Documentation for rbMIT Software: Time-Dependent Problems

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

February 24, 2009

I. Directory Structure and "Problem Management"

You have already downloaded the `rbMIT_PartII_SV1_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. This documentation focuses on time-dependent (parabolic) problems. You may wish to see rbMITdoc.pdf for the documentation on elliptic problems.

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 = 'tdeprod'`, the folder should be named `tdeprod`. (This `tdeprod` folder is in fact included in the `rbMIT` directory that you have downloaded.) We provide many other examples in the `rbUfiles` folder. Note that in the time-dependent case all provided examples have `probname` begin with "`tdep`".

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 IV. (For the problem named `probname = 'tdeprod'` the `rbU_tdeprod.m` file is included in the `rbUfiles` folder but also directly in the `tdeprod` folder provided.) Once the rbU file is ready, you can do the Offline execution (Section II) and Online execution (Section III).

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 II), 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 in the `rbUfiles` folder. To begin, do the `probname = 'tdeprod'` problem with `outputname = 'left'` directly from the `tdeprod` 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 `tdepheatshield` in the `rbMIT` directory, then copy the `rbU_tdepheatshield.m` file from the `rbUfiles` folder to the `tdepheatshield` folder). In each case, once you have the `probname` folder and `rbU_probname.m` file set up, proceed to Sections II and III below.

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

II. 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 IV 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 `meshOPT.maxsize` to `250` (related to the dimension of the underlying FE approximation); however, for actual "production" calculations, the user will undoubtedly wish to increase `meshOPT.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 are not obligated to understand.

III. Online Execution

III.a. Output Evaluation

The Online stage is your reward: for the problem named `probname`, you can evaluate the output named `outputname` extremely quickly for different values of the parameter `[mu1, mu2,...,muP]` (for P parameters) in the input parameter domain defined by

`[mu1_min, mu1_max]` $\times$ `[mu2_min,mu2_max]` $\times$ `[muP_min,muP_max].`

Note you should specify the parameter ranges in the rbU file: `mu_min = [mu1_min,...,muP_min]` and `mu_max = [mu1_max,...,muP_max]` (see IV.a.(iv)).

If you have just completed the Offline Execution, you can proceed with the Online Evaluation. Make sure you stay in the `probname` folder before continuing.

Now do

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

for example, for the problem named `probname = 'tdeprod'` with output named `outputname = 'left'`, do

```
>>[sN, DeltaN] = Online_RB('tdeprod',[2.5,0.3],'left')
```

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.

A full syntax of `Online_RB` is given below

```
[sN, DeltaN] = Online_RB('probname',[mu1,mu2,...,muP],'outputname',Npr,Ndu,gt,ht)
```

Here `Npr` and `Ndu` are the RB dimensions, `gt` is the control input associated with the forcing term, and `ht` is the control input associated with the output. Both the control inputs `gt` and `ht` are functions of the time parameter `t`. If you does not explicitly specify `gt` and `ht`, the unity constant will be used for both of them.

There are actually two kinds of outputs allowed in the time-dependent case. The first kind, namely point-wise (in time) output, is the integral of the field over selected regions at different discrete timesteps. The second kind, namely integral (in time) output, is the integral in time of the control input `ht` times the integral of the field over selected regions or over selected boundary edges. Note that only the second-kind output depends on `ht`; and that the first-kind output is a vector quantity, while the second-kind output is a scalar quantity. For a detailed description about the output, see Section

IV.

Below we provide a few illustrative examples how to evaluate the first-kind and second-kind outputs for different control inputs. To evaluate the first-kind output, you need to input `1` for `ht` in `Online_RB`. For example, to evaluate the first-kind output with `gt=sin(t)`, just do

```
>> gt='[sin(t)]';
>>[sN, DeltaN] = Online_RB('probname',[mu1,mu2,...,muP],'outputname',[],[],gt,1)
```

This will return two *vectors* `sN` and `DeltaN`. For more complicated control inputs such as the triangular profile `gt=t if t` $\leq$ `0.5 and gt= 1-t if t` $>$ `0.5`, just set

```
>> gt='[t.*(t<=0.5)+(1-t).*(t>0.5)]';
```

or such as the step profile `gt= t if t` $\leq$ `0.5 and gt = 0 if t` $>$ `0.5`, set

```
>> gt='[t.*(t<=0.5)]';
```

Note that the array multiplicative operator `.*` must be used instead of the matrix multiplicative operator `*`. Otherwise, you will see an error message.

To evaluate the second-kind output with `gt=t` and `ht=sin(t)`, do

```
>> gt='[t]';
>> ht='[sin(t)]';
>>[sN, DeltaN] = Online_RB('probname',[mu1,mu2,...,muP],'outputname',[],[],gt,ht)
```

This will return two *scalars* `sN` and `DeltaN`. Note again that if you do not specify `gt` and `ht`, then `gt` and `ht` are set to unity constant in time as default.

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 `meshOPT.maxsize` (default = `250`) and `meshOPT.refine` (default is `'adaptive'`, option is `'subdivision'`) in the rbU file, as described in Section IV. 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. The computational time does depend on and increase with the RB dimensions, `Npr` and `Ndu`, that you put in `Online_RB`. In general, the RB prediction for the output, `sN`, is more accurate and the *a posteriori* error bound, `DeltaN`, gets smaller as you increase `Npr` and `Ndu`.

III.b. 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. You should remain exclusively in the `probname` folder during visualization.

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.)

A full syntax of `Vis_RB` is given below

```
Vis_RB('probname',[mu1,mu2,...,muP],Npr,gt,nk,opts)
```

Here `nk` is the timestep at which you want to plot the time-dependent field and `opts` can be set to `'mv'` if you want to visualize the field in time from timestep 1 to `nk`. The parameter `nk` must be in the range `[1 nt]`, where `nt` is the number of timesteps specified by the user in the rbU file `rbUtdep_probname` (see Section IV). The default value for `nk` is `nt`.

IV. The rbU File

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

```
probname = 'tdeprod'
outputname = 'left'
plotdemo = 1
meshOPT.refine = 'subdivision'
meshOPT.maxsize = '500'
```

(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

1. Straight Edges

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

(i) Geometry

There is a set of points

```
points = '[0,0; mu1,0; mu1,mu2; 0,mu2]';
```

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; 4,1];
```

(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 `(0,0)` (the first point in the points array) and `(mu1,0)` (the second point in the points array); similarly, the 4,1 pair defines a straight line that connects `(0,mu2)` (the fourth point in the points array) and `(0,0)` (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];
```

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 4 (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 for 2-D problems

$$\frac{\partial u}{\partial t} - \frac{\partial}{\partial x_i}\left(c\{k\}_{ij}\,\frac{\partial u}{\partial x_j}\right) + U_i\{k\}\frac{\partial u}{\partial x_i} + r\{k\}u = gt(t)f\{k\} \text{ in Region}\{k\}, t \in [0,T]$$

with initial condition

$$u(t=0) = u_0\ .$$

Here `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, `f{k}` is a scalar, and `u_0` is the initial data. Furthermore, `t` is the time parameter which varies in the interval `[0,T]`, and `gt` is the control input associated with the forcing term `f` and depends on `t`. However, we note that `gt` is *not* a part of the rbU file. You need to specify `gt` only in the Online Stage; see Section III.

However, for axis-symmetric 3-D problems which are reduced to 2-D problems by using cylindrical coordinates, we solve

$$x_1\frac{\partial u}{\partial t} - \frac{\partial}{\partial x_i}\left(x_1 c\{k\}_{ij}\,\frac{\partial u}{\partial x_j}\right) + U_i\{k\}x_1\frac{\partial u}{\partial x_i} + r\{k\}x_1 u = gt(t)x_1 f\{k\} \text{ in Region}\{k\}, t \in [0,T]$$

In this case, the `probtype` input must be set to `'TH2'`. Since the `tdeprod` problem is axis-symmetric, we set `probtype = 'TH2'` in `rbU_tdeprod.m`. If `probtype` is not specified in a rbU file, the usual 2-D problem is solved.

The quantities c{k}, U{k} = (U_1,U_2), and r{k} 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. All of these quantities can be polynomial functions of x and may depend on the parameter [mu1,mu2,...,muP].

For the tdeprod 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

$$x_1 \frac{\partial u}{\partial t} - \frac{\partial}{\partial x_1}\left(x_1 \frac{\partial u}{\partial x_1}\right) - \frac{\partial}{\partial x_2}\left(x_1 \frac{\partial u}{\partial x_2}\right) = gt(t)f\{1\} \text{ in Region}\{1\}$$

corresponding to the heat conduction equation in an axis-symmetric domain (the rod in our case).

The f{k} are specified via the farealoadinput in rbU. In particular, farealoadconsists 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 farealoadinput 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.

(*iii*) Time Parameters

For time discretization, the time interval [0,T] is divided into nt subintervals of equal length dt = T/nt. Thus, the time parameters include nt and dt, and they must be specified by the user in the rbU file.

For the tdeprod example,

```
dt = 0.1;
nt = 50;
```

which implies T = 5.

(*iv*) Initial Conditions

The general form for the initial data u_0 is

$$u_0(x,\mu) = \sum_{q=1}^{Q_u} \Theta_u^q(\mu) u_0^q(x)$$

where Θ_u^q and u_0^q can be specified via `theta_u` and `initsol` inputs in rbU, respectively. In particular, `initsol` is a list of Q_u functions of `[x1,x2]`, while `theta_u` is a list of Q_u functions of `[mu1,mu2,...,muP]`. Note that these functions must be separated out by `;`, and that the number of functions in `initsol` and `theta_u` must be the same.

For the `tdeprod` example,

```
theta_u = '[1]';
initsol = '[0]';
```

which means `u_0 = 1*0 = 0`.

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). However, we can do so in `intsol` for the unsteady cases (infact, we must).

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.

(v) 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. Note that the dirichlet boundary conditions depend on `gt`. Recall also that `gt` appears in the right hand side of the governing equations.

For the `tdeprod` problem,

```
dirichlet = '[2,1]';
```

which indicates that on edge 2 (which from the edge array and points list corresponds to the right end of the rod) we impose Dirichlet conditions `u = gt(t) * 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

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

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]`. Note the control input `gt` are associated with `g_2` and `g_3`.

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 `tdeprod` problem,

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

which indicates that on edge 4 (which corresponds to the left end of the rod) we impose a Neumann condition with coefficient `g_3=1` and that on edge 3 (which corresponds to the outer boundary of the rod) we impose a Robin condition with coefficients `g_1=1, g_2=0, g_3=1`. (Note on edge 1 we impose homogeneous Neumann conditions, which are automatic; thus edge 1 does not appear in either the `dirichlet` or `nload` rbU inputs.)

Note that if you want to solve a problem in which the Dirichlet and Neumann conditions may have different control inputs other than `gt`, you can create a rbU file for each control input and combine these separate rbUs to get the results.

(*vi*) 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 `tdeprod` problem,

```
mu_min = [2,0.2];
mu_max = [4,0.6];
```

which means that $2 \leq$ `mu1` ≤ 0.5 (this is the length of the rod) and $0.2 \leq$ `mu2` ≤ 0.6 (this is the radius of the rod).

In addition, there are two more technical inputs related to the geometry transformations (`muref`) and a posteriori error bounds (`mu_bar`). 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 = [3,0.4];
mu_bar = [3,0.4];
```

which is in fact the arithmetic mean for the parameter (often a good choice for geometric parameters).

(*vii*) Outputs

As mentioned earlier, there are two kinds of outputs allowed in the time-dependent case. The first kind, namely point-wise (in time) output, is the integral of the field over selected regions

$$s = \Theta_s(\mu) \int_{\text{Region}\{k\}} u dx.$$

Note that `s` depends on `t` since `u` depends on `t`. In fact, `s` is evaluated at `nt` discrete timesteps.

The second kind, namely integral (in time) output, is the integral in time of the control input `ht` times the integral of the field over selected regions

$$s = \Theta_s(\mu) \int_0^T \left(\int_{\text{Region}\{k\}} u dx \right) ht(t) dt$$

or over selected boundary edges

$$s = \Theta_s(\mu) \int_0^T \left(\int_{\text{Edge}\{k\}} u dx \right) ht(t) dt.$$

Note that the second-kind output depends on `ht` and is a scalar quantity. It is important to note that `ht` is *not* a part of the rbU file. You need to specify `ht` only in the Online Stage; see Section III.

To specify an output over selected regions we set the `oareaload` input in rbU. In particular, `oareaload` consists of (Region number `k`, `prefactor` to integral over Region number `k`}) pairs; the user need only include those pairs for which the `prefactor` — the term in front of the integral $\Theta_s(\mu)$ — is non-zero. The `prefactor` may be a polynomial function 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 `k`, `prefactor` to integral over edge number) pairs; the user need only include those pairs for which the `prefactor` — the term in front of the integral $\Theta_s(\mu)$ — is non-zero. All edges must be boundary edges. The `prefactor` 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 `k`, `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 be a polynomial function of `x` and involve the parameter.

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.)

For the `tdeprod` problem, we set

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

which indicates that the output is the integral of `u` over edge 4 (the left end of the rod) multiplied by `prefactor` 1.

(*viii*) "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 ;
addpath('../rbMIT_Aux')
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.

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

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

where τ is a parametrization of the curve. (This parametrization of course refers to "τ" 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 τ. 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, ρ_1 for family 1, ρ_2 for family 1,

ϕ for family 1, $\cos(\tau)$, $\sin(\tau)$; O1 for family 2, O2 for family 2,

ρ_1 for family 2, ρ_2 for family 2, ϕ for family 2, $\cos(\tau)$, $\sin(\tau)$]';

note that the last two entries for each family are always (for elliptical arcs) $\cos(\tau)$ and $\sin(\tau)$, respectively. Then `tarclist` is a list of (edge number, family to which edge belongs, start value of τ, end value of τ) 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 τ, the `curvedat`/`tarclist` description of an elliptical-arc edge must agree with the corresponding points entries. See rbU examples `rbU_tdepbobject`, `rbU_tdepheatshield`, `rbU_tdepreactor`, and `rbU_tdepsphere` 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 $\cos(\tau)$, $\sin(\tau)$ is now replaced by general functions $c_1(\tau)$, $c_2(\tau)$ selected by the user. See `rbU_tdepcurvy` as an example.

V. rbU Files

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

rbU_tdepadvdiff.m

```
clear;
```

```
% Consider time-dependent advection-diffusion in a rectangular channel characterized by the
% length of the channel and Peclet number. Velocity field U = y in x direction and
% V = 0 in y direction. Heat flux is added in at the lower side. The left and top sides
% of the channel are subject to zero temperature, while the right side is subject
% to homogeneous Neumman condition. It simulates a simple scalar flow.
% See rbU_advdiff.m for the steady version of this problem.

probname = 'tdepadvdiff'
probtype = 'TH';
femOPT.maxsize=500;
plotdemo = 0;

points = '[0,0;mu1,0;mu1,0.5;0,0.5]';
%mu1 is the length of channel
edge = [1,2;2,3;3,4;4,1];
geometry{1} = [1,2,3,4];
gflag = [1];
kappa{1} = '[1./mu2, 0, 0; 0, 1./mu2, 0; y, 0, 0]';
%mu2 is the Peclet number
muref = [1,1];
mu_min = [1,0.1];
mu_max = [10,10];
mu_bar = [1,1];
dirichlet = '[3,0;4,0]';
%zero Dirichlet BC on side 3 and 4
nload = '[1, 0, 0, 1; 2, 0, 0, 0]';
% Neumann on other side (non homogeneous on side 1 and zero on side 2)

outputname = 'layer'
oareaload = '[1,1]';

initsol = '[0]';
theta_u = '[1]';
% The temperature is initially set 0 degrees.

nt = 50;          % Number of timesteps
dt = 0.05;        % Timestep size
% Finally, to complete the problem formulation, we specify that the starting time is 0
% and that we want to study the heat distribution during 2.5 seconds.


%%%%%%%%%%%% 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'))
%%%%%%%%%% End of the offline execution.

% Now the user can perform the online stage to rapidly simulate the problem
% for different parameter values. See the software documentation for details.
```

rbU_tdepbobject.m

```
clear all;



% This example illustrates heat conduction through an elliptical buried object with
% high conductivity.

% We consider a square domain with an embedded elliptical object of minor axis a
% and major axis b. The conductivity of the embedded object is high compared to that of
% the square region. The top surface of the square region is subject to a heat flux.
% The bottom is kept at zero temperature. The rest boundaries are subject to homogeneous Neum


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

points = '[0,0;0,-mu2;mu1,0;0,mu2;0,2;1,2;5,2;5,-5;0,-5]';
% mu1 and mu2 represent the two axes of a buried object
edge = [1,2;2,3;3,4;4,1;4,5;5,6;6,7;7,8;8,9;9,2];
geometry{1} = [1,2,3,4];
% A buried object with elliptical shape
geometry{2} = [2,3,5,6,7,8,9,10];
% Square region
kappa{1} = '[mu3 0 0; 0 mu3 0; 0 0 0]';
% mu3 is conductivity of the buried object
kappa{2} = '[1 0 0; 0 1 0; 0 0 0]';
```

```
gflag = [1,1];
muref = [0.75,0.75,10];
mu_min = [0.5,0.5,5];
mu_max = [1.0,1.0,15];
mu_bar = [0.75,0.75,10];
curvedat = '[0,0,mu1,mu2,0,cos(t),sin(t)]';
tarclist = '[2,1,3*pi/2,2*pi;3,1,0,pi/2]';
% curvedat and tarclist describe the elliptical shape
nload = '[6,0,0,1;7,0,0,1]';
% Heat flux added in at the top surface
dirichlet='[9,0]';
% Temperature is kept zero at the bottom

outputname = 'strip'
oload = '[6,1]';
% Output is the average temperature over a strip [0,2] of the top surface

% zero initial condition
initsol='[0]';
theta_u='[1]';
% The temperature field is 0 degrees at the time we start applying heat flux.

nt = 200;         % Number of timesteps
dt = 0.05;        % Timestep size
% Finally, to complete the problem formulation, we specify that the starting time is 0
% and that we want to study the heat distribution during the first 10 seconds.


%%%%%%%%%%%% 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'))
%%%%%%%%%% End of the offline execution.

% Now the user can perform the online stage to rapidly simulate the problem
% for different parameter values. See the software documentation for details.
```

rbU_tdepchannel.m

```
clear;


% This example illustates the convection and diffusion of the contaminant
% concentration in a short channel, where the velocity flows from left to right.
% Homogeneous Dirichlet BC is applied on the inflow boundary (the left of the channel),
% while homogeneous Dirichlet BCs are applied on the remaining boundaries.
% The initial distribution of the contaminant concentration is assumed to
% be Gaussian and depends on a parameter mu2. The Peclet number is mu1.

probname = 'tdepchannel'
probtype = 'TH';
plotdemo = 0;
femOPT.maxsize=500;
femOPT.refine = 'subdivision';

points = '[0,0;2,0;2,1;0,1]';
% A rectangular channel of size 2 by 1
edge = [1,2;2,3;3,4;4,1];
geometry{1} = [1,2,3,4];
gflag = [1];
% operator is 1/mu1*du/dx*du/dx+ 1/mu1*du/dy*du/dy + Udu/dxv + Vdu/dyv
kappa{1} = '[1./mu1, 0, 0; 0, 1./mu1, 0; 1, 0, 0]';
% mu1 is the Peclet number
muref = [1,1];
mu_min = [0.1,0.5];
mu_max = [20,2];
mu_bar = [1,1];
dirichlet = '[4,0]';
% Zero Dirichlet BC on the inflow boundary

outputname = 'average'
oareaload = '[1,1]';
% Output is the average concentration over the whole domain

initsol = '[exp(-15*(x1-0.5).^2-30*(x2-0.5).^2)]';
theta_u = '[mu2]';
% The initial concentration is gaussian and dependent on mu2.

nt = 50;           % Number of timesteps
dt = 0.02;         % Timestep size
% Finally, to complete the problem formulation, we specify that the starting time is 0
```

```
% and that we want to study the heat distribution during the first second.

%%%%%%%%%%%% 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'))
%%%%%%%%%% End of the offline execution.

% Now the user can perform the online stage to rapidly simulate the problem
% for different parameter values. See the software documentation for details.
```

rbU_tdepcrack.m

```
clear all;


% This example illustrates heat conduction through a rectangular plate containing a very
% thin crack. The crack is characterized by mu1 (the length) and mu2 (the location).
% Heat flux is added in at the top surface, while the bottom is subject to zero temperature.

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

points = '[0,.0001; mu1,0; 5,0; 5,mu2; 2,mu2; 0,mu2; 0,-.0001; 5,-5; 0,-5]';
% mu1 is the length of the crack
% mu2 is the location of the crack from the top surface
edge = [1,2;2,3;3,4;4,5;5,6;6,1;7,2;3,8;8,9;9,7];
% edges 1 and 7 correspond to the two sides of the plate
geometry{1} = [1,2,3,4,5,6]; % the top part of the rectangular plate
geometry{2} = [7,2,8,9,10];  % the bottom part of the rectangular plate
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 = [1,1];
mu_min = [0.5,0.5];
mu_max = [2.0,2.0];
```

```
mu_bar = [1,1];
nload = '[4,0,0,1;5,0,0,1]';
% Heat flux is added in at the top
dirichlet = '[9,0]';
% Ambient temperature at the bottom

outputname = 'heatstrip'
oload = '[5,1]';
% Corresponding to the average temperature over a strip [0 2] of the top surface.

initsol='[0]';
theta_u='[1]';
% zero initial data

nt = 50;            % Number of timesteps
dt = 0.05;          % Timestep size
% Finally, to complete the problem formulation, we specify that the starting time is 0
% and that we want to study the heat conduction in 2.5 seconds.


%%%%%%%%%%%% 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'))
%%%%%%%%%% End of the offline execution.

% Now the user can perform the online stage to rapidly simulate the problem
% for different parameter values. See the software documentation for details.
```

rbU_tdepcurvy.m

```
clear all;



% In the following example, a heat transfer problem with differing material
% parameters is solved.
```

```
% The 2-D domain consists of a square with an embedded curvy region. The square region
% consists of a material with coefficient of heat conduction of 1 and a density of 1.
% The curvy region contains a uniform heat source of 4, and it has a coefficient of heat
% conduction of mu2 and a density of 1. Both regions have a heat capacity of 1.
% The curvy region has radius mu1.

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

points = '[0,0;mu1,0;5,0;5,5;0,5;0,mu1]';
% mu1 is the radius of the curvy region.
edge = [1,2;2,3;3,4;4,5;5,6;6,1;2,6];
geometry{1} = [2,3,4,5,7];
geometry{2} = [1,6,7];     % curvy region
gflag = [1,1];
kappa{1} = '[1 0 0; 0, 1, 0; 0 0 0]';
kappa{2} = '[mu2 0 0; 0, mu2, 0; 0 0 0]';
% mu2 is the conductivity of the curvy region
muref = [1,0.3];
mu_min = [0.5,0.1];
mu_max = [2.0,0.5];
mu_bar = [1,0.3];
dirichlet = '[3,0;4,0]';
% T=0 on the far field boundary
fareaload='[2,4]';
% Radioactive heat source f=4 is applied on the curvy region
curvedat = '[0,0,mu1,mu1,0,cos(pi/2*t),t]';
tarclist = '[7,1,0,1]';
% Curvy line

outputname = 'average'
oareaload = '[2,1]';
% Output is the average temparature of the curvy region

initsol = '[0]';
theta_u = '[1]';
% The temperature is 0 degrees at the time we start applying heat.

nt = 50;         % Number of timesteps
dt = .1;         % Timestep size
% Finally, to complete the problem formulation, we specify that the starting time is 0
% and that we want to study the heat distribution during the first five seconds.
```

```
%%%%%%%%%%%% 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'))
%%%%%%%%%% End of the offline execution.

% Now the user can perform the online stage to rapidly simulate the problem
% for different parameter values. See the software documentation for details.
```

rbU_tdepgraetz.m

```
clear;



% We study time-dependent advection-diffusion in a rectangular channel characterized by the
% length of the channel and Peclet number. Velocity field U = 10*y*(1.0-y) (parabolic profile
% in x direction and V = 0 in y direction. It simulates a simple scalar Graezt flow (see Arpa
% See rbU_graetz.m for the steady version of this problem.

probname = 'tdepgraetz'
probtype = 'TH';
plotdemo = 0;
femOPT.maxsize=600;

points = '[0,0;1,0;1+mu1,0;1+mu1,1.0;1,1.0;0,1.0]';
% mu1 is the length of the channel. Note there is a free zone (a square region) before the
% fluid enters the channel
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./mu2, 0, 0; 0, 1./mu2, 0; 10*y*(1.0-y), 0, 0]';
% mu2 is the Peclet number
muref = [1,1];
mu_min = [1,0.1];
mu_max = [10,10];
mu_bar = [1,1];
```

```
dirichlet = '[1,0;2,1;4,1;5,0;6,0]';
% Zero Dirichlet BCs on all sides of the free zone. Temperature is set to 1
% on the top and bottom of the channel. Homogeneous Neumann BC on other sides.

outputname = 'average'
oareaload = '[1,1]';
% Output is the average temperature

initsol = '[0]';
theta_u = '[1]';
% The temperature is initially set 0 degrees.

nt = 50;          % Number of timesteps
dt = 0.05;        % Timestep size
% Finally, to complete the problem formulation, we specify that the starting time is 0
% and that we want to study the heat distribution during the first 2.5 seconds.

%%%%%%%%%%%% 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'))
%%%%%%%%%% End of the offline execution.

% Now the user can perform the online stage to rapidly simulate the problem
% for different parameter values. See the software documentation for details.
```

rbU_tdepheatshield.m

```
clear all;



% This example illustrates heat conduction through a heat shield (a complex
% parameter-dependent domain).

probname = 'tdepheatshield'
plotdemo=0;
```

```
femOPT.maxsize = 500;
femOPT.refine = 'subdivision';

points = '[0,0;mu1,0;mu1,4;0,4+mu2;0,4;mu3,4;mu3,3;0,3;0,2;mu3,2;mu3,1;0,1]';
edge = [1,2;2,3;3,4;4,5;5,6;6,7;7,8;8,9;9,10;10,11;11,12;12,1;6,3];
geometry{1} = [3,4,5,13];
% The top area
geometry{2} = [1,2,13,6,7,8,9,10,11,12];
% Heat shield
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 = [2,1.5,1];
mu_min = [1.75,1.25,0.75];
mu_max = [2.25,1.75,1.25];
mu_bar = [2,1.5,1];
dirichlet = '[1,0]';
% Dirichlet at the bottom
nload = '[3, 0, 0, 1]';
% Heat flux is added in at the top surface
curvedat = '[0, 4, mu1, mu2, 0, cos(t), sin(t)]';
tarclist = '[3,1,0,pi/2]';

outputname = 'heatarea'
oareaload = '[1, 1]';
% Output is the integral of the temperature over the top area

initsol='[0]';
theta_u='[1]';
% The temperature is 0 degrees at the time we start applying heat.

nt=60;   % number of timesteps
dt=0.05; % timestep size
% Finally, to complete the problem formulation, we specify that the starting time is 0
% and that we want to study the heat distribution during the first 3 seconds.

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

```
%%%%%%%%%% End of the offline execution.

% Now the user can perform the online stage to rapidly simulate the problem
% for different parameter values. See the software documentation for details.
```

rbU_tdepmetal.m

```
clear all;

% This example studies a heated metal block with a rectangular crack or cavity.
% The left side of the block is heated to 100 degrees centigrade. At the right side
% of the metal block, heat is flowing from the block to the surrounding air at
% a constant rate. All the other block boundaries are isolated.

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

points = '[-0.5,-1.0;0,-1;0.5,-1.0;0.5,0;0.5,1.0;0,1.0;-0.5,1.0;...
         -0.5,0;-mu1/2,-mu2/2;mu1/2,-mu2/2;mu1/2,mu2/2;-mu1/2,mu2/2]';
% The rectangular cavity is parametrized by mu1 and mu2
% mu1 is the width and mu2 is the height, respectively.
edge = [1,2;2,3;3,4;4,5;5,6;6,7;7,8;8,1;9,10;10,11;11,12;12,9];
geometry{1} = [1,2,3,4,5,6,7,8]; % Metal block
geometry{2} = [9,10,11,12]; % Rectangular cavity
gflag = [1,0];
kappa{1} = '[1 0 0; 0, 1, 0; 0 0 0]';
kappa{2} = '[1 0 0; 0, 1, 0; 0 0 0]';
muref = [0.1,0.5];
mu_min = [0.05,0.2];
mu_max = [0.2,0.8];
mu_bar = [0.1,0.5];
dirichlet = '[7,10;8,10]';
% T=10 on the left side
nload = '[3,0,0,-1;4,0,0,-1]';
% At the right side of the metal block, heat is flowing from the block
% to the surrounding air at a constant rate.

outputname = 'average'
oareaload = '[1,1]';
% Output is the average temparature of the metal block
```

```
initsol = '[0]';
theta_u = '[1]';
% The temperature of the block is 0 degrees at the time we start applying heat.

nt = 50;          % Number of timesteps
dt = .1;          % Timestep size
% Finally, to complete the problem formulation, we specify that the starting time is 0
% and that we want to study the heat distribution during the first five seconds.

%%%%%%%%%%%% 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'))
%%%%%%%%%% End of the offline execution.

% Now the user can perform the online stage to rapidly simulate the problem
% for different parameter values. See the software documentation for details.
```

rbU_tdepreactor.m

```
clear all;



% "Reactor" problem illustrates Robin conditions and parameter-dependent initial data.

probname = 'tdepreactor'
plotdemo = 0;
femOPT.refine = 'subdivision';
femOPT.maxsize = 400;

points = '[0,0;1.5,0;0,1.5;0.5+0.2,0.5;0.5,0.5+0.2;0.5-0.2,0.5;0.5,0.5-0.2]';
edge = [1,2;2,3;3,1;4,5;5,6;6,7;7,4];
geometry{1} = [1,2,3];
geometry{2} = [4,5,6,7];
gflag = [1,0];
```

```
kappa{1} = '[1 0 0; 0 1 0; 0 0 0]';
kappa{2} = '[1 0 0; 0 1 0; 0 0 0]';
muref=[1,1];
mu_min=[0.1,0.5];
mu_max=[10,2];
mu_bar=[1,1];
nload = '[2,mu1,0,0;4,0,0,1; 5,0,0,1; 6,0,0,1; 7,0,0,1]';
% mu1 is the Bi number
curvedat = '[0,0,1.5,1.5,0,cos(t),sin(t);0.5,0.5,0.2,0.2,0,cos(t),sin(t)]';
tarclist = '[2,1,0,pi/2;4,2,0,pi/2;5,2,pi/2,pi;6,2,pi,3*pi/2;7,2,3*pi/2,2*pi]';

outputname = 'area';
oareaload = '[1,1]';

% Parameter-dependent initial data
initsol='[1.5-x1.^2-x2.^2]';
theta_u='[mu2]';

nt=50;   % number of timesteps
dt=0.02; % timestep size

%%%%%%%%%%%% 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'))
return;
%%%%%%%%%% End of the offline execution.

% Now the user can perform the online stage to rapidly simulate the problem
% for different parameter values. See the software documentation for details.
```

rbU_tdeprod.m

```
clear all;


% This heat distribution problem is an example of a 3-D heat problem that
% is reduced to a 2-D problem by exploiting axis-symmetrical coordinates.
```

```
% Consider a cylindrical radioactive rod with radius r and length L. At the left end,
% heat is continuously added. The right end is kept at a constant temperature.
% At the outer boundary, heat is exchanged with the surroundings by transfer.
% At the same time, heat is uniformly produced in the whole rod due to radioactive processes.
% Assume that the initial temperature is zero.

probname = 'tdeprod'
plotdemo = 0;
probtype = 'TH2'; % axis-symmetric flag
femOPT.refine = 'subdivision';
femOPT.maxsize = 500;

points = '[0,0;mu1,0;mu1,mu2;0,mu2]';
% mu1 is the length of the rod
% mu2 is the radius of the rod
edge = [1,2;2,3;3,4;4,1];
geometry{1} = [1,2,3,4];
gflag = [1];
kappa{1} = '[1 0 0; 0, 1, 0; 0 0 0]';
muref = [3,0.4];
mu_min = [2,0.2];
mu_max = [4,0.6];
mu_bar = [3,0.4];
fareaload='[1,5]';
% Radioactive heat source f=5
nload = '[4,0,0,1;3,1,0,1]';
% Heat flux at the left end and Robin condition at the outer boundary
% Homogenuous Neumann condition at the bottom due to symmetry
dirichlet = '[2,1]';
% Dirichlet T=1 on the right end

outputname = 'bottom'
oload = '[1,1]';
% Output is the average temperature at the bottom of the rod

initsol = '[0]';
theta_u = '[1]';
% The temperature of the rod is 0 degrees at the time we start applying heat.

nt = 50;         % Number of timesteps
dt = 0.1;        % Timestep size
% Finally, to complete the problem formulation, we specify that the starting time is 0
% and that we want to study the heat distribution during the first five seconds.

%%%%%%%%%%% 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'))
%%%%%%%%%% End of the offline execution.

% Now the user can perform the online stage to rapidly simulate the problem
% for different parameter values. See the software documentation for details.
```

rbU_tdepsphere.m

```
clear all;



% This example illustrates axis-symmetric case which reduces a 3D problem in a sphere to
% a 2D problem in a plane. There is a steady-state solution in some limit.

probname = 'tdepsphere'
plotdemo = 0;
probtype = 'TH2'; % axis-symmetric flag
femOPT.refine = 'subdivision';
femOPT.maxsize = 500;

points = '[1,0;3,0;3,3;0,3;0,1;mu2,0;mu2,mu2;0,mu2]';
% mu2 is the size of the truncated domain
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} = '[1 0 0; 0, 1, 0; 0 0 mu1]';
kappa{2} = '[1 0 0; 0, 1, 0; 0 0 mu1]';
muref = [1,6];
mu_min = [1/2,4];
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 = 'integ'
oareaload = '[1,1;2,1]';

%analytical expression for steady-state output is (1/mu1 + 1/sqrt(mu1))
%note the above only true in limit mu2 tends to infinity, however for mu1
%larger even small mu2 are effectively infinite

initsol = '[0]'; % Zero initial data
theta_u = '[1]';
nt = 50;         % Number of timesteps
dt = .1;         % Timestep size


%%%%%%%%%%%% 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'))
%%%%%%%%%% End of the offline execution.

% Now the user can perform the online stage to rapidly simulate the problem
% for different parameter values. See the software documentation for details.
```

rbU_tdepstagnation.m

```
clear all;


% This is a time-dependent convection-diffusion problem which has the velocity
% depending on the spatial coordinates

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

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]';
% mu1 is the conductivity
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.
% Note in the limit that mu1 tends to zero, the field approaches
% u(x_1,x_2,mu1) = erf(x_2/(2 sqrt(mu1)) as t tends to infinity, and hence
% s(mu1) tends to \int_0^1 erf(x_2/(2 sqrt(mu1)) dx_2 in the same limits.

initsol = '[x2]'; % Initial data
theta_u = '[1]';  %
nt = 50;          % Number of timesteps
dt = .1;          % Timestep size


%%%%%%%%%%%% 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'))
%%%%%%%%%% End of the offline execution.

% Now the user can perform the online stage to rapidly simulate the problem
% for different parameter values. See the software documentation for details.
```