;13,14;14,15;15,16;1,5;2,6;...
        3,7;4,8;5,9;6,10;7,11;8,12;9,13;10,14;11,15;12,16];
geometry{1} = [1,14,4,13];
geometry{2} = [2,15,5,14];
geometry{3} = [3,16,6,15];
geometry{4} = [4,18,7,17];
geometry{5} = [5,19,8,18];
geometry{6} = [6,20,9,19];
geometry{7} = [7,22,10,21];
geometry{8} = [8,23,11,22];
geometry{9} = [9,24,12,23];
gflag = [1,1,1,1,1,1,1,1,1];

% Laplacian (steady heat conduction)
kappa{1} = '[mu1, 0,0; 0, mu1, 0; 0, 0, 0]';
kappa{2} = '[mu2, 0,0; 0, mu2, 0; 0, 0, 0]';
kappa{3} = '[mu3, 0,0; 0, mu3, 0; 0, 0, 0]';
kappa{4} = '[mu4, 0,0; 0, mu4, 0; 0, 0, 0]';
kappa{5} = '[mu5, 0,0; 0, mu5, 0; 0, 0, 0]';
kappa{6} = '[mu6, 0,0; 0, mu6, 0; 0, 0, 0]';
kappa{7} = '[mu7, 0,0; 0, mu7, 0; 0, 0, 0]';
kappa{8} = '[mu8, 0,0; 0, mu8, 0; 0, 0, 0]';
kappa{9} = '[mu9, 0,0; 0, mu9, 0; 0, 0, 0]';
%top side has Dirichet BC
dirichlet = '[10,0;11,0;12,0]';
%botton side has Neumann non-homogeneous BC, other sides are Neumann
%homogeneous
nload = '[1, 0, 0, 1; 2, 0, 0, 1;3,0,0,1]';
muref = [0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1];
mu_min = [0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1];
mu_max = [10,10,10,10,10,10,10,10,10];
mu_bar = [0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1];

% block below for each output (1 output only for dummies)
```

```
% The output is the averag etemperature on the bottom side.

outputname = 'average'
oload = '[1,1;2,1;3,1]';


%%%%%%%%%%% no user input required beyond this point
if ~exist('plotdemo')
    plotdemo = 0;
end

save rbU ;
addpath('../rbMIT_Aux')
copyfile('../rbMIT_Aux/Step1_coer_noncompliant.m',...
    strcat(probname,'_','Step1_coer_noncompliant.m'))
eval(strcat(probname,'_','Step1_coer_noncompliant'))
```

rbU_TBaniso.m

```
clear;

probname = 'TBaniso'

%This problem describes a Thermal Block made up by 3x3 cells with  9
%variable anisotropic conductivity coefficients (geometry is fixed).

points = '[0,0;0.5,0;1.0,0;1.5,0;0,0.5;0.5,0.5;1.0,0.5;1.5,0.5;0,1.0;...
0.5,1.0;1.0,1.0;1.5,1.0;0,1.5;0.5,1.5;1.0,1.5;1.5,1.5]';
edge = [1,2;2,3;3,4;5,6;6,7;7,8;9,10;10,11;11,12;13,14;14,15;..
    15,16;1,5;2,6;3,7;4,8;5,9;6,10;7,11;8,12;9,13;10,14;11,15;12,16];
geometry{1} = [1,14,4,13];
geometry{2} = [2,15,5,14];
geometry{3} = [3,16,6,15];
geometry{4} = [4,18,7,17];
geometry{5} = [5,19,8,18];
geometry{6} = [6,20,9,19];
geometry{7} = [7,22,10,21];
geometry{8} = [8,23,11,22];
geometry{9} = [9,24,12,23];
gflag = [1,1,1,1,1,1,1,1,1];
% Laplacian and mixed terms....the operator is du/dx*dv/dx + du/dy*dv/dy
% -mu*du/dx*dv/dy -mudu/dy*dv/dx.....
```

```
kappa{1} = '[1, -mu1, 0; -mu1, 1, 0; 0, 0, 0]';
kappa{2} = '[1, -mu2, 0; -mu2, 1, 0; 0, 0, 0]';
kappa{3} = '[1, -mu3, 0; -mu3, 1, 0; 0, 0, 0]';
kappa{4} = '[1, -mu4, 0; -mu4, 1, 0; 0, 0, 0]';
kappa{5} = '[1, -mu5, 0; -mu5, 1, 0; 0, 0, 0]';
kappa{6} = '[1, -mu6, 0; -mu6, 1, 0; 0, 0, 0]';
kappa{7} = '[1, -mu7, 0; -mu7, 1, 0; 0, 0, 0]';
kappa{8} = '[1, -mu8, 0; -mu8, 1, 0; 0, 0, 0]';
kappa{9} = '[1, -mu9, 0; -mu9, 1, 0; 0, 0, 0]';
%Dirichlet BC at the top
dirichlet = '[10,0;11,0;12,0]';
% Neumann BC at the bottom and on the other sides
nload = '[1, 0, 0, 1; 2, 0, 0, 1;3,0,0,1]';

muref = [0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1];
mu_min = [0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1];
mu_max = [0.7,0.7,0.7,0.7,0.7,0.7,0.7,0.7,0.7];
mu_bar = [0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1];

% block below for each output (1 output only for dummies)
%output is average temperature at the bottom (compliant output)
outputname = 'average'
oload = '[1,1;2,1;3,1]';


%%%%%%%%%%%% no user input required beyond this point
if ~exist('plotdemo')
    plotdemo = 0;
end

save rbU ;
addpath('../rbMIT_Aux')
copyfile('../rbMIT_Aux/Step1_coer_noncompliant.m',...
    strcat(probname,'_','Step1_coer_noncompliant.m'))
eval(strcat(probname,'_','Step1_coer_noncompliant'))
```

rbU_tailpiano.m

```
clear;

plotdemo = 1;

probname = 'tailpiano'
% This problem illustrates a Curvy Edge domain with Curvy Triangles. It
% also illustrates anisotropic conductivity.
```

```
points = '[0,mu1;1/2,0;1,-mu1;1,-1;0,-1;0,0]';
% mu1 is the amplitude of the cosinusoidal variation of the top boundary
edge = [1,2;2,3;3,4;4,5;5,6;6,1];
geometry{1} = [1,2,3,4,5,6];
gflag = [1];
kappa{1} = '[1 0 0; 0, mu2, 0; 0 0 0]';
% mu2 is the conductivity in the x_2 direction (relative to the
% conductivity in the x_1 direction)
muref = [1/3,5];
mu_min = [1/6,1];
mu_max = [1/2,25];
mu_bar = [1/3,5];
nload = '[5,0,0,1;6,0,0,1]';
% flux conditions on the left boundary
dirichlet = '[3,0]';
% zero Dirichlet conditions on the right boundary
curvedat = '[0,0,1,mu1,0,t,cos(pi*t);1,0,1,mu1,0,t-1,cos(pi*t)]';
% There are two families of curvy edges: the first is [0,0]' + [1 0; 0
% mu1]*[t, cos(pi*t)]'; the second is [1,0]' + [1 0; 0 mu1]*[t-1,
% cos(pi*t)]'.
tarclist = '[1,1,0,1/2;2,2,1/2,1]';
% Edge 1 is in family 1 with t varying between 0 and 1/2; Edge 2 is in
% family 2 with t varying between 1/2 and 1.

outputname = 'int'

oareaload = '[1,1]';
% The output is the integral of the field over the domain.
% Note in the limit that mu2 tends to infinity, the field approaches
% u(x) = \int_x1 \frac{1 + mu1}{1 + mu1 cos(\pi z)} dz and the output is then
% given by \int_01 u(x) dx.

%%%%%%%%%%%% no user input required beyond this point
if ~exist('plotdemo')
    plotdemo = 0;
end

save rbU ;
addpath('../rbMIT_Aux')
copyfile('../rbMIT_Aux/Step1_coer_noncompliant.m',...
    strcat(probname,'_','Step1_coer_noncompliant.m'))
eval(strcat(probname,'_','Step1_coer_noncompliant'))
```

rbU_plateaddedmass.m

```
clear;

probname = 'plateaddedmass'
% Added mass of an infinitely thin plate: illustrates in the scalar context
% the proper way to input a crack (with inhomogeneous boundary conditions);
% also represents case in which user should comment out standard mesh
% refines and instead use adaptmesh (go to primal_pdesolution.m, near the
% top).

points = '[0,.0001; 1,0; mu1,0; mu1,mu1; 0,mu1; 0,-.0001; mu1,-mu1; 0,-mu1]';
% Note we exploit symmetry in x_2 and consider only half the domain;
% mu1 is the half--size of the "basin" in which the plate (of half--length
% unity) oscillates.
% To be able to input boundary conditions on the crack, we slightly
% separate the (in fact coincident) two sides of the plate in order to
% create two edges: at x_1 = 0, the two edges are separated by .0001; at
% x_1 = 1 the two edges come together.
plotdemo = 1;
edge = [1,2;2,3;3,4;4,5;5,1;6,2;3,7;7,8;8,6];
% edges 1 and 6 correspond to the two sides of the plate
geometry{1} = [1,2,3,4,5]; % the top part of the basin
geometry{2} = [6,2,7,8,9]; % the bottom part of the basin
gflag = [1,1];
kappa{1} = '[1 0 0; 0 1 0; 0 0 0]';
kappa{2} = '[1 0 0; 0 1 0; 0 0 0]';
muref = [3];
mu_min = [2];
mu_max = [5];
mu_bar = [3];
nload = '[1,0,0,1;6,0,0,-1]';
% corresponds to dpressure/dn = + and - unity (which corresponds to a
% normal velocity of +1)
dirichlet = '[4,0]';
% a free surface condition (pressure = 0)

outputname = 'pforce'

oload = '[1,1;6,-1]';
% corresponds to integral of pressure over plate and hence force exerted by
% fluid on plate

% in limit of mu1 tends to infinity, output should be pi/2 (analytical
% result for added mass)
```

```
%%%%%%%%%%%% no user input required beyond this point
if ~exist('plotdemo')
    plotdemo = 0;
end

save rbU ;
addpath('../rbMIT_Aux')
copyfile('../rbMIT_Aux/Step1_coer_noncompliant.m',...
    strcat(probname,'_','Step1_coer_noncompliant.m'))
eval(strcat(probname,'_','Step1_coer_noncompliant'))
```

rbU_advdiff.m

```
clear;
%
probname = 'advdiff'
probtype = 'TH';
plotdemo = 0;
% Advection-diffusion problem with two parameters (lentgh of the channel
% and Peclet number), advection field is polynomial (=y in x direction and
% zero in y direction). Mixed Dirichlet and Neumann BC. It simulates a
% simple scalar flow.
%mu1 L
%mu2 Peclet
% a rectangular channel
points = '[0,0;mu1,0;mu1,0.5;0,0.5]';
edge = [1,2;2,3;3,4;4,1];
geometry{1} = [1,2,3,4];
gflag = [1];
% operator is 1/mu2*du/dx*du/dx+ 1/mu2*du/dy*du/dy + Udu/dxv + Vdu/dyv
kappa{1} = '[1./mu2, 0, 0; 0, 1./mu2, 0; y, 0, 0]';
muref = [1,1];
mu_min = [1,0.1];
mu_max = [1,100];
mu_bar = [1,1];
%zero Dirichlet BC on side 3 and 4
dirichlet = '[3,0;4,0]';
% Neumann on other side (non homogeneous on side 1 and zero on side 2)
nload = '[1, 0, 0,1./mu2; 2, 0, 0, 0]';

%output is  compliant and it is the "concentration" at the bottom
%side where we have Neumann BC.
```

```
outputname = 'layer'
%oareaload = '[1,1]';
oload = '[1,1]';


%%%%%%%%%%%% no user input required beyond this point
if ~exist('plotdemo')
    plotdemo = 0;
end

save rbU ;
addpath('../rbMIT_Aux')
copyfile('../rbMIT_Aux/Step1_coer_noncompliant.m',...
    strcat(probname,'_','Step1_coer_noncompliant.m'))
eval(strcat(probname,'_','Step1_coer_noncompliant'))
```

rbU_graetz.m

```
clear;

probname = 'graetz'
probtype = 'TH';
plotdemo = 0;
% Advection-diffusion problem with two parameters (lentgh of the channel
% and Peclet number), advection field is polynomial (parabolic in x direction and
% zero in y direction). Mixed Dirichlet and Neumann BC. It simulates a
% simple scalar Graezt flow (see Arpaci, "Conduction heat transfer").
%mu1 L
%mu2 Peclet
% a rectangular channel
points = '[0,0;1,0;1+mu1,0;1+mu1,1.0;1,1.0;0,1.0]';
edge = [1,2;2,3;3,4;4,5;5,6;6,1];
geometry{1} = [1,2,3,4,5,6];
gflag = [1];
% operator is 1/mu2*du/dx*du/dx+ 1/mu2*du/dy*du/dy + Udu/dxv + Vdu/dyv
kappa{1} = '[1./mu2, 0, 0; 0, 1./mu2, 0; 10*y*(1.0-y), 0, 0]';
muref = [1,1];
mu_min = [1,0.1];
mu_max = [10,100];
mu_bar = [1,1];
%zero Dirichlet BC on side 3 and 4
dirichlet = '[1,0;2,1;4,1;5,0;6,0]';
% Neumann on other side (non homogeneous on side 1 and zero on side 2)
```

```
nload = '[3, 0, 0, 0]';
%areaload='[1,1]';
%output is  compliant and it is the "concentration" at the bottom
%side where we have Neumann BC.

outputname = 'average'
oareaload = '[1,1]';
%oload = '[2,1]';


%%%%%%%%%%%% no user input required beyond this point
if ~exist('plotdemo')
    plotdemo = 0;
end

save rbU ;
addpath('../rbMIT_Aux')
copyfile('../rbMIT_Aux/Step1_coer_noncompliant.m',...
    strcat(probname,'_','Step1_coer_noncompliant.m'))
eval(strcat(probname,'_','Step1_coer_noncompliant'))
```

rbU_stagnation.m

```
clear;

plotdemo = 0;
femOPT.refine = 'subdivision';
femOPT.maxsize = 500;
probname = 'stagnation'

points = '[0,0;1,0;1,1;0,1]';
edge = [1,2;2,3;3,4;4,1];
geometry{1} = [1,2,3,4];
gflag = [1];
kappa{1} = '[mu1, 0, 0; 0, mu1, 0; x, -y, 0]';
muref = [1];
mu_min = [.1];
mu_max = [10.];
mu_bar = [1];
dirichlet = '[1,0;3,1]';


outputname = 'intsol'

oareaload = '[1,1]';
```

```
% The output is the integral of the field over the domain.



%%%%%%%%%%% no user input required beyond this point
if ~exist('plotdemo')
    plotdemo = 0;
end

save rbU ;
addpath('../rbMIT_Aux')
copyfile('../rbMIT_Aux/Step1_coer_noncompliant.m',...
    strcat(probname,'_','Step1_coer_noncompliant.m'))
eval(strcat(probname,'_','Step1_coer_noncompliant'))
```

rbU_steadysphere.m

```
clear;

plotdemo = 1;

%
probname = 'steadysphere'

points = '[1,0;3,0;3,3;0,3;0,1;mu2,0;mu2,mu2;0,mu2]';
edge = [1,2;2,3;3,4;4,5;1,5;2,6;6,7;7,8;8,4];
geometry{1} = [1,6,7,8,9,4,5];
geometry{2} = [1,2,3,4,5];
gflag = [1,1];
kappa{1} = '[y 0 0; 0, y, 0; 0 0 y*mu1]';
kappa{2} = '[y 0 0; 0, y, 0; 0 0 y*mu1]';
muref = [1,6];
mu_min = [0,4];
mu_max = [4,8];
mu_bar = [1,6];
dirichlet = '[5,1;7,0;8,0]';
curvedat = '[0,0,1,1,0,cos(pi*t),sin(pi*t)]';
tarclist = '[5,1,0,1/2]';


outputname = 'flux'

dLIFT = '[5,1]';
```

```
%analytical answer for flux (without the axisymmetric 2*pi): 1 + sqrt(mu1)
%analytical expression for field: exp(-sqrt(mu1)*(r-1))/r (where r is radius)
%note the above only true in limit mu2 tends to infinity, however for mu1
%larger even small mu2 are effectively infinite

%%%%%%%%%%% no user input required beyond this point
if ~exist('plotdemo')
    plotdemo = 0;
end

save rbU ;
addpath('../rbMIT_Aux')
copyfile('../rbMIT_Aux/Step1_coer_noncompliant.m',...
    strcat(probname,'_','Step1_coer_noncompliant.m'))
eval(strcat(probname,'_','Step1_coer_noncompliant'))
```

rbU_LEiso.m

```
clear all;

probname = 'LEiso'
%This problem describes an Elastic Block made up by 3x3 cells with
%variable isotropic Young's modulus in each subdomain (geometry is fixed).

probtype = 'LE1';
points = '[0,0;0.5,0;1.0,0;1.5,0;0,0.5;0.5,0.5;1.0,0.5;1.5,0.5;0,1.0;...
0.5,1.0;1.0,1.0;1.5,1.0;0,1.5;0.5,1.5;1.0,1.5;1.5,1.5]';
edge = [1,2;2,3;3,4;5,6;6,7;7,8;9,10;10,11;11,12;13,14;14,15;15,16;1,5;...
        2,6;3,7;4,8;5,9;6,10;7,11;8,12;9,13;10,14;11,15;12,16];
geometry{1} = [1,14,4,13];
geometry{2} = [2,15,5,14];
geometry{3} = [3,16,6,15];
geometry{4} = [4,18,7,17];
geometry{5} = [5,19,8,18];
geometry{6} = [6,20,9,19];
geometry{7} = [7,22,10,21];
geometry{8} = [8,23,11,22];
geometry{9} = [9,24,12,23];
gflag = [1,1,1,1,1,1,1,1,1];
%parameter is Young modulus, Poisson Coefficient is fixed
matprop{1} = '[mu1, 3/10]';
matprop{2} = '[mu2, 3/10]';
matprop{3} = '[mu3, 3/10]';
matprop{4} = '[mu4, 3/10]';
```

```
matprop{5} = '[mu5, 3/10]';
matprop{6} = '[mu6, 3/10]';
matprop{7} = '[mu7, 3/10]';
matprop{8} = '[mu8, 3/10]';
matprop{9} = '[mu9, 3/10]';
% on the upper side we have zero Dirichlet condition
dirichlet = '[10,x,0;11,x,0;12,x,0;10,y,0;11,y,0;12,y,0]';
% on other sides we have Neumann conditions
nload = '[1,-1,0;2,-1,0;3,-1,0]';
muref = [0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1];
mu_min = [0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1,0.1];
mu_max = [10,10,10,10,10,10,10,10,10];
mu_bar = [1,1,1,1,1,1,1,1,1];

% block below for each output (1 output only for dummies)

%output is the displacement on the loaded side (compliant)
outputname = 'average'
oload = '[1,-1,0;2,-1,0;3,-1,0]';


%%%%%%%%%%%% no user input required beyond this point
if ~exist('plotdemo')
    plotdemo = 0;
end

save rbU ;
addpath('../rbMIT_Aux')
copyfile('../rbMIT_Aux/Step1_coer_noncompliant.m',...
    strcat(probname,'_','Step1_coer_noncompliant.m'))
eval(strcat(probname,'_','Step1_coer_noncompliant'))
```

rbU_LEaniso.m

```
clear all;

probname = 'LEaniso'
%This problem describes an Elastic Block made up by 3x3 cells with
%variable orthotropic Young modulus in each subdomain (geometry is fixed).
probtype = 'LE4';
points = '[0,0;0.5,0;1.0,0;1.5,0;0,0.5;0.5,0.5;1.0,0.5;1.5,0.5;0,1.0;...
0.5,1.0;1.0,1.0;1.5,1.0;0,1.5;0.5,1.5;1.0,1.5;1.5,1.5]';
edge = [1,2;2,3;3,4;5,6;6,7;7,8;9,10;10,11;11,12;13,14;14,15;15,16;1,5;2,6;...
        3,7;4,8;5,9;6,10;7,11;8,12;9,13;10,14;11,15;12,16];
```

```
geometry{1} = [1,14,4,13];
geometry{2} = [2,15,5,14];
geometry{3} = [3,16,6,15];
geometry{4} = [4,18,7,17];
geometry{5} = [5,19,8,18];
geometry{6} = [6,20,9,19];
geometry{7} = [7,22,10,21];
geometry{8} = [8,23,11,22];
geometry{9} = [9,24,12,23];
gflag = [1,1,1,1,1,1,1,1,1];
% Ex,Ey,\nu,G
matprop{1} = '[mu1,mu2,3/10,1]';
matprop{2} = '[mu2,mu1,3/10,1]';
matprop{3} = '[mu3,mu4,3/10,1]';
matprop{4} = '[mu4,mu3,3/10,1]';
matprop{5} = '[mu5,mu6,3/10,1]';
matprop{6} = '[mu6,mu5,3/10,1]';
matprop{7} = '[mu1,mu2,3/10,1]';
matprop{8} = '[mu2,mu1,3/10,1]';
matprop{9} = '[mu1,mu2,3/10,1]';
%the upper side is clamped (zero Dirichlet BC)
dirichlet = '[10,x,0;11,x,0;12,x,0;10,y,0;11,y,0;12,y,0]';
% other sides are under Neumann condition (loaded or free-stress)
nload = '[1,-1,0;2,-1,0;3,-1,0]';
muref = [1,1,1,1,1,1,1,1,1];
mu_min = [1,1,1,1,1,1,1,1,1];
mu_max = [4,4,4,4,4,4,4,4,4];
mu_bar = [1,1,1,1,1,1,1,1,1];

% block below for each output (1 output only for dummies)
%average displacement on the loaded side (compliant)
outputname = 'average'
oload = '[1,-1,0;2,-1,0;3,-1,0]';


%%%%%%%%%%%% no user input required beyond this point
if ~exist('plotdemo')
    plotdemo = 0;
end

save rbU ;
addpath('../rbMIT_Aux')
copyfile('../rbMIT_Aux/Step1_coer_noncompliant.m',...
    strcat(probname,'_','Step1_coer_noncompliant.m'))
eval(strcat(probname,'_','Step1_coer_noncompliant'))
```

rbU_elas4.m

```
clear;

probname = 'elas4'
% This problem corresponds to a plate contains an ellipse hole under tension at two
% opposite side, assuming isotropic plane stress. The model modelled as a quarter
% of the original domain and symmetric boundary conditions are considered.
% mu1: major semiaxis
% mu2: minor semiaxis

probtype = 'LE2';
points = '[mu1,0;1,0;1,1;0,1;0,mu2]';
edge = [1,5;5,4;4,3;3,2;2,1];
geometry{1} = [1,2,3,4,5];
gflag = [1];
% material properties, Young modulus and Poisson ratio
matprop{1} = '[1, 3/10]';
muref = [0.3,0.3];
mu_min = [0.05,0.05];
mu_max = [0.55,0.55];
mu_bar = [0.3,0.3];
% nload: edge fn ft
% ft is tangential force (aligned with the en, direction is from p1 to p2)
% fn is normal force (cross(en,et)>0))
nload = '[4,1,0]';
% dirichlet: edge lab val
% lab can be "x" and "y"
dirichlet = '[2,x,0;5,y,0]';
curvedat = '[0,0,mu1,mu2,0,cos(t),sin(t)]';
tarclist = '[1,1,0,pi/2]';

% Output: The output is the integral of the horizontal displacement along the edge
% of which tension is applied. This output will approach the analytical solution
% for the infinite case when the size of the elliptic hole decreases.
% Analytical solution for the infinite plate with a circular hole is
% available in Roark's Formulas for Stress and Strain.
outputname = 'elas4'
oload = '[4,1,0]';

%%%%%%%%%%% no user input required beyond this point
if ~exist('plotdemo')
    plotdemo = 0;
end
```

```
save rbU ;
addpath('../rbMIT_Aux')
copyfile('../rbMIT_Aux/Step1_coer_noncompliant.m',...
    strcat(probname,'_','Step1_coer_noncompliant.m'))
eval(strcat(probname,'_','Step1_coer_noncompliant'))
```

rbU_crack1.m

```
clear all;

probname = 'crack1'
% plotdemo = 1;
% Problem: This problem corresponds to a plate contains a central crack under tension,
% assuming isotropic plane stress.
% The model is modelled as a quarter of the original domain, symmetric boundary
% conditions are considered.
% mu1: half-length of the crack
% mu2: half-length of the plate
probtype = 'LE2';
points = '[0,0;mu1,0;1,0;1,mu2;0,mu2]';
edge = [1,2;2,3;3,4;4,5;5,1];
geometry{1} = [1,2,3,4,5];
gflag = [1];
% material properties, Young modulus and Poisson ratio
matprop{1} = '[1, 3/10]';
muref = [0.5,1.25];
mu_min = [0.2,0.5];
mu_max = [0.8,2.0];
mu_bar = [0.5,1.25];
% nload: edge fn ft
% ft is tangential force (aligned with the en, direction is from p1 to p2)
% fn is normal force (cross(en,et)>0))
nload = '[4,1,0]';
% dirichlet: edge lab val
% lab can be "x" and "y"
dirichlet = '[2,y,0;5,x,0]';

% Output: The output is the compliant output of the system. This compliant output can
% then be used to calculate the energy release rate and stress intensity factor
% of the model. The calculated stress intensity factor will
% approach the analytical stress intensity factor when the length of the plate increases.
% Analytical solution is available in Murakami's Stress Intensity Factor Handbook
outputname = 'compliance'
```

```
oload = '[4,1,0]';

%%%%%%%%%%%% no user input required beyond this point
if ~exist('plotdemo')
    plotdemo = 0;
end

save rbU ;
addpath('../rbMIT_Aux')
copyfile('../rbMIT_Aux/Step1_coer_noncompliant.m',...
    strcat(probname,'_','Step1_coer_noncompliant.m'))
eval(strcat(probname,'_','Step1_coer_noncompliant'))
```

rbU_crack2.m

```
clear all;

probname = 'crack2'

% Problem: This problem corresponds to a plate contains a circular hole under tension,
% assuming isotropic plane stress.
% There are two cracks developed from two sides of the hole. The model is modelled
% as a quarter of the original domain,  symmetric boundary conditions are considered.
% mu1: half-length of the crack
% mu2: radius of the circular hole
probtype = 'LE2';
points = '[mu2,0;mu2+mu1,0;1,0;1,2;0,2;0,mu2]';
edge = [1,2;2,3;3,4;4,5;5,6;1,6];
geometry{1} = [1,2,3,4,5,6];
gflag = [1];
% material properties, Young modulus and Poisson ratio
matprop{1} = '[1, 3/10]';
muref = [0.3,0.2];
mu_min = [0.1,0.1];
mu_max = [0.5,0.3];
mu_bar = [0.3,0.2];
% nload: edge fn ft
% ft is tangential force (aligned with the en, direction is from p1 to p2)
% fn is normal force (cross(en,et)>0))
nload = '[4,1,0]';
% dirichlet: edge lab val
% lab can be "x" and "y"
dirichlet = '[2,y,0;5,x,0]';
curvedat = '[0,0,mu2,mu2,0,cos(t),sin(t)]';
```

```
tarclist = '[6,1,0,pi/2]';

% Output: The output is the compliant output of the system. This compliant output can then
% be used to calculate  the energy release rate and stress intensity factor of the model.
% Reference solutions are available for comparison in Murakami's Stress Intensity Factor Hand
outputname = 'compliance'
oload = '[4,1,0]';


%%%%%%%%%%% no user input required beyond this point
if ~exist('plotdemo')
    plotdemo = 0;
end

save rbU ;
addpath('../rbMIT_Aux')
copyfile('../rbMIT_Aux/Step1_coer_noncompliant.m',...
    strcat(probname,'_','Step1_coer_noncompliant.m'))
eval(strcat(probname,'_','Step1_coer_noncompliant'))
```

rbU_crack3.m

```
clear all;

probname = 'crack3'
% Problem: This problem corresponds to a plate contains two opening edge cracks
% under tension,  assuming isotropic plane stress. The model is modelled as a
% quarter of the original domain, symmetric boundary conditions are considered.
% mu1: half-length of the crack
% mu2: half-length of the opening crack
probtype = 'LE2';
points = '[0,mu2;mu1,0;1,0;1,2;0,2]';
edge = [1,2;2,3;3,4;4,5;5,1;2,4];
geometry{1} = [1,6,4,5];
geometry{2} = [2,3,6];
gflag = [1,1];
% material properties, Young modulus and Poisson ratio
matprop{1} = '[1, 3/10]';
matprop{2} = '[1, 3/10]';
muref = [0.5,0.6];
mu_min = [0.2,0.0];
mu_max = [0.8,1.2];
mu_bar = [0.5,0.6];
% nload: edge fn ft
```

```
% ft is tangential force (aligned with the en, direction is from p1 to p2)
% fn is normal force (cross(en,et)>0))
nload = '[4,1,0]';
% dirichlet: edge lab val
% lab can be "x" and "y"
dirichlet = '[2,y,0;3,x,0]';

% Output: The output is the compliant output of the system. This compliant output can then
% be used to calculate the energy release  rate and stress intensity factor of the model.
% For the case there are no opening, analytical stress intensity factors are also
% available for comparison in Murakami's Stress Intensity Factos's Handbook
outputname = 'compliance'
oload = '[4,1,0]';


%%%%%%%%%%% no user input required beyond this point
if ~exist('plotdemo')
    plotdemo = 0;
end

save rbU ;
addpath('../rbMIT_Aux')
copyfile('../rbMIT_Aux/Step1_coer_noncompliant.m',...
strcat(probname,'_','Step1_coer_noncompliant.m'))
eval(strcat(probname,'_','Step1_coer_noncompliant'))
```

rbU_elashole.m

```
clear;

probname = 'elashole'
probtype = 'LE1';
plotdemo = 0;
% LE problem  in a rectangular  region with a hole made by a parametrized
% ellipse: mu1 and mu2 are the semi-axis
% the region has also a parametrized Young Modulus mu3
points = '[mu1,0;0,mu2;-mu1,0;0,-mu2;2,2;-2,2;-2,-2;2,-2]';
edge = [1,2;2,3;3,4;4,1;5,6;6,7;7,8;8,5];
geometry{1} = [1,2,3,4];
geometry{2} = [5,6,7,8];
matprop{1} = '[mu3, 3/10]';
matprop{2} = '[mu3, 3/10]';
gflag = [0,1];
muref = [1,1,1];
```

```
mu_min = [0.8,.8,1];
mu_max = [1.2,1.2,5];
mu_bar = [1.0,1.0,1];
curvedat = '[0,0,mu1,mu2,0,cos(t),sin(t)]';
tarclist = '[1,1,0,pi/2;2,1,pi/2,pi;3,1,pi,3*pi/2;4,1,3*pi/2,2*pi]';

% nload: edge fn ft
% ft is tangential force (aligned with the en, direction is from p1 to p2)
% fn is normal force (cross(en,et)>0))

nload = '[8,1,0]';
% dirichlet: edge lab val
% lab can be "x" and "y"
dirichlet = '[6,x,0;6,y,0]';

%output is a displacement on side #3
outputname = 'elas3'
oload = '[3,0,0,0,1]';


%%%%%%%%%%%% no user input required beyond this point
if ~exist('plotdemo')
    plotdemo = 0;
end

save rbU ;
addpath('../rbMIT_Aux')
copyfile('../rbMIT_Aux/Step1_coer_noncompliant.m',...
    strcat(probname,'_','Step1_coer_noncompliant.m'))
eval(strcat(probname,'_','Step1_coer_noncompliant'))
```

rbU_elascoat.m

```
clear;

probname = 'elashole'
% LE problem  in two regions (a rectangular area containing a parametrized
% ellipse: mu1 and mu2 are the semi-axis)
% the external region has also a parametrized Young Modulus mu3
probtype = 'LE1';
points = '[mu1,0;0,mu2;-mu1,0;0,-mu2;2,2;-2,2;-2,-2;2,-2]';
edge = [1,2;2,3;3,4;4,1;5,6;6,7;7,8;8,5];
geometry{1} = [1,2,3,4];
geometry{2} = [5,6,7,8];
```

```
matprop{1} = '[mu3, 3/10]';
matprop{2} = '[1, 3/10]';
gflag = [1,1];
muref = [1,1,1];
mu_min = [0.8,.8,1];
mu_max = [1.2,1.2,5];
mu_bar = [1.0,1.0,1];
curvedat = '[0,0,mu1,mu2,0,cos(t),sin(t)]';
tarclist = '[1,1,0,pi/2;2,1,pi/2,pi;3,1,pi,3*pi/2;4,1,3*pi/2,2*pi]';
% nload: edge fn ft
% ft is tangential force (aligned with the en, direction is from p1 to p2)
% fn is normal force (cross(en,et)>0))
nload = '[8,1,0]';
% dirichlet: edge lab val
% lab can be "x" and "y"
dirichlet = '[6,x,0;6,y,0]';

outputname = 'elas8'
%displacement on side 3
oload = '[8,0,0,0,1]';

%%%%%%%%%%%% no user input required beyond this point
if ~exist('plotdemo')
    plotdemo = 0;
end

save rbU ;
addpath('../rbMIT_Aux')
copyfile('../rbMIT_Aux/Step1_coer_noncompliant.m',...
    strcat(probname,'_','Step1_coer_noncompliant.m'))
eval(strcat(probname,'_','Step1_coer_noncompliant'))
```

rbU_cantilever1.m

```
clear all;

probname = 'cantilever1'
% Problem: This problem corresponds to a cantilever beam clamped at both side
% under a uniform load on the upper side.
% mu1: length of the beam
probtype = 'LE2';
points = '[0,-1/2;mu1/4,-1/2;mu1/2,-1/2;3*mu1/4,-1/2;mu1,-1/2;...
         mu1,1/2;3*mu1/4,1/2;mu1/2,1/2;mu1/4,1/2;0,1/2]';
edge = [1,2;2,3;3,4;4,5;5,6;6,7;7,8;8,9;9,10;10,1];
```

```
geometry{1} = [1,2,3,4,5,6,7,8,9,10];
gflag = [1];
% material properties, Young modulus and Poisson ratio
matprop{1} = '[1, 3/10]';
muref = [10];
mu_min = [2];
mu_max = [18];
mu_bar = [10];
% nload: edge fn ft
% ft is tangential force (aligned with the en, direction is from p1 to p2)
% fn is normal force (cross(en,et)>0))
nload = '[6,-1,0;7,-1,0;8,-1,0;9,-1,0]';
% dirichlet: edge lab val
% lab can be "x" and "y"
dirichlet = '[5,x,0;5,y,0;10,x,0;10,y,0]';

% Output: The output is the integral of the vertical displacement along the upper side.
% This output  will approach the classical solution given by the beam theory
% when mu1 increases.
outputname = 'cantilever1'
oload = '[6,-1,0;7,-1,0;8,-1,0;9,-1,0]';


%%%%%%%%%%% no user input required beyond this point
if ~exist('plotdemo')
    plotdemo = 0;
end

save rbU ;
addpath('../rbMIT_Aux')
copyfile('../rbMIT_Aux/Step1_coer_noncompliant.m',...
    strcat(probname,'_','Step1_coer_noncompliant.m'))
eval(strcat(probname,'_','Step1_coer_noncompliant'))
```

rbU_cantilever2.m

```
clear all;

probname = 'cantilever2'
% Problem: This problem corresponds to a curved cantilever beam clamped at one side
% under a uniform pressure at the free end.
% mu1: thickness of the beam
probtype = 'LE2';
points = '[1-mu1,0;0,1-mu1;0,1+mu1;1+mu1,0]';
```

```
edge = [1,2;2,3;4,3;4,1];
geometry{1} = [1,2,3,4];
gflag = [1];
% material properties, Young modulus and Poisson ratio
matprop{1} = '[1, 3/10]';
muref = [0.8];
mu_min = [0.7];
mu_max = [0.9];
mu_bar = [0.8];
% nload: edge fn ft
% ft is tangential force (aligned with the en, direction is from p1 to p2)
% fn is normal force (cross(en,et)>0))
nload = '[2,1,0]';
% dirichlet: edge lab val
% lab can be "x" and "y"
dirichlet = '[4,x,0;4,y,0]';
curvedat = '[0,0,1-mu1,1-mu1,0,cos(t),sin(t);0,0,1+mu1,1+mu1,0,cos(t),sin(t)]';
tarclist = '[1,1,0,pi/2;3,2,0,pi/2]';

%Output: The output is the integral of the horizontal displacement along the side of
% the free end.
% The output results can be compared with the thick curved beam solution in Roark's book.
outputname = 'cantilever2_y'
oload = '[2,0,1]';

%%%%%%%%%%%% no user input required beyond this point
if ~exist('plotdemo')
    plotdemo = 0;
end

save rbU ;
addpath('../rbMIT_Aux')
copyfile('../rbMIT_Aux/Step1_coer_noncompliant.m',...
    strcat(probname,'_','Step1_coer_noncompliant.m'))
eval(strcat(probname,'_','Step1_coer_noncompliant'))
```

rbU_cork.m

```
clear all;

probname = 'cork'
probtype = 'LE1';
plotdemo = 1;
points = '[0,0;mu1,0;2,0;2,1/2;mu1,1/2;0,1/2]';
```

```
edge = [1,2;2,3;3,4;4,5;5,6;6,1];
geometry{1} = [1,2,3,4,5,6];
matprop{1} = '[1, mu2]';
gflag = [1];
muref = [1,1/8];
mu_min = [1/2,0];
mu_max = [3/2,1/4];
mu_bar = [1,1/8];

nload = '[6,-1,0]';
dirichlet = '[1,y,0;2,y,0;4,x,0;4,y,0]';

outputname = 'normalforce' % to compute friction force in x_1
dLIFT = '[4,0,1]';


%%%%%%%%%%% no user input required beyond this point
if ~exist('plotdemo')
    plotdemo = 0;
end

save rbU ;
addpath('../rbMIT_Aux')
copyfile('../rbMIT_Aux/Step1_coer_noncompliant.m',...
    strcat(probname,'_','Step1_coer_noncompliant.m'))
eval(strcat(probname,'_','Step1_coer_noncompliant'))
```

rbU_squareinchannel.m

```
clear all;

probname = 'squareinchannel'
probtype = 'LE1';
plotdemo = 1;
points = '[0,-1/2+mu1;0,-1;1,-1;1,1;0,1;0,1/2+mu1;1/2,1/2+mu1;1/2,-1/2+mu1;1,0]';
edge = [1,2;2,3;3,9;4,5;5,6;6,7;7,8;8,1;9,4];
geometry{1} = [1,2,3,9,4,5,6,7,8];
matprop{1} = '[1, mu2]';
gflag = [1];
muref = [0,1/4];
mu_min = [-1/10,2/10];
mu_max = [1/10,3/10];
mu_bar = [0,1/4];
```

```
dirichlet = '[6,x,0;6,y,1;7,x,0;7,y,1;8,x,0;8,y,1;1,x,0;2,x,0;...
    2,y,0;3,x,0;3,y,0;9,x,0;9,y,0;5,x,0]';

outputname = 'drag'
dLIFT = '[6,0,1;7,0,1;8,0,1]';


%%%%%%%%%%%% no user input required beyond this point
if ~exist('plotdemo')
    plotdemo = 0;
end

save rbU ;
addpath('../rbMIT_Aux')
copyfile('../rbMIT_Aux/Step1_coer_noncompliant.m',...
    strcat(probname,'_','Step1_coer_noncompliant.m'))
eval(strcat(probname,'_','Step1_coer_noncompliant'))
```

rbU_ellipsedrag.m

```
clear all;

plotdemo = 0;
femOPT.refine = 'subdivision';
femOPT.maxsize = 500;
probtype = 'LE5'; % axis-symmetric flag


%
% Computation of the force (drag) around an ellipsoid with axis equal to 1
% and mu2 (axisymmetric). We use Stokes by penalty (Poisson ratio \nu is next to 0.5).
% Output should be multiplied by 2*pi to get the complete drag around
% ellipsoid. Reference viscosity is given by 1/2(1+\nu).

probname = 'ellipsedrag'
points = '[1,0;0,-mu2;0,-8;8,-8;8,8;0,8;0,mu2]';
edge = [2,1;2,3;3,4;4,5;5,6;6,7;1,7];
geometry{1} = [1,2,3,4,5,6,7];
gflag = [1];
% material properties, Young modulus and Poisson ratio
matprop{1} = '[1, mu1]';
muref = [0.4,1];
mu_min = [0.4,0.9];
mu_max = [0.475,1.2];
mu_bar = [0.4,1];
```

```
curvedat = '[0,0,1,mu2,0,cos(t),sin(t)]';


tarclist = '[7,1,0,pi/2;1,1,-pi/2,0]';
dirichlet = '[7,y,0;7,x,0;1,x,0;1,y,0;3,x,0;3,y,1;5,x,0;5,y,1;4,x,0;4,y,1]';
outputname = 'drag'
dLIFT = '[1,0,1;7,0,1]';


% %%%%%%%%%%%% no user input required beyond this point
if ~exist('plotdemo')
    plotdemo = 0;
end

save rbU ;
addpath('../rbMIT_Aux')
copyfile('../rbMIT_Aux/Step1_coer_noncompliant.m',...
    strcat(probname,'_','Step1_coer_noncompliant.m'))
eval(strcat(probname,'_','Step1_coer_noncompliant'))
```

rbMIT on Athena clusters

A possibility for people at MIT with a regular account on Athena (students, faculty, staff, affiliate, visiting) is to use and run extensive calculations on Athena cluster, the Unix-based campus-wide computing facility provided by MIT (see `http://web.mit.edu/olh/` for user policy and remote access) made up of 32 and 64 bits workstations. (Note that at the present people without any computing facilities and access can use rbMIT software for online calculations through our Matlab webserver facility by testing only some selected worked problems at the address `http://augustine.mit.edu`.)

To use rbMIT software on Athena Clusters, after logging in the system with your personal *Kerberos* and password, you should reach the directory location where you want the local rbMIT directory will be placed (for example `Public` or `Private`). Then you should run the command `setup rbMIT` on the Unix prompt. At this time a local directory is created. This directory contains indications about the current version of rbMIT you are installing and using on Athena (for example `rbMIT_Part_II_and_IV_25Sept07`). Inside this directory you can find the usual subdirectories such as `rbMIT_Library`, `rbMIT_Aux`, `rbUfiles`, the documentation (for elliptic and parabolic problems), some problems as example (`tailpiano`, `fin`, `tdeprod`), the license file and the M-file to add the Matlab path. To go in this directory please type `cd rbMIT_Part_II_and_IV_25Sept07`.

To launch Matlab you should type on the Unix shell the command `add matlab`, press `Enter` and then type `matlab` and press `Enter` again. With this command at the present you are using Matlab 7.4 by default. If you are using a 64 bits workstation (a Sun architecture on Athena) you have to specify also the Matlab version by typing `matlab -ver 7.3` or `matlab -ver 7.3 tty` if you do not want a separate Matlab window. This version specification is due to the fact that on 64 bits machines only Matlab 7.3 is at the present supporting the Matlab Symbolic Toolbox.

Once you are in Matlab (and in the correct version according the machine you are using) please go inside the rbMIT local directory `rbMIT_Part_II_and_IV_25Sept07` if you are still outside (using standard `cd` UNIX commands) and then launch in Matlab `addrbMITpath` so that also paths are set.

Now you can go inside a problem directory (for example `tailpiano`, by typing `cd tailpiano`) and you may want to launch an rbU file (for example `rbU_tailpiano`) and then use rbMIT software for online computing and visualization.

If you want to create new problems and/or modifying existing ones you can create new problem directories and copying inside old rbU's files to be

modified just using the standard UNIX commands such as `mkdir` and `cp` to create a new directory or to copy a file, respectively.

An important option to be used is the command `cp -r directory_to_be_copied new_location` to copy a directory (and all its contents and subdirectories) from a remote location (for example a course or instructor locker) to a local subdirectory inside the local rbMIT software directory. This option allows to move data and to avoid extensive offline calculations just copying data already stored (and available) and to directly use online facilities during classes, homework, demo...