User's Manual for the Movgrid Toolbox

The MatMOL Group (2009)

Introduction

The Movgrid toolbox consists of a set of functions included in MatMOL which allows the user to solve systems described by partial differential equations (PDEs) using the Dynamic Regridding approach. This dynamic regridding is carried out based in the equidistribution principle. The algorithm for the moving grid was originally proposed in [1, 3]. A description (including some benchmark examples) of the algorithm employed in the MatMOL toolbox is given in [2].

The Movgrid Toolbox by the Example

In this section we will show how to use the Movgrid toolbox to solve the following hypothetic system of PDEs:

$$\frac{\partial u_1}{\partial t} = -\frac{\partial f_1(u_3)}{\partial z} + \varepsilon \frac{\partial^2 u_1}{\partial z^2} + k_1 u_2 \tag{1}$$

$$\frac{\partial u_2}{\partial t} = -\frac{\partial f_2(u_2, u_3)}{\partial z} + \varepsilon \frac{\partial^2 u_2}{\partial z^2} \tag{2}$$

$$\frac{\partial u_3}{\partial t} = -\frac{\partial f_3(u_1, u_2, u_3)}{\partial z} + \varepsilon \frac{\partial^2 u_1}{\partial z^2} \tag{3}$$

where

$$f_1 = u_3; \quad f_2 = (\gamma - 1)u_3 - \frac{(\gamma - 3)u_2^2}{2u_3}; \quad f_3 = \left(\gamma u_3 - \frac{(\gamma - 1)u_2^2}{2u_1}\right)\frac{u_2}{u_1}$$

with boundary and initial conditions of the form:

$$\frac{\partial u_1(0,t)}{\partial z} = 0; \quad \frac{\partial u_1(1,t)}{\partial z} = 0; \tag{4}$$

$$u_2(0,t) = 0; \quad u_2(1,t) = 0; \tag{5}$$

$$\frac{\partial u_3(0,t)}{\partial z} = 0; \quad \frac{\partial u_3(1,t)}{\partial z} = 0; \tag{6}$$

$$u_1(z,0) = \begin{cases} 1 & \text{if} \quad 0 \le z \le 0.5 \\ 0.125 & \text{if} \quad 0.5 < z \le 1 \end{cases}; \quad u_2(z,0) = 0;$$

$$u_3(z,0) = \begin{cases} 2.5 & \text{if} \quad 0 \le z \le 0.5 \\ 0.25 & \text{if} \quad 0.5 < z \le 1 \end{cases} \tag{7}$$

The structure of the software of the Movgrid in Matlab© is as follows:

- A set of Matlab© functions that performs automatic operations for computing the different terms in the moving grid algorithm (see [2] for details). It should be mention that this functions are independent of the problem and they are included in the MatMOL toolbox.

- A main program that allows the user to specify the space and time domains, the PDE parameters, the initial conditions, the type of discretization scheme, the type of monitor function, the tuning parameters of the adaptive grid, etc. This file is problem dependent. In the sequel we will explain how to construct the different elements of this file which we will call `Main_file.m`.

- A set of MATLAB functions where a user input is needed for defining

  a) The right-hand side of the PDEs (1)-(3) is included in `right_vect.m`

  b) The boundary conditions (4)-(6) are included in `Bcintroduct.m`

  c) Possibly the flux function if the PDE is in conservation form

  d) Possibly the Jacobian of the flux function if the PDE is in conservation form

  These files are also problem dependent

Let us start the description of the implementation with the `Main_file.m`.

The structure of `Main_file.m`

First, the user should indicate the global variables that will be employed by the other files in the toolbox

```
% Global variables
global zL zR n ne X eq choice
global alpha kappa mu tau
global u z
global gam eps k1
```

The firsts three lines of global variables are common to all the problems. The last one is for the PDE parameters that the user wants to pass to the `right_vect.m` function.

No the user can define, for instance, the parameters which are common to other finite difference schemes like the temporal conditions, the parameters of the PDEs and the spatial grid

```
% Temporal conditions
t = [0 .15 .23 .28 .32];

% PDE parameters
ne  = 3;            % Number of PDE equations
eps = 10^(-3);
gam = 1.4;
k1  = 1;
```

```
% Initial spatial grid z : if n is the number of moving
%                          grid points, then dim(z) = n+2
zL   = 0;                 % Initial point of the grid
zM   = 0.5;               % A middle point of the grid
zR   = 1;                 % Final point of the grid
nLM  = 21;                % Number of points in the first interval
nMR  = 61;                % Number of points in the second interval
dzLM = (zM-zL)/(nLM-1);
dzMR = (zM-zR)/(nMR-1);
z    = [zL:dzLM:zLM zLM+dzMR:dzMR:zMR]'; % The grid
n    = length(z)-2;                     % Moving points
```

where `ne` is the number of partial differential equations. A middle point (`zM`) was included to introduce the initial conditions.

After this, the user can define the parameters for the moving grid algorithm (see [2] for details).

```
% Moving grid parameters
alpha = 0.05;
kappa = 1;
mu    = kappa*(kappa+1);
tau   = 1e-4;
```

In the next block of the m-file, the initial conditions (7) will be implemented:

```
% Initial conditions
for i=1:n+2,
    if (z(i) <= zM)
        u(i,1) = 1;
        u(i,2) = 0;
        u(i,3) = 2.5;
    else
        u(i,1) = 0.125;
        u(i,2) = 0;
        u(i,3) = 0.25;
    end
end
```

Now the user has to specify the variables that function JPat will employ to implement the sparsity pattern of the Jacobian in `X`. A tensor `eq` of dimension $n_e \times 3 \times n_e$, with n_e being the number of PDEs, is created based on the PDEs (1)-(3) and the chosen finite difference scheme.

Consider a three point centered finite difference scheme for the first derivative and a five point centered finite difference scheme for the second derivative. Let us now show how the tensor `eq` is constructed in Matlab©

```
% Global variables for JPat
X = zeros(n*(ne+1),n*(ne+1));

% Tensor eq
```

```
eq(:,:,1) = [-2 2 1 ;  0 0 1 ; -1 1 1];
eq(:,:,2) = [ 0 0 0 ; -2 2 1 ; -1 1 1];
eq(:,:,3) = [-1 1 1 ; -1 1 1 ; -2 2 1];
```

In the construction of `eq` one can note the following:

- Matrix `eq(:,:,k)` describes the pattern of points used in equation `k` in order to take into account the chosen finite difference schemes.
- The line `j` in matrix `eq(:,:,k)` refers to the presence of variable `j` (u_j) in equation `k`.
- In order to fill the elements of the matrix `eq(:,:,k)` we look at the finite difference schemes.
  - For the computation of the second derivative in the point i, the following points will be involved: $i - \mathbf{2}$, $i - 1$, i, $i + 1$ and $i + \mathbf{2}$ (five point centered scheme). The extremes points in this case are $i - \mathbf{2}$ and $i + \mathbf{2}$.
  - For the computation of the first derivative in the point i, the following points will be involved: $i - \mathbf{1}$, i and $i + \mathbf{1}$ (three point centered scheme). The extremes points in this case are $i - \mathbf{1}$ and $i + \mathbf{1}$

  The first column of matrix `eq(:,:,k)` refer to the left extreme of the finite difference scheme (in the case of the second derivative $i-2$ so -2, in the case of the first derivative $i - 1$ so -1). The second column of matrix `eq(:,:,k)` refer to the right extreme of the finite difference scheme (in the case of the second derivative $i+2$ so 2, in the case of the first derivative $i + 1$ so 1). The third column indicates either if the variable appears implicity in the equation (in which case it takes the value 1) or not (in which case it takes the value 0).
- If we look at the first equation -Eqn (1)-, we see that:
  - The first variable (u_1) is present only via the second derivative $\left(\frac{\partial^2 u_1}{\partial z^2}\right)$ term. Since for the second derivative we have chosen a five point centered scheme with extremes $i - \mathbf{2}$ and $i + \mathbf{2}$, we have:

    `eq(1,:,1) = [-2 2 1]`
  - The second variable (u_2) appears only without derivative ($k_1 u_2$) so :

    `eq(2,:,1) = [0 0 1]`
  - The third variable (u_3) appears only in the first derivative term through $\left(\frac{\partial f_1(u_3)}{\partial z}\right)$. Since for the first derivative we have chosen a three point centered scheme with extremes $i - \mathbf{1}$ and $i + \mathbf{1}$, we have:

    `eq(3,:,1) = [-1 1 1]`
- If we look at the second equation -Eqn (2)-, we see that:

- The first variable (u_1) does not appear so:

  `eq(1,:,2) = [0 0 0]`

- The second variable (u_2) appears in the second derivative term and in the first derivative term through function $f_2(u_2, u_3)$. In this case we have to choose the set that include all the points, so:

  `eq(2,:,2) = [-2 2 1]`

  Since $[-1, 1]$ is included in $[-2, 2]$.

- The third variable (u_3) appears only in the first derivative term through $f_2(u_2, u_3)$ so:

  `eq(3,:,2) = [-1 1 1]`

- Similarly we can construct `eq(:,:,3)` by looking at the third equation -Eqn (3)-.

Example 1 (Another example for constructing tensor `eq(:,:,k)`) *Consider the system:*

$$\frac{\partial u_1}{\partial t} = D\frac{\partial^2 u_1}{\partial z^2} - v\frac{\partial u_1}{\partial z} + u_2$$

$$\frac{\partial u_2}{\partial t} = -v\frac{\partial u_2}{\partial z}$$

Compute the tensor `eq(:,:,:)` *if for the first derivative a five point biased upwind finite difference scheme ($i-\mathbf{3}$, $i-2$, $i-1$, i and $i+\mathbf{1}$) is employed, while for the second derivative a five point centered finite difference ($i-\mathbf{2}$, $i-1$, i, $i+\mathbf{1}$ and $i+\mathbf{2}$) is preferred.*

If we look at the first equation, we see that:

- *The first variable (u_1) is present via the first and second derivative terms. In this case we have to choose the set that include all the points, so:*

  `eq(1,:,1) = [-3 2 1]`

  Since the set $[-3, 2]$ includes both the set $[-3, 1]$ from the five point biased upwind and the set $[-2, 2]$ from five point centered.

- *The second variable (u_2) appears only without derivative (u_2) so :*

  `eq(2,:,1) = [0 0 1]`

If we look at the second equation, we see that:

- *The first variable (u_1) does not appear so:*

  `eq(1,:,2) = [0 0 0]`

- *The second variable (u_2) is present only via the first derivative term so:*

  `eq(2,:,2) = [-3 1 1]`

In the next block of `Main_file.m` to solve system (1)-(7) the user will choose the monitor function employed by the adaptive grid. Three different possibilities are available:

- `'der1'`. Monitoring based on the first derivative of the field.

$$m(i) = \sqrt{\alpha + \frac{1}{n_e}\sum_{j=1}^{n_e} \frac{\left(u(i+1,j) - u(i,j)\right)^2}{\left(z(i+1) - z(i)\right)^2}}$$

- `'der2'`. Monitoring based on the second derivative of the field.

$$m(i) = \sqrt{\alpha + \frac{1}{n_e}\sum_{j=1}^{n_e} \frac{\frac{\partial^2 u(i+1,j)}{\partial z^2} + \frac{\partial^2 u(i,j)}{\partial z^2}}{2}}$$

- `'derf1'`. Monitoring based on the first derivative of the flux.

$$m(i) = \sqrt{\alpha + \frac{1}{n_e}\sum_{j=1}^{n_e} \frac{\left(fl[u(i+1,j)] - fl[u(i,j)]\right)^2}{\left(z(i+1) - z(i)\right)^2}}$$

This choice is easily implemented in Matlab©, for instance:

```
% Monitoring choice
choice = 'der1';
```

Usually when we are considering a conservation equation (like in this example) `'der1'` and `'derf1'` work better than `'der2'`. On the contrary when trying to solve wave equations, `'der2'` will be, usually, preferable.

It should be mentioned here that, in some problems, it may be useful to perform an adaptation of the initial grid. Some examples indicating how to do this were included in the MatMOL toolbox (see, for instance, `fischkol_main.m` or `burger_main.m` files).

In the moving grid algorithm, the state variables and the coordinates of the internal (moving) grid points are taken together to solve the problem. The new vector x constructed from the state variables and grid coordinates is of the form:

$$x = [u_1^1, u_1^2, ..., u_1^{n_e}, z_1, u_2^1, u_2^2, ..., u_2^{n_e}, z_2, ..., u_n^1, u_n^2, ..., u_n^{n_e}, z_n]^T$$

where the subindices indicate the position in the grid and the superindices denote the field considered. In this way, u_i^j refers to the value of the field u^j in the grid point z_i.

The next block in the script `Main_file.m` will assemble state variables and grid coordinates in x:

```
% Global initial condition
for jj = 1:ne,
    x(jj:ne+1:n*(ne+1)) = u(2:n+1,jj);
end
x(ne+1:ne+1:n*(ne+1))   = z(2:n+1);
```

Finally we choose the parameters of the ODE solver and call it using, for instance, the following instructions

```
% ODE Solver parameters
options = odeset('RelTol',1e-6,'AbsTol',1e-6,'Mass',@mass,...
                 'MStateDependence','strong','JPattern',JPat,...
                 'MvPattern',MvPat,'MassSingular','no');

% Call to the ODE solver
[tout, yout] = ode15s(@right_vect,t,x,options);
```

After this, the user can include a block for visualizing the results.

The structure of `right_vect.m`

After applying the moving grid approach to eqns (1)-(2) a semidiscrete system of ODEs is obtained. The general form of this system is [2]:

$$A(x,t)\frac{dx}{dt} = b(x,t)$$

where $A(x,t)$ is the mass matrix and $b(x,t)$ is the discrete version of the right hand side terms of eqns (1)-(2) with x being the global coordinates as indicated in `Main_file.m`. This is, a vector which includes the state variables and the grid coordinates.

In this file the user will define the right hand side terms ($b(x,t)$). These terms are implemented in matrix `udot(:,k)` where `k` indicates the number of the equation. In this file, the user will also choose the finite difference schemes for the computation of the spatial derivatives. It should be noted that vector x contains only the interior (moving) grid points, nevertheless, $b(x,t)$ also makes use of the boundary points z_0, z_{n+1}, u_0^j, u_{n+1}^j with $j = 1,..,n_e$. The structure of $b(x,t) \in \mathbb{R}^{(n_e+1)n\times 1}$ is:

$$b(x,t) = \left[\frac{du_1^1}{dt}, \frac{du_1^2}{dt}, ..., \frac{du_1^{n_e}}{dt}, g_1, \frac{du_2^1}{dt}, \frac{du_2^2}{dt}, ..., \frac{du_2^{n_e}}{dt}, g_2, ..., \frac{du_n^1}{dt}, \frac{du_n^2}{dt}, ..., \frac{du_n^{n_e}}{dt}, g_n\right]^T$$

where

$$g_i = \frac{1}{M_i}\left[-\frac{\mu}{\Delta z_{i+1}} + \frac{1+2\mu}{\Delta z_i} - \frac{\mu}{\Delta z_{i-1}}\right] - \frac{1}{M_{i-1}}\left[-\frac{\mu}{\Delta z_i} + \frac{1+2\mu}{\Delta z_{i-1}} - \frac{\mu}{\Delta z_{i-2}}\right]$$

For details see [1, 3].

In the `right_vect.m` function the user will first declare the global variables employed also in other functions of the toolbox:

```
% Global variables
global n ne choice
global dltz mu
global gam eps k1
```

The second block will make a call to the function which implements the boundary conditions (this function, as we will show in the next section, will also separate the state variables and the node positions).

```
% Separate state variables and node positions and implement the BCs
[u z] = Bcintroduct(t,x);
```

In the next block the user will compute the monitoring function (by calling the Matlab© function `monitor.m`) and the right vector g (by calling function `rightmon.m`)

```
% Compute the monitoring function mon(i)
mon = monitor(u,z,choice,t);

% Compute the right vector g(n) of the moving grid equation
g = rightmon(mon,dltz,mu);
```

The blocks presented so far are independent of the problem (except the last line of global variables).

After this, the user will define the nonlinear functions, choose the finite difference scheme and compute the spatial derivatives of the fields and nonlinear functions:

```
% Nonlinear functions
f1 = u(:,3);
f2 = (gam-1)*u(:,3)- ((gam-3)*u(:,2).^2)./(2*u(:,3));
f3 = (gam*u(:,3) - (gam-1)*u(:,2).^2./(2*u(:,1))).*u(:,2)./u(:,1);

% Compute the finite difference matrices
D1   = matfd(z, 'non_uni', '1st', 3, 'centered');
D2   = matfd(z, 'non_uni', '2nd', 5, 'centered');

% Field second derivatives
u1zz = D2*u(:,1);
u2zz = D2*u(:,2);
u3zz = D2*u(:,3);

% Nonlinear functions first derivatives
f1z  = D1*f1;
f2z  = D1*f2;
f3z  = D1*f3;
```

The implementation of the right hand side terms is carried out in the next block:

```
% Temporal derivatives (RHS terms)
udot(:,1) = -f1z + eps*u1zz + k1*u(:,2);
udot(:,2) = -f2z + eps*u2zz;
udot(:,3) = -f3z + eps*u3zz;
```

Finally, the last block, which is independent of the problem, is for the assembly of the global right vector $b(x,t)$

```
% Assemble the global right vector
for jj = 1:ne
    xt(jj:ne+1:n*(ne+1),1) = udot(2:n+1,jj);
end
xt(ne+1:ne+1:n*(ne+1),1)   = g;
```

The structure of `Bcintroduct.m`

In this function the user must separate the state variables from the grid positions and implement the boundary conditions. The boundary conditions need to be specified in Dirichlet form. This means that for Neumann and Robin type a transformation have to be performed.

The first part (separate state variables from grid positions) is the same for all problems and it is easily carried out with the following Matlab© lines

```
% Separate dependent variables and node positions
for jj = 1:ne
    u(2:n+1,jj) = x(jj:ne+1:n*(ne+1));
end
    z = [zL x(ne+1:ne+1:n*(ne+1))' zR];
```

For the implementation of the boundary conditions in Dirichlet form, two possibilities are available:

- To compute the value of the field in the boundary by hand. If, for instance, we have the following boundary condition at the right boundary:

$$a_1 \frac{\partial u_k(z_R)}{\partial z} + a_0 u_k(z_R) = f(t) \tag{8}$$

  we can approximate the first derivative by a simple finite difference scheme so that

$$\frac{\partial u_k(z_R)}{\partial z} = \frac{u(n+2,k) - u(n+1,k)}{z(n+2) - z(n+1)}$$

  Substituting this expression into equation (8) and after some algebraic manipulations we obtain:

$$u(n+2,k) = \frac{f(t) + a_1 \dfrac{u(n+1,k)}{z(n+2) - z(n+1)}}{a_0 + \dfrac{a_1}{z(n+2) - z(n+1)}}$$

- To use the MatMOL function `BC_dir`. This function is called as follows:

```
u = BC_dir(bc_form, u, uin, D1, pos, v, Dif)

% Input parameters:
%   bc_form : Type of the boundary condition 'dir' for Dirichlet
%             boundary type, 'neu' for Neumann type and 'rob'
%             for Robin type.
%   u       : Value of the field in the spatial domain
%   uin     : Value of the non homogeneous part of the boundary.
```

```
%             For homogeneous boundary conditions, set this
%             value to 0.
%   D1      : Finite difference matrix for the first derivative
%   pos     : Position of the boundary condition: 'begin' if the
%             boundary condition refers to the first point of
%             the spatial domain, 'end' if it refers to the last
%             point.
%   v       : Fluid velocity. This parameter is only necessary
%             in the case of Robin boundary conditions,
%             otherwise it can be set to 0
%   Dif     : Difussivity. This parameter is only necessary in
%             the case of Robin boundary conditions, otherwise
%             it can be set to 0
%
% Output parameters:
%   ubc     : Value of the field in the boundary (Dirichlet
%             form)
```

Using this function, boundary condition (8) can be computed as

```
Dz       = matfd (z, 'non_uni', '1st', 3, 'upwind',v);
u(n+2,k) = BC_dir('rob', u(:,k), f/a0, Dz, 'end', a1, a0);
```

Boundary conditions (4)-(6) are implemented in Matlab© as follows

```
% Implement the BCs
% Computation of the derivative matrix
Dz       = matfd (z, 'non_uni', '1st', 5, 'centered');
% Boundary conditions for u1
u(1,1)   = BC_dir('neu',u(:,1),0,Dz,'begin',0,0);
u(n+2,1) = BC_dir('neu',u(:,1),0,Dz,'end',0,0);
% Boundary conditions for u2
u(1,2)   = 0;
u(n+2,2) = 0;
% Boundary conditions for u3
u(1,3)   = BC_dir('neu',u(:,3),0,Dz,'begin',0,0);
u(n+2,3) = BC_dir('neu',u(:,3),0,Dz,'end',0,0);
```

Note that other finite difference schemes can be used (when choosing `D1`) to approximate the boundary conditions.

Remark: The boundary conditions must be always introduced, even in the cases where mathematically it is not necessary. Consider, for instance, an equation which is purely convective. From the mathematical point of view, only one boundary condition is required to solve it. Nevertheless, with the moving grid algorithm, both boundary conditions (at the beginning and at the end) have to be specified. It is necessary, then, to intoduce a "fictitious" boundary condition. From the point of view of time integration, the most flexible way of defining such

conditions is to make the first spatial derivative of the corresponding function to be constant. If we consider the first point of the spatial domain, this translates into:

$$\frac{\partial u(1,k)}{\partial z} = \frac{\partial u(2,k)}{\partial z}$$

References

[1] J. G. Blom and P. A. Zegeling. Algorithm 731: A moving-grid interface for systems of one-dimensional time-dependent partial differential equations. *ACM Transactions on Mathematical Software*, 20:194–214, 1994.

[2] P. Saucez, L. Some, and A. Vande Wouwer. Matlab implementation of a moving grid method based on the equidistribution principle. *Applied Mathematics and Computation*, d.o.i. 10.1016/j.amc.2009.07.34, 2009.

[3] J. G. Verwer, J. G. Blom, R. M. Furzeland, and P. A. Zegeling. A moving-grid method for one-dimensional pdes based on the method of lines. In J.E. Flaherty, P.J. Paslow, M.S. Shephard, and J.D. Vasilakis, editors, *Adaptive Methods for Partial Differential Equations*, pages 160–175. SIAM, Philadelphia, 1989.