SPINV code

User's guide

1 General information

SPINV is a 2D Matlab code which allows to solve forward and inverse problems for electrical potential equation. It gives also the possibility to couple ground water equation with electrical potential equation. Simulations can be run in transient and stationary modes.
The main folder named **SPINV** is divided into five subfolders :

- **Distmesh** : contains functions for generating geometry and meshes.
- **Finite Elements Functions** : contains all functions necessary for implementing the finite elements method.
- **Inverse-Reg** : contains inversion and regularization functions.
- **Plot tools** : contains post-processing functions which permit to visualize results.
- **Examples** : contains several forward and inverse problems solved using the code.

Separating all the functions in different folders depending on their functionality makes the code cleaner and easier to understand nevertheless in order to run a simulation, the user needs to add all folders to the Matlab root folder. This can be simply done using the predefined Matlab function **set path** as follows :
In the main terminal of Matlab, click on **File**, then go to **Set Path** then click on **Add with Subfolders**.

2 How to run a simulation ?

In order to solve any forward problem four steps are required :
– Creating Geometry and Mesh.
– Fixing boundary conditions.
– Assembling Stiffness, Mass matrices and right hand side.
– Solving resulting linear system.

2.1 Creating Geometry and Mesh

This step is performed by distmesh code. For more informations see Per-Olof Persson, 2005. Mesh Generation for Implicit Geometries.

2.2 Fixing boundary conditions

SPINV uses logical indexing for defining boundary conditions.

Suppose you want to solve your problem on unit square and apply non homogeneous Dirichlet boundary conditions on top and bottom boundaries and non homogeneous Neumann boundary conditions on remaining borders.
You can define your borders as follows :

```
gamm0=( abs(p(:,2)-ymax<epsilon );
gamma1=( abs(p(:,2)-ymin)<epsilon );
gamma2=( abs(p(:,1)-xmin)<epsilon );
gamma3=( abs(p(:,1)-xmax)<epsilon );
```

where p is a 2 column vector containing all nodes coordinates, epsilon is a tolerance value choosen by the user depending on to which extinct the mesh is refined. $xmin, xmax, ymin, ymax$ are domain limits ; in unit square case $xmin = 0$, $ymin = 0$, $xmax = 1$ and $ymax = 1$.
After defining your boundaries you create vectors containing values you want to affect to those boundaries. For example, if we want to impose Dirichlet boundary conditions with values 1.5 and 4 on top and bottom boundaries and Neumann boundaries(left and right borders) with values 2 and 5 , we do as follows :

For Dirichlet boundary conditions

```
gammaD=gamma0 | gamma1 % defining Dirichlet boundary

zD=zeros(size(p,1),1) % initializing vector of Dirichlet values
zD(gamma0)=1.5;
zD(gamma1)=4;
```

The sign | denotes the logical operator **OR**.

For Neumann boundary conditions

```
gammaN=gamma2 | gamma3 % defining Neumann boundary

zN=zerops(size(p,1),1) % initializing vector of Neumann values
zN(gamma2)=2;
zN(gamma3)=5;
```

2.3 Assembling Stiffness, Mass matrices, right hand side and Solving linear system of the problem

Local, global Stiffness and Mass matrices and right hand side are all computed by **Quasi_static** function or **Time_dependent** function depending on whether we use stationary or unstationary simulations.Those functions return the solution of the problem and the Stiffness/Mass matrices. Code syntax is used as follows :

```
[K_global,second_term,U]=Quasi_static(Q,GammaD,coord,nodes,S,zD,zN,boolean,r1,r2)
```

- Inputs
  - **Q** : sink source.
  - **coord** : node coordinates (in the example given in section 2.2 **coord** was represented by **p**).
  - **nodes** : mesh connectivities (in the example given in section 2.2 **nodes** was represented by **t**).
  - **S** : parameter of the equation, in case of ground water flow it represents hydraulic conductivity, in the case of electrical potential equation, it represents electrical conductivity.
  - **zD** : values of Dirichlet boundary conditions.
  - **zN** : values of Neumann boundary conditions.
  - **boolean** : if **boolean** = 1 parameter **S** is taken into account in the equation as $\frac{10^{-S}}{\rho g}$ where ρ is water density, g is intensity of gravity and parameter **S** represents though in this case the decimal logarithm of hydraulic conductivity. If **boolean** = 0 the parameter **S** is directly included in the equation ; this should be used when we want to solve electrical problem (in this case **S** represents electrical conductivity).
  - **r1, r2** : anisotropy coefficients. (**r1** coefficient in x direction **r2** coefficient in y direction). If **r1**=**r2**=1, parameter is considered as isotropic.
- Outputs
  - **K_global** : matrix problem.
  - **second_term** : right-hand side.
  - **U** : solution of the problem.

3 Inverse problem

The inverse problem consists to recover the horizontal and vertical components of the source current density. The most time-consuming part of the inversion process is the compute of the kernel matrix. In **SPINV** code, this is done with **Kernel** function which take as inputs the inverse problem discretization sizes and gives as output the kernel matrix.

Another important step in the inversion process is the regularization. It can be done with the function named **Regularization**.There are two implemented regularization methods : Tikhonov method and Truncated Singular Value Decomposition method. The regularization parameter is computed using the function named **Regularization_parameter**.This function uses two different methods : L-curve and GCV methods.
For example, suppose we want to use Tikhonov method and computes the regularization parameter with the GCV method, we do like this :

```
Regularization('TIKHONOV','GCV');
```

List of functions used in the code :

Function	Description
bndproj	Project boundary points to true boundary
BOUNDEDGES	Find boundary edges from triangular mesh
center	computes triangle centers coordinates
CIRCUMCENTER	computes the circumcenters of a set of triangles
CSVD	Compact singular value decomposition
dcircle	returns the signed distance of one or more points to a circle
dam_test	solves Darcy's equation coupled with electrical potential equation for the dam study case
ddiff	returns the signed distance to a region that is the difference of two regions
dexpr	computes the signed distance for a general implicit expression
dintersect	sets the signed distance to the intersection of two regions
DISTMESH2D	2-D Mesh Generator using Distance Functions
distmesh2doriginal	2-D Mesh Generator using Distance Functions
DISTMESHND	N-D Mesh Generator using Distance Functions
DISTMESHSURFACE	3-D Surface Mesh Generator using Distance Functions
Divergence_Operator	computes discrete divergence
Divergence_Operator_time	computes transient discrete divergence
dmatrix	returns the signed distance by interpolation from known values on a Cartesian grid
dmatrix3d	returns the signed distance by interpolation from known values on a Cartesian grid in 3D
dpoly	returns the signed distance of one or more points to a polygon
drectangle	returns the signed distance of one or more points to a rectangle
DSEGMENT	computes the distance of points to line segments
dsphere	returns the signed distance of one or more points to a sphere
dunion	returns the signed distance to a union of two regions
edgelist	Compute a list of edges in a triangulation
Fast_Forward_Problem	computes forward problem solution for the inverse problem
fast_geothermy	solves inverse problem for the geothermal study case
Fast_Kernel	computes kernel matrix faster than classical method
fast_synthetic	solves inverse problem for a synthetic test case
find_edge	computes connectivities of nodes of a border
find_element	finds an element (triangle) knowing one of its edges
find_indice	find nodes indices of a boundary

findinterior	Returns a list of which edges in a triangulation are edges
FIXMESH	Remove duplicated/unused nodes and fix element orientation
Forward_Problem	computes forward problem solution for the inverse problem
geothermy	solves inverse problem for the geothermal study case
GET_L	Compute discrete derivative operators
Gradient_operator	computes discrete gradient
hmatrix	computes the mesh size function from values specified on a Cartesian grid
hmatrix3d	computes the mesh size function from values specified on a Cartesian grid in 3D
huniform	returns a uniform mesh size function
interpolation	interpolates electric density current (defined on a grid) on a finite element mesh
inversion_plot	plots exact solution and inverted solution
Kernel	computes kernel matrix
KMG2D	2D-mesh generator using signed distance and size functions
KMG2DREF	Refine a two-dimensional triangular mesh
L_CORNER	Locate the corner of the L-curve
L_CURVE	Plot the L-curve and find its corner
MESHDEMO2d	Distmesh2d examples
MESHDEMOND	distmeshnd examples
MKT2T	Compute element connectivities from element indices
nearest_point	gives index of the nearest point in the mesh to the point which coordinates are (x,y)
nearest_point_border	gives index of the nearest point in the mesh to the point which abscissa is x and which is belonging to the boundary "border"
Neumann_border	computes integrals of shape functions on a Neumann border for P1 finite elements
Neumann_borderP2	computes integrals of shape functions on a Neumann border for P2 finite elements
normal_orientation	computes outward normal (using cosine directions) on an edge
normal_orientation_border	find nodes indexes of a boundary
plot_num	plots linear finite elements solution

plot_QFEM	plots quadratic finite elements solution
plot_QFEM_interp	interpolates a distribution "data" (defined on a grid) on a quadratic finite element mesh and plots it
plot_solution	plots solution on a finite element mesh
plot_tri_interp	this function interpolates a distribution "data" (defined on a grid)on a linear finite element mesh and plots it
plot_vector	plots vector defined on a grid
plotmesh_num	plotmesh plots an unstrucured grid in 2D
plotmesh_tru	plotmesh plots an unstrucured grid in 2D
protate	rotates a set of points by a given angle
pshift	shifts a set of points by a given increment
Quadratic	solves Darcy equation on a square using quadratic finite element
Quasi_static	computes stiffness matrix, right hand-side and solution U
Quasi_static_bis	computes right hand-side and solution U while stiffness matrix is already known
Quasi_staticP2	computes stiffness matrix, right hand-side and solution U using P2 finite
Regularization	computes the regularized solution of optimization problem
Regularization_parameter	computes regularization parameter
Regularization_term	computes the regularization term
scdm	evaluates a function f on a set of points
secondM	computes time-dependent sink source
simpplot	simplex plot
simpqual	Simplex quality
SIMPVOL	Simplex volume
SURFTRI	Find surface triangles from tetrahedral mesh
synthetic	solves inverse problem for a synthetic test case

tarea	computes triangle area
Tikhonov_gcv_function	computes GCV function for Tikhonov regularization
Time_dependent	computes stiffness matrix, right hand-side and solution for time-dependent problem
Time_dependentP2	computes stiffness matrix, right hand-side and solution for time-dependent problem using P2 fem
Transformation	transforms a general inverse problem into a standard one
Transformation_back	computes solution of standard inverse problem using its general formulation
Triangulation_order6_contour	contour plots data at the nodes of a six node triangle
Triangulation_order6_to_order3	linearizes a quadratic triangulation
uniformity	determines the uniformity of a mesh
UNIREF	Uniform mesh refinement
Velocity	computes Darcy's velocity
volcano_test	solves Darcy's equation coupled with electrical potential equation for the volcano study case