A Gauss Implementation of Hopscotch Methods
The PDE2D library

Thierry Roncalli

Financial Econometric Research Centre
City University Business School
Frobisher Crescent
Barbican Center
London EC2Y 8HB

Adam Kurpiel

L.A.R.E
Université Montesquieu-Bordeaux IV
Avenue Léon Duguit
33608 Pessac
France

October 26, 1998

Contents

1. INTRODUCTION

1.1 Installation

1. The file *pde2d.zip* is a zipped archive file. Copy this file under the root directory of Gauss, for example **C:\GAUSS**.
2. Unzip it with archive mode. It is automatically recognized by WinZip. With Unzip or PKunzip, use the -d flag

   pkunzip -d pde2d.zip

   Directories will then be created and files will be copied over them:

target_path	*readme.pde*
*target_path***lib**	library file
*target_path***examples****pde2d**	example and tutorial files
*target_path***src****pde2d**	source code files

3. Run Gauss. **Log on to the *src**pde2d* directory**[1] and add the path to the library file *pde2d.lcg* in the following way:

   lib pde2d /addpath

1.2 Getting started

Gauss 3.2.20+ for OS/2, Windows NT/95 or Unix is required to use the **PDE2D** routines.

1.2.1 readme.PDE file

The file *readme.PDE* contains last minute information on the **PDE2D** procedures. Please read it before using them.

1.2.2 Setup

In order to use these procedures, the **PDE2D** library must be active. This is done by including **PDE2D** in the LIBRARY statement at the top of your program:

library PDE2D;

To reset global variables in subsequent executions of the program, the following instruction should be used:

PDE2Dset;

[1] You may use the commands `ChangeDir` or `chdir` and you may verify that you are in the *src**pde2d* directory with the `cdir(0)` command.

If you plan to make any right-hand reference to the global variables, you will also need the statement:

#include *target_path***src****pde2d****pde2d.ext;**

The **PDE2D** version number is stored as a global variable:

_PDE2D_ver	3×1 matrix where the first element indicates the major version number, the second element the minor version number, and the third element the revision number

1.3 What is PDE2D ?

PDE2D is a Gauss library for solving Parabolic and Elliptic Partial Differential Equations (PDE) in 2 space dimensions. It includes Hopscotch and θ-schemes algorithms with finite difference methods.

PDE2D contains the procedures whose list is given below. See the command reference part for a full description.

- **Derive**: Computes the derivative of the data with respect to the mesh spacing h
- **ExtractSolution**: Extracts solution $u_{i,j}^m$ for a specific value of t, x or y
- **FindIndex**: Returns the indices of the elements of a vector x equal to the elements of a vector v
- **Hopscotch**: Solves the PDE problem with Hopscotch methods
- **PDE2D**: Initializes the PDE problem
- **PDE2Dset**: Resets defaults

1.4 Using Online Help

PDE2D library supports Windows Online Help. Before using the browser, you have to verify that the **PDE2D** library is activated by the `library` command.

2. PARTIAL DIFFERENTIAL EQUATIONS IN 2 SPACE DIMENSIONS

The library **PDE2D** is a Gauss implementation of the Hopscotch methods described in GOURLAY and MCKEE [1977] and KURPIEL and RONCALLI [1998]. The reader may refer to the second article to understand the notations used in this manual.

2.1 The PDE problem

The PDE problem consists of the linear parabolic equation

$$\frac{\partial u(t,x,y)}{\partial t} + f(t,x,y)\, u(t,x,y) = \mathcal{A}_t u(t,x,y) + g(t,x,y) \tag{2.1}$$

where $\mathcal{A}_t$ is the general two-space dimensions differential operator

$$\begin{aligned} \mathcal{A}_t u(t,x,y) \quad = \quad & a(t,x,y)\frac{\partial^2 u(t,x,y)}{\partial x^2} + 2b(t,x,y)\frac{\partial^2 u(t,x,y)}{\partial x \partial y} + c(t,x,y)\frac{\partial^2 u(t,x,y)}{\partial y^2} + \\ & d(t,x,y)\frac{\partial u(t,x,y)}{\partial x} + e(t,x,y)\frac{\partial u(t,x,y)}{\partial y} \end{aligned} \tag{2.2}$$

The **PDE2D** library solves the problem (2.1) in the region of the (t,x,y) space given by $\mathfrak{T} \times \mathfrak{R}$ with

$$\mathfrak{R} = \left[x^-, x^+\right] \times \left[y^-, y^+\right] \tag{2.3}$$

and

$$\mathfrak{T} = \left[t^-, t^+\right] \tag{2.4}$$

We could impose Dirichlet or Neumann conditions:

$$\begin{aligned} & u\left(t^-,x,y\right) = u_{(t^-)}(x,y) \\ & u\left(t,x^-,y\right) = u_{(x^-)}(t,y) \quad \bigvee \quad \left.\frac{\partial u(t,x,y)}{\partial x}\right|_{x=x^-} = u'_{(x^-)}(t,y) \\ & u\left(t,x^+,y\right) = u_{(x^+)}(t,y) \quad \bigvee \quad \left.\frac{\partial u(t,x,y)}{\partial x}\right|_{x=x^+} = u'_{(x^+)}(t,y) \\ & u\left(t,x,y^-\right) = u_{(y^-)}(t,x) \quad \bigvee \quad \left.\frac{\partial u(t,x,y)}{\partial y}\right|_{y=y^-} = u'_{(y^-)}(t,x) \\ & u\left(t,x,y^+\right) = u_{(y^+)}(t,x) \quad \bigvee \quad \left.\frac{\partial u(t,x,y)}{\partial y}\right|_{y=y^+} = u'_{(y^+)}(t,x) \end{aligned} \tag{2.5}$$

To initialize the PDE problem, we use the `PDE2D` procedure

```
call PDE2D(&aProc,&bProc,&cProc,&dProc,&eProc,&fProc,&gProc,&hProc,
           &tminBound,&xminBound,&xmaxBound,&yminBound,&ymaxBound,
           &DxminBound,&DxmaxBound,&DyminBound,&DymaxBound);
```

The general form of the procedures is

```
proc (1) = aProc(t,x,y);
  local a;

  a =

  retp(a);
endp;
```

Remark 1 *In the **PDE2D** library, x and y are treated respectively as a $N_x \times 1$ column vector and a $1 \times N_x$ row vector and the procedures* `*Proc` *must return a $N_x \times N_y$ matrix.*

h is a special function. If it is not initialized to 0, the Hopscotch algorithms use this function at each iteration m to modify the numerical solution

$$\mathbf{u}_m := h\left(t, x, y, \mathbf{u}_m\right)$$

The form of the `hProc` procedure is also

```
proc (1) = hProc(t,x,y,u);
  local h;

  h =

  retp(h);
endp;
```

For each bound, you have to specify a boundary condition, either a Dirichlet or a Neumann Condition. For example, if we have the following command line

```
call PDE2D(&aProc,&bProc,&cProc,&dProc,&eProc,&fProc,&gProc,&hProc,
           &tminBound,0,&xmaxBound,&yminBound,0,&DxminBound,0,0,0);
```

Dirichlet conditions are imposed for $x = x^+$ and $y = y^-$ and we put a user-defined Neumann condition on $x = x^-$. Because `ymaxBound` and `DymaxBound` are both set to 0, `PDE2D` uses the usual Neumann condition

$$\left.\frac{\partial\, u\left(t, x, y\right)}{\partial\, y}\right|_{y=y^+} = 0$$

Remark 2 *The `PDE2D` procedure prints information about the boundary nature of the PDE problem if `__output` is set to 1.*

For the precedent example, we have

```
=========================================================
Bound             Dirichlet             Neumann

xmin                                   *********
xmax              *********
ymin              *********
ymax                                         0
=========================================================
```

2.2 The Hopscotch algorithms

The procedure `Hopscotch` enables you to solve the PDE problem. Its syntax is

```
call Hopscotch(tmin,tmax,Nt,xmin,xmax,Nx,ymin,ymax,Ny,NameFile);
```

The variables `tmin`, `tmax`, `xmin`, `xmax`, `ymin` and `ymax` correspond to t^-, t^+, x^-, x^+, y^- and y^+. `Nt`, `Nx` and `Ny` are respectively the number of discretisation points for the time t (N_t) and the space directions x (N_x) and y (N_y). The numerical solution of the PDE problem $u^m_{i,j}$ is stored in the `NameFile` dataset in the following way:

	$x_0 = x^-$	x_1	x_2	$\cdots$	x_{N_x-2}	$x_{N_x-1} = x^+$							
	$y_0 = y^-$	y_1	y_2	$\cdots$	y_{N_y-2}	$y_{N_y-1} = y^+$							
$t_0 = t^-$	$u^0_{0,0}$	$u^0_{1,0}$	$u^0_{2,0}$	$\cdots$	$u^0_{N_x-2,0}$	$u^0_{N_x-1,0}$	$u^0_{0,1}$	$\cdots$	$u^0_{N_x-1,1}$	$\cdots$	u^0_{0,N_y-1}	$\cdots$	$u^0_{N_x-1,N_y-1}$
t_1	$u^1_{0,0}$	$u^1_{1,0}$	$u^1_{2,0}$	$\cdots$	$u^1_{N_x-2,0}$	$u^1_{N_x-1,0}$	$u^1_{0,1}$	$\cdots$	$u^1_{N_x-1,1}$	$\cdots$	u^1_{0,N_y-1}	$\cdots$	$u^1_{N_x-1,N_y-1}$
$\vdots$				$\vdots$				$\vdots$		$\vdots$		$\vdots$	
$t_{N_t-1} = t^+$	$u^1_{0,0}$	$u^1_{1,0}$	$u^1_{2,0}$	$\cdots$	$u^1_{N_x-2,0}$	$u^1_{N_x-1,0}$	$u^1_{0,1}$	$\cdots$	$u^1_{N_x-1,1}$	$\cdots$	u^1_{0,N_y-1}	$\cdots$	$u^1_{N_x-1,N_y-1}$

with

$$\begin{aligned} t_m &= t^- + m \cdot k \\ x_i &= x^- + i \cdot h_x \\ y_j &= y^- + j \cdot h_y \end{aligned}$$

and

$$k = \frac{t^+ - t^-}{N_t - 1}$$

$$h_x = \frac{x^+ - x^-}{N_x - 1}$$

$$h_y = \frac{y^+ - y^-}{N_y - 1}$$

The first and second rows of the dataset contain the N_x values x_i and N_y values y_j. The t_m and $u^m_{i,j}$ values are stored in the next N_t rows. Let u^m be the matrix with the (i,j) entry $\left(u^m_{i,j}\right)$. Then, the storage method corresponds to the following stacking method

$$\left[\begin{array}{cc} t_m & \mathrm{vec}\left(u^m\right)^\top \end{array} \right]$$

Remark 3 *The `Hopscotch` procedure prints information about mesh ratios if `__output` is set to 1.*

This is an output of the `Hopscotch` procedure :

```
Filling Hopscotch method: left-right
Theta Hopscotch method: ordered odd-even

Mesh spacing in time    :  k =   0.05000000
Mesh spacing in space x : hx =   0.10000000
Mesh spacing in space y : hy =   0.10000000

Mesh ratio r(x,x):     5.000000
           r(x,y):     5.000000
           r(y,y):     5.000000


Hopscotch completed
```

Remark 4 *You could use the `_Hopscotch_Elliptic` variable to specify that the PDE problem is an elliptic problem. In this case, `Hopscotch` stops iterations if the following condition is verified*

$$\mathbf{u}_{m+1} = \mathbf{u}_m$$

Note that `Hopscotch` uses the fuzzy comparison function `feq` to perform the test. You could also modify the value taken by `_fcmptol`.

Remark 5 *You may access to the mesh ratios and mesh spacing with the data buffer* `_Hopscotch_MeshRatios`. *They may be extracted with the* `vread` *command using the following strings:*

`''k''`	*Mesh spacing* k *in time* t
`''hx''`	*Mesh spacing* h_x *in space* x
`''hy''`	*Mesh spacing* h_y *in space* y
`''r(x,x)''`	*Mesh ratio* $r\,(x,x) = \frac{k}{h_x h_x}$
`''r(x,y)''`	*Mesh ratio* $r\,(x,y) = \frac{k}{h_x h_y}$
`''r(y,y)''`	*Mesh ratio* $r\,(y,y) = \frac{k}{h_y h_y}$

Remark 6 `_Hopscotch_Methods` *is the indicator variable determining the filling and theta Hopscotch methods. It is a* 2×1 *vector. You could set* `_Hopscotch_Methods[1]` *to 1 for the left-right method, 2 for the center method or to a pointer to a procedure that computes a user-defined method. The syntax of this procedure is*

```
{deltaMinusMinus,deltaZeroMinus,deltaPlusMinus,
 deltaMinusZero,deltaZeroZero,deltaPlusZero,
 deltaMinusPlus,deltaZeroPlus,deltaPlusPlus} =
 procFillingMethod(a_ij,b_ij,c_ij,d_ij,e_ij,hx,hy,Nx,Ny);
```

The theta Hopscotch method is determined by `_Hopscotch_Methods[2]`. *It takes the values* $0 - 1$ *for a* θ*-method (explicit* $= 0$ *– Crank-Nicholson* $= 0.5$ *– implicit* $= 1$*), 2 for the ordered odd-even method and 3 for the line method. It may also correspond to a pointer to a procedure if you want to employ a user-defined theta method. The syntax of this procedure is also*

```
{theta_mm,theta_m} = procThetaMethod(i,j,m);
```

Remark 7 *For some PDE problems, computations may be very long. You could use the* `_Hopscotch_PrintIters` *to check the number of performed iterations.*

Remark 8 *The* `_Hopscotch_Retcode` *is a* $N_t \times 1$ *vector which contains the sparse system termination condition given by the* `SparseSolve` *procedure. The possible values of* `_Hopscotch_Retcode[m+1]` *are:*

- 0 – $\mathbf{u}_{m+1}$ *is the exact solution, no iterations performed.*
- 1 – $\mathbf{u}_{m+1}$ *is nearly exact with accuracy on the order of* `_sparse_Atol` *and* `_sparse_Btol`.
- 2 – $\mathbf{u}_{m+1}$ *is not exact and a least square solution has been found with accuracy on the order of* `_sparse_Atol`.
- 3 – *The estimate of the condition of* Ψ_{m+1} *has exceeded* `_sparse_CondLimit`.*The system appears to be ill-conditioned.*
- 4 – $\mathbf{u}_{m+1}$ *is nearly exact with reasonable accuracy.*
- 5 – $\mathbf{u}_{m+1}$ *is not exact and a least squares solution has been found with reasonable accuracy.*
- 6 – *Iterations halted due to poor condition given machine precision.*
- 7 – `_sparse_NumIters` *exceeded.*

Remark 9 *If you would to save only the last iteration solution, you could use the following syntax*

```
_Hopscotch_SaveLastIter = 1
```

In this case, the `NameFile` *dataset becomes*

	$x_0 = x^-$	x_1	x_2	$\cdots$	x_{N_x-2}	$x_{N_x-1} = x^+$							
	$y_0 = y^-$	y_1	y_2	$\cdots$	y_{N_y-2}	$y_{N_y-1} = y^+$							
$t_{N_t-1} = t^+$	$u^1_{0,0}$	$u^1_{1,0}$	$u^1_{2,0}$	$\cdots$	$u^1_{N_x-2,0}$	$u^1_{N_x-1,0}$	$u^1_{0,1}$	$\cdots$	$u^1_{N_x-1,1}$	$\cdots$	u^1_{0,N_y-1}	$\cdots$	$u^1_{N_x-1,N_y-1}$

Remark 10 *If* `_Hopscotch_Convergence` *is not equal to 0, the procedure checks the* `_sparse_Retcode` *value at each iteration and stops if it encounters convergence problems.*

2.3 Solution extracting

You could of course use the Gauss commands to extract solution from the data set `NameFile`. The `ExtractSolution` procedure is provided to make it easier. Its syntax is

```
U = ExtractSolution(NameFile,cn);
```

The variable `cn` could take differents values. If `cn` is the string `''t''`, then `U` corresponds to the column vector $\{t_m\}$. We obtain the column vector $\{x_i\}$ or the row vector $\{y_j\}$ by setting `cn` to `''x''` or `''y''`. We could also extract specific solutions $u^m_{i,j}$ by using a 2×1 vector. We have

cn	U
`''t|tm''`	$N_x \times N_y$ matrix with entry the (i,j) entry $\left(u^m_{i,j}\right)$
`''x|xi''`	$N_t \times N_y$ matrix with entry the (m,j) entry $\left(u^m_{i,j}\right)$
`''y|yj''`	$N_x \times N_t$ matrix with entry the (i,m) entry $\left(u^m_{i,j}\right)$

2.4 `Derive` and `FindIndex` procedures

We could employ the procedure `Derive` to compute the derivative matrix $\mathcal{D}^u$ of a $N \times M$ data matrix $u = (u_{i,j})$. Its syntax is

```
Du = Derive(u,h,mtd);
```

Let h be the mesh spacing. For $2 \leqslant i \leqslant N-1$, we have

$$\mathcal{D}^u_{i,j} = \frac{u_{i,j} - u_{i-1,j}}{h}$$

for the **left** difference method (`mtd = -1`),

$$\mathcal{D}^u_{i,j} = \frac{u_{i+1,j} - u_{i-1,j}}{2h}$$

for the **center** difference method (`mtd = 0`) and

$$\mathcal{D}^u_{i,j} = \frac{u_{i+1,j} - u_{i,j}}{h}$$

for the **right** difference method (`mtd = +1`). The derivatives for $i = 1$ and $i = N$ are **always** computed with the following formulas

$$\mathcal{D}^u_{1,j} = \frac{u_{2,j} - u_{1,j}}{h}$$

and

$$\mathcal{D}^u_{N,j} = \frac{u_{N,j} - u_{N-1,j}}{h}$$

Remark 11 *This procedure is useful to compute numerically the greek coefficients for Stochastic Volatility options.*

`FindIndex` returns the indices of the elements of a vector x equal to the elements of a vector v. To understand how the procedure works, let's try an example:

```
new;
library pde2D;

xmin = -3;
xmax = 3;
Nx = 101;
hx = (xmax-xmin)/(Nx-1);

x = seqa(xmin,hx,Nx);

FindIndex(x,0|3);
       51.000000
       101.00000
indexcat(x,0);
          .
indexcat(x,3);
       101.00000
```

The `indexcat` procedure does not find the index of an element of x equal to 0 due to numerical truncation. In this case, you may use the `FindIndex` procedure.

3. COMMAND REFERENCE

The following global variables and procedures are defined in **PDE2D**. They are the *reserved words* of **PDE2D**.

BandTimesVector, _Center_FillingMethod, Derive, DiagPlusBand, ExtractSolution, FindIndex, Hopscotch, _Hopscotch, __Hopscotch, _Hopscotch_Convergence, _Hopscotch_MeshRatios, _Hopscotch_Methods, _Hopscotch_PrintIters, _Hopscotch_RetCode, _Hopscotch_SaveLastIter, _Hopscotch_Theta, isEven, isOdd, _LeftRight_FillingMethod, _Line_ThetaMethod, _Odd-Even_ThetaMethod, PDE2D, _pde2d_aProc, _pde2d_bProc, _pde2d_Build, _pde2d_cProc, _pde2d_DerivCond,_pde2d_dProc, _pde2d_DxmaxBound, _pde2d_DxminBound, _pde2d_Dymax-Bound, _pde2d_DyminBound, _pde2d_eProc, _pde2d_fProc, _pde2d_gProc, _pde2d_hProc, _pde2d_Neumann, PDE2Dset, _pde2d_tminBound, _pde2d_ver, _pde2d_xmaxBound, _pde2d_xminBound, _pde2d_ymaxBound, _pde2d_yminBound, SparseBandMatrix, _Theta_Theta-Method, _UstarProc

The default global control variables are

`_Hopscotch_Elliptic`	0
`_Hopscotch_Methods`	{1,2}
`_Hopscotch_PrintIters`	0
`_Hopscotch_SaveLastIter`	0
`_PDE2D_Build`	0

Derive

■ **Purpose**
Computes the derivative of the data with respect to the mesh spacing h.

■ **Format**
Du = Derive(u,h,mtd);

■ **Input**

u	$N \times M$ matrix
h	scalar, mesh spacing h
mtd	scalar
	-1 for the left difference method
	0 for the center difference method
	1 for the right difference method

■ **Output**

Du	$N \times M$ matrix, derivative matrix $\mathcal{D}^u$

■ **Source**
pde2d.src

ExtractSolution

■ **Purpose**
Extracts solution $u_{i,j}^m$ of the PDE problem for a specific value of t, x or y.

■ **Format**
U = ExtractSolution(NameFile,cn);

■ **Input**

NameFile	string, name of the solution dataset file
cn	scalar or vector 2×1

■ **Output**

U	$N_t \times 1$ vector, the t_m values if cn is the string `"t"` $N_x \times 1$ vector, the x_i values if cn is the string `"x"` $1 \times N_y$ vector, the y_j values if cn is the string `"y"`
	– or –
	$N_x \times N_y$ matrix, the $u(t_m, x, y)$ values if cn is equal `"t"|tm` $N_t \times N_y$ matrix, the $u(t, x_i, y)$ values if cn is equal to `"x"|xi` $N_x \times N_t$ matrix, the $u(t, x, y_j)$ values if cn is equal to `"y"|yj`

■ **Source**
pde2d.src

FindIndex

Purpose

Returns the indices of the elements of a vector x equal to the elements of a vector v.

Format

y = FindIndex(x,v);

Input

x	$N \times 1$ vector
v	$L \times 1$ vector

Output

y	$L \times 1$ vector, `y[i]` contains the indice of the first element of x which is equal to `v[i]`

Globals

_fcmptol	scalar (default = 1e-15) the procedure `FindIndex` uses _fcmptol to fuzz the comparison operations to allow for round off error

Remarks

The procedure `FindIndex` is similar to the Gauss `indexcat` command. The main difference is that `FindIndex` returns only one index (or a missing value) for each value v_i. Note that the global variable `_fcmptol` is used to check the $x_{y_i} = v_i$ equality relation.

Source

pde2d.src

Hopscotch

Purpose
Solves the PDE problem with Hopscotch methods.

Format
call Hopscotch(tmin,tmax,Nt,xmin,xmax,Nx,ymin,ymax,Ny,NameFile);

Input

tmin	scalar, value of $t_{\min}$.
tmax	scalar, value of $t_{\max}$.
Nt	scalar, number of discretization points in time.
xmin	scalar, value of $x_{\min}$.
xmax	scalar, value of $x_{\max}$.
Nx	scalar, number of discretization points in x direction.
ymin	scalar, value of $y_{\min}$.
ymax	scalar, value of $y_{\max}$.
Ny	scalar, number of discretization points in y direction.
NameFile	string, name of the solution dataset file.

Output

Globals

_Hopscotch_Elliptic scalar (default = 0)
0 if the PDE problem is not an elliptic problem
1 if the PDE problem is an elliptic problem

_Hopscotch_MeshRatios
vector, created using `VPUT`. Contains mesh ratios $r(x,x)$, $r(x,y)$, $r(y,y)$ and mesh spacing k, h_x and h_y. They may be extracted with the `VREAD` command using the following strings:

`"r(x,x)"` — Mesh ratio $r(x,x) = k/(h_x h_x)$
`"r(x,y)"` — Mesh ratio $r(x,y) = k/(h_x h_y)$
`"r(y,y)"` — Mesh ratio $r(y,y) = k/(h_y h_y)$

`"k"` — Mesh spacing in time
`"hx"` — Mesh spacing in space x
`"hy"` — Mesh spacing in space y

_Hopscotch_Methods 2×1 vector (default = {1,2})

_Hopscotch_Methods[1] — the filling Hopscotch method
1 for the left-rigth method,
2 for the center method
or a pointer to a procedure that computes a user-defined filling method

_Hopscotch_Methods[2] — the theta Hopscotch method
0-1 for a θ-method (explicit = 0, Crank-Nicholson = 0.5 and implicit = 1),
2 for the ordered odd-even method,
3 for the line method
or a pointer to a procedure that computes a user-defined theta method

_Hopscotch_PrintIters	scalar (default = 0) 0 – does not print iterations I – printing after each I iterations
_Hopscotch_Retcode	$N_t \times 1$ vector, sparse solver return codes vector (see `SparseSolve` for more details)
_Hopscotch_SaveLastIter	scalar (default = 0) 0 for saving the solution for all the iterations m 1 for saving only the last solution for $t_m = t^+$
__output	scalar (default = 0) 1 – print informations about the algorithm and the mesh ratios 0 – no printing

■ **Remarks**

To extract the solution, you may use the `ExtractSolution` procedure.

■ **Source**

pde2d.src

PDE2D

Purpose

Initializes the PDE problem.

Format

call PDE2D(aProc,bProc,cProc,dProc,eProc,fProc,gProc,hProc,
tminBound,xminBound,xmaxBound,yminBound,ymaxBound,
DxminBound,DxmaxBound,DyminBound,DymaxBound);

Input

aProc	scalar, pointer to a procedure which computes $a(t,x,y)$
bProc	scalar, pointer to a procedure which computes $b(t,x,y)$
cProc	scalar, pointer to a procedure which computes $c(t,x,y)$
dProc	scalar, pointer to a procedure which computes $d(t,x,y)$
eProc	scalar, pointer to a procedure which computes $e(t,x,y)$
fProc	scalar, pointer to a procedure which computes $f(t,x,y)$
gProc	scalar, pointer to a procedure which computes $g(t,x,y)$
hProc	scalar, pointer to a procedure which computes $h(t,x,y)$ – or – scalar 0
tminBound	scalar, pointer to a procedure which computes $u_{(t-)}(x,y)$
xminBound	scalar, pointer to a procedure which computes $u_{(x-)}(t,y)$
xmaxBound	scalar, pointer to a procedure which computes $u_{(x+)}(t,y)$
yminBound	scalar, pointer to a procedure which computes $u_{(y-)}(t,x)$
ymaxBound	scalar, pointer to a procedure which computes $u_{(y+)}(t,x)$
DxminBound	scalar, pointer to a procedure which computes $u'_{(x-)}(t,y)$
DxmaxBound	scalar, pointer to a procedure which computes $u'_{(x+)}(t,y)$
DyminBound	scalar, pointer to a procedure which computes $u'_{(y-)}(t,x)$
DymaxBound	scalar, pointer to a procedure which computes $u'_{(y+)}(t,x)$

Output

Globals

__output	scalar 1 – print information about the PDE problem 0 – no printing

Source

pde2d.src

PDE2Dset

■ **Purpose**
Resets the global control variables declared in *PDE2D.DEC*.

■ **Format**
PDE2Dset;

■ **Remarks**
The default global control variables are

_Hopscotch_Elliptic	0
_Hopscotch_Methods	{1,2}
_Hopscotch_PrintIters	0
_Hopscotch_SaveLastIter	0
_PDE2D_Build	0

■ **Source**
pde2d.src

4. TUTORIAL

4.1 The PDE2D procedure

We consider the linear parabolic PDE problem defined by

$$\begin{aligned} a(t,x,y) &= \frac{1}{2}x^2 + y^2 \\ b(t,x,y) &= -\frac{1}{2}\left(x^2+y^2\right) \\ c(t,x,y) &= x^2 + \frac{1}{2}y^2 \\ d(t,x,y) &= x \\ e(t,x,y) &= -y \\ f(t,x,y) &= 1 \\ g(t,x,y) &= 2xy^2e^{-t} \end{aligned}$$

$\mathfrak{R}$ is set to $[0,1]\times[0,1]$ and we have

$$\begin{aligned} u(t,0,y) = 0 \quad &\bigvee \quad u_x(t,0,y) = y^2e^{-t}+1 \\ u(t,1,y) = \left(y+y^2\right)e^{-t}+1 \quad &\bigvee \quad u_x(t,1,y) = \left(2y+y^2\right)e^{-t}+1 \\ u(t,x,0) = x \quad &\bigvee \quad u_y(t,x,0) = x^2e^{-t} \\ u(t,x,1) = \left(x+x^2\right)e^{-t}+x \quad &\bigvee \quad u_x(t,1,y) = \left(2x+x^2\right)e^{-t} \end{aligned}$$

The solution of the Cauchy problem with $u(0,x,y) = x^2y + xy^2 + x$ is

$$u(t,x,y) = \left(x^2y+xy^2\right)e^{-t} + x$$

Remark 12 *The procedures* ****Proc*** *return a $N_x \times N_y$ matrix. So, you must use the element-by-element operators.*

```
/*
** tutor.inc
*/

proc aProc(t,x,y);
  retp( 0.5*x^2 + y^2 );
endp;

proc bProc(t,x,y);
  retp( -0.5*(x^2 + y^2) );
endp;

proc cProc(t,x,y);
  retp( x^2 + 0.5*y^2 );
endp;

proc dProc(t,x,y);
  retp( ones(rows(x),cols(y)).*x );
endp;

proc eProc(t,x,y);
  retp( -ones(rows(x),cols(y)).*y );
endp;

proc fProc(t,x,y);
  retp( ones(rows(x),cols(y)) );
```

```
endp;

proc gProc(t,x,y);
  retp( 2*x.*(y^2).*exp(-t) );
endp;


proc tminBound(t,x,y);
  retp( (x^2).*y + x.*(y^2) + x );
endp;

proc xminBound(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

proc xmaxBound(t,x,y);
  local Nx,Ny;
  Nx = rows(x); Ny = cols(y);
  retp( ones(Nx,Ny).*( (y+y^2)*exp(-t) + 1 ) );
endp;

proc yminBound(t,x,y);
  local Nx,Ny;
  Nx = rows(x); Ny = cols(y);
  retp( zeros(Nx,Ny) + x );
endp;

proc ymaxBound(t,x,y);
  local Nx,Ny;
  Nx = rows(x); Ny = cols(y);
  retp( ones(Nx,Ny).*( (x+x^2)*exp(-t) + x) );
endp;

proc DxminBound(t,x,y);
  local Nx,Ny;
  Nx = rows(x); Ny = cols(y);
  retp( ones(Nx,Ny).*( (y^2)*exp(-t) + 1 ) );
endp;

proc DxmaxBound(t,x,y);
  local Nx,Ny;
  Nx = rows(x); Ny = cols(y);
  retp( ones(Nx,Ny).*( (2*y + y^2)*exp(-t) + 1 ) );
endp;

proc DyminBound(t,x,y);
  local Nx,Ny;
  Nx = rows(x); Ny = cols(y);
  retp( ones(Nx,Ny).*(x^2)*exp(-t) );
endp;

proc DymaxBound(t,x,y);
  local Nx,Ny;
  Nx = rows(x); Ny = cols(y);
  retp( ones(Nx,Ny).*(2*x + x^2)*exp(-t) );
endp;
```

4.1.1 Dirichlet conditions

Dirichlet conditions are imposed in the following way:

```
/*
** tutor1.prg
*/

new;
```

```
library PDE2D;

PDE2Dset;

#include tutor.inc;

output file = tutor1.out reset;

call PDE2D(&aProc,&bProc,&cProc,&dProc,&eProc,&fProc,&gProc,0,
           &tminBound,&xminBound,&xmaxBound,&yminBound,&ymaxBound,
           0,0,0,0);

output off;


=========================================================
Bound             Dirichlet            Neumann

xmin              *********
xmax              *********
ymin              *********
ymax              *********
=========================================================
```

4.1.2 Neumann conditions

Neumann conditions are imposed in the following way:

```
/*
** tutor2.prg
*/

new;
library PDE2D;

PDE2Dset;

#include tutor.inc;

output file = tutor2.out reset;

call PDE2D(&aProc,&bProc,&cProc,&dProc,&eProc,&fProc,&gProc,0,
           &tminBound,0,0,0,0,
           &DxminBound,&DxmaxBound,&DyminBound,&DymaxBound);

output off;


=========================================================
Bound             Dirichlet            Neumann

xmin                                  *********
xmax                                  *********
ymin                                  *********
ymax                                  *********
=========================================================
```

4.1.3 Mixing Dirichlet and Neumann conditions

Dirichlet and Neumann boundary conditions may be mixed:

```
/*
** tutor3.prg
*/

new;
library PDE2D;

PDE2Dset;

#include tutor.inc;

output file = tutor3.out reset;

call PDE2D(&aProc,&bProc,&cProc,&dProc,&eProc,&fProc,&gProc,0,
           &tminBound,&xminBound,0,0,0,
           0,&DxmaxBound,&DyminBound,&DymaxBound);

output off;


=========================================================
Bound             Dirichlet            Neumann

xmin              *********
xmax                                  *********
ymin                                  *********
ymax                                  *********
=========================================================
```

Remark 13 *Remind that the Dirichlet and Neumann conditions must be compatible.*

```
/*
** tutor4.prg
*/

new;
library PDE2D;

PDE2Dset;

#include tutor.inc;

output file = tutor4.out reset;

call PDE2D(&aProc,&bProc,&cProc,&dProc,&eProc,&fProc,&gProc,0,
           &tminBound,&xminBound,0,0,&ymaxBound,
           0,&DxmaxBound,&DyminBound,&DymaxBound);

output off;


error: too many Dirichlet and Neumann Conditions
Currently active call: PDE2D [130]
```

4.2 The Hopscotch procedure

We solve the PDE problem with the **Hopscotch** procedure. In the following program, we set $t^+ = 1$, $N_t = 11$, $N_x = 4$ and $N_y = 5$. The solution values are stored in the dataset *tutor5*.

```
/*
** tutor5.prg
*/

new;
```

```
library PDE2D;

PDE2Dset;

#include tutor.inc;

output file = tutor5.out reset;

call PDE2D(&aProc,&bProc,&cProc,&dProc,&eProc,&fProc,&gProc,0,
           &tminBound,&xminBound,&xmaxBound,&yminBound,&ymaxBound,
           0,0,0,0);

xmin = 0; xmax = 1; Nx = 4;
ymin = 0; ymax = 1; Ny = 5;
tmin = 0; tmax = 1; Nt = 11;

call Hopscotch(tmin,tmax,Nt,xmin,xmax,Nx,ymin,ymax,Ny,''tutor5'');

sol = loadd(''tutor5'');

call printfm(sol,1,''*.*lf'' ~5 ~2);

output off;


=========================================================
Bound             Dirichlet            Neumann

xmin              *********
xmax              *********
ymin              *********
ymax              *********
=========================================================

Filling Hopscotch method: left-right
Theta Hopscotch method: ordered odd-even

Mesh spacing in time    :  k =   0.10000000
Mesh spacing in space x : hx =   0.33333333
Mesh spacing in space y : hy =   0.25000000

Mesh ratio r(x,x):     0.900000
           r(x,y):     1.200000
           r(y,y):     1.600000

Hopscotch completed

 0.00 0.33 0.67 1.00    .    .    .    .    .    .    .    .    .    .    .    .    .    .    .    .    .
 0.00 0.25 0.50 0.75 1.00    .    .    .    .    .    .    .    .    .    .    .    .    .    .    .    .
 0.00 0.00 0.33 0.67 1.00 0.00 0.38 0.82 1.31 0.00 0.47 1.06 1.75 0.00 0.60 1.38 2.31 0.00 0.78 1.78 3.00
 0.10 0.00 0.33 0.67 1.00 0.00 0.38 0.80 1.28 0.00 0.46 1.02 1.68 0.00 0.58 1.30 2.19 0.00 0.74 1.67 2.81
 0.20 0.00 0.33 0.67 1.00 0.00 0.37 0.79 1.26 0.00 0.45 0.98 1.61 0.00 0.55 1.25 2.07 0.00 0.70 1.58 2.64
 0.30 0.00 0.33 0.67 1.00 0.00 0.37 0.78 1.23 0.00 0.43 0.95 1.56 0.00 0.53 1.19 1.97 0.00 0.66 1.49 2.48
 0.40 0.00 0.33 0.67 1.00 0.00 0.37 0.77 1.21 0.00 0.43 0.92 1.50 0.00 0.51 1.14 1.88 0.00 0.63 1.41 2.34
 0.50 0.00 0.33 0.67 1.00 0.00 0.36 0.76 1.19 0.00 0.42 0.90 1.45 0.00 0.50 1.09 1.80 0.00 0.60 1.34 2.21
 0.60 0.00 0.33 0.67 1.00 0.00 0.36 0.75 1.17 0.00 0.41 0.88 1.41 0.00 0.48 1.05 1.72 0.00 0.58 1.28 2.10
 0.70 0.00 0.33 0.67 1.00 0.00 0.36 0.74 1.16 0.00 0.40 0.86 1.37 0.00 0.47 1.02 1.65 0.00 0.55 1.22 1.99
 0.80 0.00 0.33 0.67 1.00 0.00 0.35 0.73 1.14 0.00 0.40 0.84 1.34 0.00 0.45 0.98 1.59 0.00 0.53 1.17 1.90
 0.90 0.00 0.33 0.67 1.00 0.00 0.35 0.73 1.13 0.00 0.39 0.82 1.30 0.00 0.44 0.95 1.53 0.00 0.51 1.12 1.81
 1.00 0.00 0.33 0.67 1.00 0.00 0.35 0.72 1.11 0.00 0.38 0.81 1.28 0.00 0.43 0.93 1.48 0.00 0.50 1.08 1.74
```

4.2.1 The _Hopscotch_MeshRatios data buffer

```
/*
** tutor6.prg
*/
```

```
new;
library PDE2D;

PDE2Dset;

#include tutor.inc;

__output = 0;
output file = tutor6.out reset;

call PDE2D(&aProc,&bProc,&cProc,&dProc,&eProc,&fProc,&gProc,0,
           &tminBound,&xminBound,&xmaxBound,&yminBound,&ymaxBound,
           0,0,0,0);

xmin = 0; xmax = 1; Nx = 21;
ymin = 0; ymax = 1; Ny = 11;
tmin = 0; tmax = 1; Nt = 101;

call Hopscotch(tmin,tmax,Nt,xmin,xmax,Nx,ymin,ymax,Ny,''tutor6'');

k = vread(_Hopscotch_MeshRatios,''k'');
hx = vread(_Hopscotch_MeshRatios,''hx'');
hy = vread(_Hopscotch_MeshRatios,''hy'');
rxx = vread(_Hopscotch_MeshRatios,''r(x,x)'');
rxy = vread(_Hopscotch_MeshRatios,''r(x,y)'');
ryy = vread(_Hopscotch_MeshRatios,''r(y,y)'');

print ftos(k ,''k = %lf'',6,3);
print ftos(hx,''hx = %lf'',6,3);
print ftos(hy,''hy = %lf'',6,3);
print;
print ftos(rxx,''r(x,x): %lf'',6,3);
print ftos(rxy,''r(x,y): %lf'',6,3);
print ftos(ryy,''r(y,y): %lf'',6,3);

output off;


k =  0.010
hx =  0.050
hy =  0.100

r(x,x):  4.000
r(x,y):  2.000
r(y,y):  1.000
```

4.2.2 The _Hopscotch_Methods variable

If you prefer to use the center filling method and the line Hopscotch scheme, you may modify the program as following

```
_Hopscotch_Methods = {2,3};
call Hopscotch(tmin,tmax,Nt,xmin,xmax,Nx,ymin,ymax,Ny,''tutor5'');
```

To impose a Crank-Nicholson filling method, you specify

```
_Hopscotch_Methods[2] = 0.5;
```

You can also define your own filling method or θ-method.

- The general form of the user-defined filling procedure is

```
proc (9) = procFillingMethod(a_ij,b_ij,c_ij,d_ij,e_ij,hx,hy,Nx,Ny);
  local deltaMinusMinus,deltaZeroMinus,deltaPlusMinus;
  local deltaMinusZero,deltaZeroZero,deltaPlusZero;
  local deltaMinusPlus,deltaZeroPlus,deltaPlusPlus;

  deltaMinusMinus =
  deltaZeroMinus =
  deltaPlusMinus =
  deltaMinusZero =
  deltaZeroZero =
  deltaPlusZero =
  deltaMinusPlus =
  deltaZeroPlus =
  deltaPlusPlus =

  retp(deltaMinusMinus,deltaZeroMinus,deltaPlusMinus,
       deltaMinusZero,deltaZeroZero,deltaPlusZero,
       deltaMinusPlus,deltaZeroPlus,deltaPlusPlus);
endp;
```

Remark 14 *The dimensions of the input and output matrices are $N_x \times N_y$. You must also use the element-by-element operators to define the output matrices.*

For example, the left-right method is implemented with the following procedure[1]

```
proc (9) = _LeftRight_FillingMethod(a_ij,b_ij,c_ij,d_ij,e_ij,hx,hy,Nx,Ny);
  local deltaMinusMinus,deltaZeroMinus,deltaPlusMinus;
  local deltaMinusZero,deltaZeroZero,deltaPlusZero;
  local deltaMinusPlus,deltaZeroPlus,deltaPlusPlus;

  a_ij = a_ij / (hx^2);
  b_ij = b_ij / (hx*hy);
  c_ij = c_ij / (hy^2);
  d_ij = d_ij / (2*hx);
  e_ij = e_ij / (2*hy);

  deltaMinusMinus = zeros(Nx,Ny);
  deltaZeroMinus = b_ij + c_ij - e_ij;
  deltaPlusMinus = - b_ij;
  deltaMinusZero = a_ij + b_ij - d_ij;
  deltaZeroZero = -2 * (a_ij + b_ij + c_ij);
  deltaPlusZero = a_ij + b_ij + d_ij;
  deltaMinusPlus = - b_ij;
  deltaZeroPlus = b_ij + c_ij + e_ij;
  deltaPlusPlus = zeros(Nx,Ny);

  retp(deltaMinusMinus,deltaZeroMinus,deltaPlusMinus,
       deltaMinusZero,deltaZeroZero,deltaPlusZero,
       deltaMinusPlus,deltaZeroPlus,deltaPlusPlus);
endp;
```

`_Hopscotch_Methods[1] = 1` is also equivalent to

```
_Hopscotch_Methods[1] = &_LeftRight_FillingMethod
```

[1] We have

$$\delta^m_{i,j,-1,-1} = \mathbf{0}$$

The corresponding matrix is then a $N_x \times N_y$.matrix of zeros.

- The general form of the user-defined θ-method procedure is

```
proc (2) = procThetaMethod(i,j,m);
  local theta_m,theta_mm;

  theta_m =
  theta_mm =

  retp(theta_m,theta_mm);
endp;
```

Remark 15 *theta_m and theta_mm correspond to the $N_x \times N_y$ θ_m and θ_{m+1} matrices. i and j are respectively a $N_x \times 1$ column vector and a $1 \times N_y$ row vector.*

For example, the odd-even Hopscotch method is implemented by the following procedure:

```
proc (2) = _OddEven_ThetaMethod(i,j,m);
  local theta_m,theta_mm;

  theta_m = isOdd(m+i+j);
  theta_mm = isOdd(m+i+j+1);

  retp(theta_m,theta_mm);
endp;
```

4.2.3 The _Hopscotch_PrintIters variable

If `_Hopscotch_PrintIters` is different to zero, `Hopscotch` displays the number of iterations performed.

4.2.4 The _Hopscotch_Retcode vector

If `Hopscotch` encounters difficulties to solve the PDE problem (that is the sparse system $\Psi_{m+1} u_{m+1} = \phi_{m+1}$), it displays an ERRORLOG message on the screen. In this case, you may investigate the `_Hopscotch_Retcode` vector to determine the problem. For example, the convergence is not achieved in the program below because of the number of iterations. If we modify the value taken by `_sparse_NumIters`, `Hopscotch` converges.

```
/*
** tutor7.prg
*/

new;
library PDE2D;

PDE2Dset;

#include tutor.inc;

__output = 0;

output file = tutor7.out reset;
```

```
call PDE2D(&aProc,&bProc,&cProc,&dProc,&eProc,&fProc,&gProc,0,
           &tminBound,&xminBound,&xmaxBound,&yminBound,&ymaxBound,
           0,0,0,0);

xmin = 0; xmax = 1; Nx = 21;
ymin = 0; ymax = 1; Ny = 11;
tmin = 0; tmax = 5; Nt = 51;

_Hopscotch_PrintIters = 1;
call Hopscotch(tmin,tmax,Nt,xmin,xmax,Nx,ymin,ymax,Ny,''tutor7'');

print _Hopscotch_Retcode[1:5];

_sparse_NumIters = 1000;
_Hopscotch_PrintIters = 3;
call Hopscotch(tmin,tmax,Nt,xmin,xmax,Nx,ymin,ymax,Ny,''tutor7'');
print _Hopscotch_Retcode[1:5];

output off;


Begin iterations
    1      2      3      4      5      6      7      8      9     10  ...
   11     12     13     14     15     16     17     18     19     20  ...
   21     22     23     24     25     26     27     28     29     30  ...
   31     32     33     34     35     36     37     38     39     40  ...
   41     42     43     44     45     46     47     48     49     50  ...
   51
Warning: The solution is not exact

      0.00000000
       7.0000000
       7.0000000
       7.0000000
       7.0000000

Begin iterations
    3      6      9     12     15     18     21     24     27     30  ...
   33     36     39     42     45     48     51

      0.00000000
       4.0000000
       4.0000000
       4.0000000
       4.0000000
```

4.2.5 The `_Hopscotch_SaveLastIter` variable

If you are just interested in the solution for $t = t^+$, you may use the `_Hopscotch_SaveLastIter` variable.

4.3 The `ExtractSolution` procedure

To extract solution from the dataset, we may use the Gauss commands `open`, `seekr` and `readr` or the PDE2D procedure `ExtractSolution`. You must remind that the time values and the space x and y values are respectively treated as a **column** vector, a **row** vector and a **column** vector. The only one exception is the command `U = ExtractSolution(NameFile,''y|yj'')`. In this case, the time values are stored in row format.

```
/*
```

```
** tutor8.prg
*/

new;
library PDE2D;

PDE2Dset;

#include tutor.inc;

__output = 0;

output file = tutor8.out reset;

call PDE2D(&aProc,&bProc,&cProc,&dProc,&eProc,&fProc,&gProc,0,
           &tminBound,&xminBound,&xmaxBound,&yminBound,&ymaxBound,
           0,0,0,0);

xmin = 0; xmax = 1; Nx = 13;
ymin = 0; ymax = 1; Ny = 11;
tmin = 0; tmax = 0.5; Nt = 6;

call Hopscotch(tmin,tmax,Nt,xmin,xmax,Nx,ymin,ymax,Ny,''tutor8'');

x = ExtractSolution(''tutor8'',''x'');
y = ExtractSolution(''tutor8'',''y'');
t = ExtractSolution(''tutor8'',''t'');

print ''-- x --'';
call printfm(x,1,''*.*lf'' ~7 ~5);
print ''-- y --'';
call printfm(y,1,''*.*lf'' ~5 ~2);
print; print ''-- t --'';
call printfm(t,1,''*.*lf'' ~5 ~2);

output off;


-- x --
0.00000
0.08333
0.16667
0.25000
0.33333
0.41667
0.50000
0.58333
0.66667
0.75000
0.83333
0.91667
1.00000
-- y --
 0.00 0.10 0.20 0.30 0.40 0.50 0.60 0.70 0.80 0.90 1.00
-- t --
 0.00
 0.10
 0.20
 0.30
 0.40
 0.50
```

Because the dataset is stored in the hard drive, you may investigate the solution without solving the problem at each time. In the program below, we extract the following solution:

- U_1 is a 6×11 matrix. It contains the values $u(t, x = 0.5, y)$. The rows correspond to the solution for t equal to 0, 0.1, 0.2, 0.3, 0.4 and 0.5. The columns correspond to the solution for y equal to 0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1 ;

- U_2 is a 13×6 matrix. It contains the values $u(t, x, y = 0.8)$. The rows correspond to the solution for x equal to 0, $\frac{1}{12}$, $\frac{1}{6}$, $\frac{1}{4}$, $\frac{1}{3}$, $\frac{5}{12}$, 0.5, $\frac{7}{12}$, $\frac{2}{3}$, $\frac{3}{4}$, $\frac{5}{6}$, $\frac{11}{12}$ and 1. The columns correspond to the solution for t equal to 0, 0.1, 0.2, 0.3, 0.4 and 0.5 ;
- U_3 is a 13×11 matrix. It contains the values $u(t = 0.5, x, y)$. The rows correspond to the solution for x equal to 0, $\frac{1}{12}$, $\frac{1}{6}$, $\frac{1}{4}$, $\frac{1}{3}$, $\frac{5}{12}$, 0.5, $\frac{7}{12}$, $\frac{2}{3}$, $\frac{3}{4}$, $\frac{5}{6}$, $\frac{11}{12}$ and 1. The columns correspond to the solution for y equal to 0, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9, 1.

```
/*
** tutor9.prg
*/

new;
library PDE2D;
PDE2Dset;

output file = tutor9.out reset;

U1 = ExtractSolution(''tutor8'',''x''|0.5);
U2 = ExtractSolution(''tutor8'',''y''|0.8);
U3 = ExtractSolution(''tutor8'',''t''|0.5);

call printfm(U1,1,''*.*lf'' ~7 ~3);
print ''============================================================'';
call printfm(U2,1,''*.*lf'' ~7 ~3);
print ''============================================================'';
call printfm(U3,1,''*.*lf'' ~7 ~3);

output off;


  0.500  0.530  0.570  0.620  0.680  0.750  0.830  0.920  1.020  1.130  1.250
  0.500  0.527  0.563  0.608  0.662  0.725  0.797  0.878  0.968  1.068  1.179
  0.500  0.524  0.557  0.596  0.645  0.701  0.765  0.837  0.918  1.006  1.114
  0.500  0.522  0.550  0.586  0.628  0.678  0.734  0.798  0.868  0.954  1.056
  0.500  0.519  0.545  0.576  0.613  0.656  0.706  0.760  0.827  0.901  1.003
  0.500  0.517  0.539  0.567  0.599  0.637  0.678  0.729  0.785  0.864  0.955
============================================================
  0.000  0.000  0.000  0.000  0.000  0.000
  0.142  0.136  0.131  0.125  0.120  0.115
  0.296  0.283  0.270  0.258  0.247  0.236
  0.460  0.439  0.418  0.399  0.380  0.365
  0.636  0.605  0.576  0.547  0.522  0.497
  0.822  0.782  0.742  0.705  0.669  0.640
  1.020  0.968  0.918  0.868  0.827  0.785
  1.229  1.165  1.101  1.044  0.986  0.946
  1.449  1.371  1.296  1.221  1.164  1.106
  1.680  1.588  1.496  1.417  1.339  1.293
  1.922  1.813  1.713  1.612  1.545  1.477
  2.176  2.053  1.931  1.839  1.747  1.675
  2.440  2.303  2.179  2.067  1.965  1.873
============================================================
  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000  0.000
  0.083  0.084  0.086  0.089  0.093  0.097  0.102  0.108  0.115  0.123  0.138
  0.167  0.169  0.174  0.180  0.188  0.198  0.209  0.222  0.236  0.256  0.285
  0.250  0.255  0.263  0.274  0.287  0.302  0.320  0.340  0.365  0.394  0.440
  0.333  0.342  0.354  0.369  0.388  0.410  0.435  0.465  0.497  0.543  0.603
  0.417  0.429  0.446  0.467  0.492  0.521  0.555  0.592  0.640  0.697  0.775
  0.500  0.517  0.539  0.567  0.599  0.637  0.678  0.729  0.785  0.864  0.955
  0.583  0.605  0.634  0.668  0.709  0.754  0.810  0.867  0.946  1.033  1.144
  0.667  0.695  0.730  0.773  0.821  0.880  0.942  1.023  1.106  1.222  1.341
  0.750  0.785  0.829  0.879  0.942  1.008  1.092  1.177  1.293  1.412  1.546
  0.833  0.876  0.929  0.994  1.065  1.152  1.243  1.357  1.477  1.616  1.760
  0.917  0.970  1.035  1.111  1.200  1.298  1.412  1.534  1.675  1.822  1.982
  1.000  1.067  1.146  1.237  1.340  1.455  1.582  1.722  1.873  2.037  2.213
```

We may use the `FindIndex` procedure to extract more specific solution.

```
/*
```

```
** tutor10.prg
*/

new;
library PDE2D;
PDE2Dset;

output file = tutor10.out reset;

x = ExtractSolution(''tutor8'',''x'');
y = ExtractSolution(''tutor8'',''y'');

indx = FindIndex(x,0.5);
indy = FindIndex(y',0.5);

u = submat(ExtractSolution(''tutor8'',''t''|0.5),indx,indy);
sol = SolutionProc(0.5,0.5,0.5);

print ftos(sol,''u(0.5,0.5,0.5) = %lf'',4,6);
print ftos(u,  ''Numerical sol. = %lf'',4,6);

output off;

proc solutionProc(t,x,y);
  retp( ((x^2).*y + x.*(y^2)).*exp(-t) + x );
endp;


u(0.5,0.5,0.5) = 0.651633
Numerical sol. = 0.636775
```

4.4 Elliptic problems

We consider the elliptic problem of Gourlay and McKee [1977], page 204:

$$\begin{aligned} a(x,y) &= 1 \\ b(x,y) &= -\frac{1}{2} \\ c(x,y) &= 1 \\ d(x,y) &= 0 \\ e(x,y) &= 0 \\ f(x,y) &= 0 \\ g(x,y) &= 0 \end{aligned}$$

with the boundary conditions

$$\begin{aligned} u(0,y) &= 0 \\ u(1,y) &= y + y^2 \\ u(x,0) &= 0 \\ u(x,1) &= x + x^2 \end{aligned}$$

The solution of this problem is

$$u(x,y) = x^2 y + x y^2$$

For the initial guess solution, we use random numbers.

```
/*
** elliptic.inc
```

```
*/

proc aProc(t,x,y);
  retp( ones(rows(x),cols(y)) );
endp;

proc bProc(t,x,y);
  retp( -0.5*ones(rows(x),cols(y)) );
endp;

proc cProc(t,x,y);
  retp( ones(rows(x),cols(y)) );
endp;

proc dProc(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

proc eProc(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

proc fProc(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

proc gProc(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;


proc xminBound(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

proc xmaxBound(t,x,y);
  retp( ones(rows(x),1) .* (y + y^2) );
endp;

proc yminBound(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

proc ymaxBound(t,x,y);
  retp( ones(rows(x),1) .* (x + x^2) );
endp;


proc InitialGuessSolution(t,x,y);
  retp( rndn(rows(x),cols(y)) );
endp;


proc solutionProc(t,x,y);
  retp( (x^2).*y + x.*(y^2) );
endp;
```

The program below shows the convergence of the Hopscotch method to the elliptic solution.

```
/*
** tutor11.prg
*/

new;
library PDE2D,pgraph;
PDE2Dset;

#include elliptic.inc;

rndseed 123;
```

```
call PDE2D(&aProc,&bProc,&cProc,&dProc,&eProc,&fProc,&gProc,0,
           &InitialGuessSolution,
           &xminBound,&xmaxBound,&yminBound,&ymaxBound,
           0,0,0,0);

xmin = 0; xmax = 1; Nx = 21;
ymin = 0; ymax = 1; Ny = 21;
tmin = 0; tmax = 0.425; Nt = 101;

_Hopscotch_Methods = {2,3}; /* center filling method
                                    &
                               line Hopscotch method */

call Hopscotch(tmin,tmax,Nt,xmin,xmax,Nx,ymin,ymax,Ny,''tutor11'');

x = ExtractSolution(''tutor11'',''x'');
y = ExtractSolution(''tutor11'',''y'');
t = ExtractSolution(''tutor11'',''t'');

sol = solutionProc(tmax,x,y);


graphset;

  begwind;
  window(4,4,0);

  _pdate = ''''; _pnum = 0; _paxes = 0; _pframe = 0; _psurf = {0,0};
  _ptitlht = 0.45;

  i = 1;
  do until i > 14;

    U = ExtractSolution(''tutor11'',''t''|t[2*i]);

    setwind(i);
      str = ftos(2*i - 1,''iter no %lf'',1,0);
      title(str);
      surface(x',y',U');

    i = i + 1;
  endo;

  U = ExtractSolution(''tutor11'',''t''|t[Nt]);

  setwind(15);
    str = ftos(Nt,''iter no %lf'',1,0);
    title(str);
    surface(x',y',U');

  setwind(16);
    title(''Exact solution'')
    surface(x',y',sol');

  graphprt(''-c=1 -cf=tutor11.eps'');

  endwind;
```

The next program is an illustration of the use of the `_Hopscotch_Elliptic` variable. In this case, the convergence is achieved after 79 iterations.

```
/*
** tutor12.prg
*/

new;
```

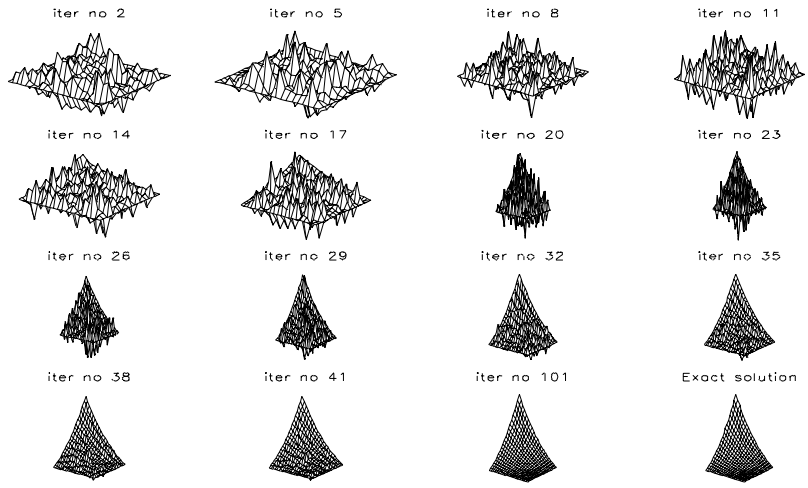

Figure 4-1

```
library PDE2D,pgraph;
PDE2Dset;

#include elliptic.inc;

rndseed 123;

output file = tutor12.out reset;

call PDE2D(&aProc,&bProc,&cProc,&dProc,&eProc,&fProc,&gProc,0,
           &InitialGuessSolution,
           &xminBound,&xmaxBound,&yminBound,&ymaxBound,
           0,0,0,0);

xmin = 0; xmax = 1; Nx = 21;
ymin = 0; ymax = 1; Ny = 21;
tmin = 0; tmax = 0.425; Nt = 101;

_Hopscotch_Methods = {1,2}; /* left-right filling method
                                    &
                               ordered odd-even Hopscotch method */

_Hopscotch_Elliptic = 1;
_Hopscotch_SaveLastIter = 1;
_fcmptol = 1e-4;

call Hopscotch(tmin,tmax,Nt,xmin,xmax,Nx,ymin,ymax,Ny,''tutor12'');

x = ExtractSolution(''tutor12'',''x'');
y = ExtractSolution(''tutor12'',''y'');
t = ExtractSolution(''tutor12'',''t'');

print; print ''t vector = '' t;

U = ExtractSolution(''tutor12'',''t''|t);
sol = solutionProc(t,x,y);

output off;

graphset;
  _pdate = '''';
  title(''Relative numerical error (%)'');
  xlabel(''x'');
  ylabel(''y'');
  ztics(0,1.5,0.5,5);
```

```
  graphprt(''-c=1 -cf=tutor12.eps'');
  surface(x',y',100*missrv((U-sol)'./miss(sol,0),0));


=========================================================
Bound              Dirichlet            Neumann

xmin               *********
xmax               *********
ymin               *********
ymax               *********
=========================================================

Filling Hopscotch method: left-right
Theta Hopscotch method: ordered odd-even

Mesh spacing in time    :  k =   0.00425000
Mesh spacing in space x : hx =   0.05000000
Mesh spacing in space y : hy =   0.05000000

Mesh ratio r(x,x):     1.700000
           r(x,y):     1.700000
           r(y,y):     1.700000

Elliptic Problem - Algorithm stopped at the 79th iteration
Hopscotch completed

t vector =       0.33150000
```

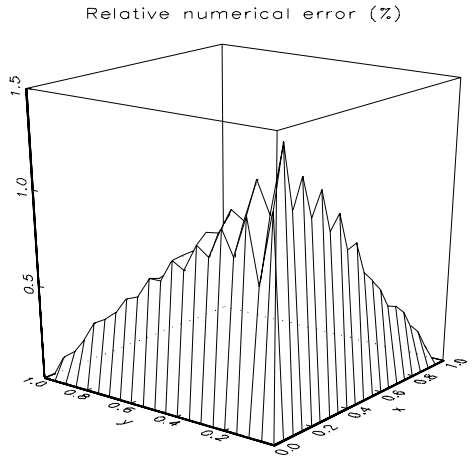

Figure 4-2

4.5 Some examples

4.5.1 1st example of Gourlay and McKee [1977]

```
/*
**> ex1.prg
**
** Gourlay, A.R. and S. McKee [1977], The construction of hopscotch methods
```

```
** for parabolic and elliptic equations in two space dimensions with a mixed
** derivative, Journal of Computational and Applied Mathematics, 3, 201-205
**
** page 203, Example 1
**
** a(t,x,y) = a = 0.1
** b(t,x,y) = b = 0.05
** c(t,x,y) = c = 0.15
** d(t,x,y) = 0
** e(t,x,y) = 0
** f(t,x,y) = 0
** g(t,x,y) = 0
**
** u(0,x,y) = sin(x+y)
** u(t,0,y) = exp(-(a+2b+c)*t)*sin(y)
** u(t,1,y) = exp(-(a+2b+c)*t)*sin(1+y)
** u(t,x,0) = exp(-(a+2b+c)*t)*sin(x)
** u(t,x,1) = exp(-(a+2b+c)*t)*sin(1+x)
**
** h(t,x,y,u) = u
*/

new;
library PDE2D,pgraph;
PDE2Dset;

a = 0.1; b = 0.05; c = 0.15;

proc aProc(t,x,y);
  retp( a*ones(rows(x),cols(y)) );
endp;

proc bProc(t,x,y);
  retp( b*ones(rows(x),cols(y)) );
endp;

proc cProc(t,x,y);
  retp( c*ones(rows(x),cols(y)) );
endp;

proc dProc(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

proc eProc(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

proc fProc(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

proc gProc(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

proc tminBound(t,x,y);
  retp( sin(x+y) );
endp;

proc xminBound(t,x,y);
  retp( exp(-(a+2*b+c)*t)*sin(y) );
endp;

proc xmaxBound(t,x,y);
  retp( exp(-(a+2*b+c)*t)*sin(1+y) );
endp;

proc yminBound(t,x,y);
  retp( exp(-(a+2*b+c)*t)*sin(x) );
```

```
endp;

proc ymaxBound(t,x,y);
  retp( exp(-(a+2*b+c)*t)*sin(1+x) );
endp;


call PDE2D(&aProc,&bProc,&cProc,&dProc,&eProc,&fProc,&gProc,0,
               &tminBound,&xminBound,&xmaxBound,&yminBound,&ymaxBound,
               0,0,0,0);

xmin = 0; xmax = 1; Nx = 11;
ymin = 0; ymax = 1; Ny = 21;
tmin = 0; tmax = 5; Nt = 51;

call Hopscotch(tmin,tmax,Nt,xmin,xmax,Nx,ymin,ymax,Ny,''ex1'');

x = ExtractSolution(''ex1'',''x'');
y = ExtractSolution(''ex1'',''y'');
t = ExtractSolution(''ex1'',''t'');
U = ExtractSolution(''ex1'',''t''|tmax);

/* The exact solution is
**
**          u(t,x,y) = exp(-(a+2b+c)*t)*sin(x+y)
**
*/

proc solutionProc(t,x,y);
  retp( exp(-(a+2*b+c)*t)*sin(x+y) );
endp;

sol = solutionProc(tmax,x,y);


graphset;

  begwind;
  makewind(9,1,0,6.855-1,1);
  makewind(9/2,5.855,0,0,1);
  makewind(9/2,5.855,9/2,0,1);

  setwind(1);
    _pdate = '''';  _ptitlht = 1.25; _pnum = 0; _paxes = 0;
    title(''Gourlay and McKee @[1977@] - Example 1 page 203'');
    draw;

  graphset;
    _pdate = '''';  _pnum = 2; _ptitlht = 0.25; _paxht = 0.25; _pnumht = 0.20;
    xlabel(''x'');
    ylabel(''y'');
    zlabel(''u(5,x,y)'');

  setwind(2);
    title(''Numerical solution'')
    surface(x',y',U');

  setwind(3);
    title(''Exact solution'')
    surface(x',y',sol');
    graphprt(''-c=1 -cf=ex1.eps'');

  endwind;
```

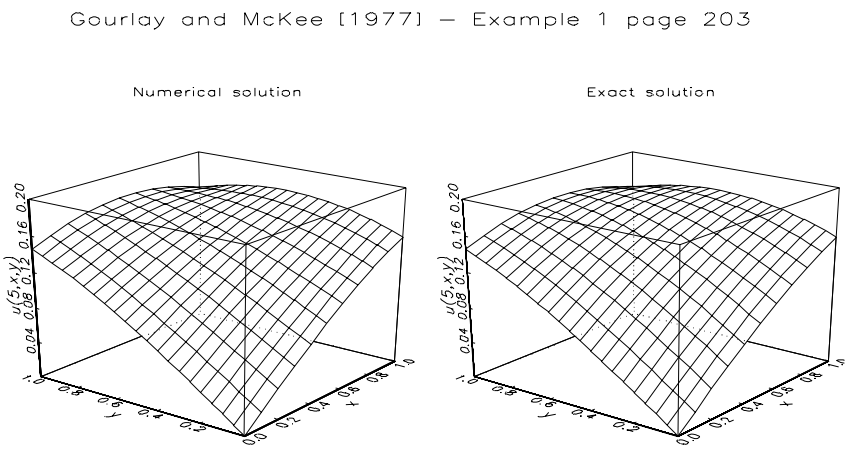

Figure 4-3

4.5.2 2nd example of Gourlay and McKee [1977]

```
/*
**> ex2.prg
**
** Gourlay, A.R. and S. McKee [1977], The construction of hopscotch methods
** for parabolic and elliptic equations in two space dimensions with a mixed
** derivative, Journal of Computational and Applied Mathematics, 3, 201-205
**
** page 203, example 2
**
** a(t,x,y) = 0.5*x^2 + y^2
** b(t,x,y) = -0.5 * (x^2 + y^2)
** c(t,x,y) = x^2 + 0.5*y^2
** d(t,x,y) = 0
** e(t,x,y) = 0
** f(t,x,y) = 0
** g(t,x,y) = 0
**
** u(0,x,y) = x^2*y + x*y^2
** u(t,0,y) = 0
** u(t,1,y) = (y + y^2) * exp(-t)
** u(t,x,0) = 0
** u(t,x,1) = (x + x^2) * exp(-t)
**
** h(t,x,y,u) = u
*/

new;
library PDE2D,pgraph;
PDE2Dset;


proc aProc(t,x,y);
  retp(0.5*x^2 + y^2);
endp;

proc bProc(t,x,y);
  retp( -0.5 * (x^2 + y^2) );
endp;

proc cProc(t,x,y);
  retp(x^2 + 0.5*y^2);
endp;
```

```
proc dProc(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

proc eProc(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

proc fProc(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

proc gProc(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

proc tminBound(t,x,y);
  retp( (x^2).*y + x.*(y^2) );
endp;

proc xminBound(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

proc xmaxBound(t,x,y);
  retp( (y + y^2) * exp(-t) );
endp;

proc yminBound(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

proc ymaxBound(t,x,y);
  retp( (x + x^2) * exp(-t) );
endp;


call PDE2D(&aProc,&bProc,&cProc,&dProc,&eProc,&fProc,&gProc,0,
               &tminBound,&xminBound,&xmaxBound,&yminBound,&ymaxBound,
               0,0,0,0);

xmin = 0; xmax = 1; Nx = 21;
ymin = 0; ymax = 1; Ny = 21;
tmin = 0; tmax = 0.25; Nt = 51;

call Hopscotch(tmin,tmax,Nt,xmin,xmax,Nx,ymin,ymax,Ny,''ex2'');

x = ExtractSolution(''ex2'',''x'');
y = ExtractSolution(''ex2'',''y'');
t = ExtractSolution(''ex2'',''t'');
U = ExtractSolution(''ex2'',''t''|tmax);

/* The exact solution is
**
**          u(t,x,y) = (x^2*y + x*y^2) * exp(-t)
**
*/

proc solutionProc(t,x,y);
  retp( ( (x^2).*y + x.*(y^2) ) * exp(-t) );
endp;

sol = solutionProc(tmax,x,y);

graphset;

  begwind;
  makewind(9,1,0,6.855-1,1);
  makewind(9/2,5.855,0,0,1);
```

```
makewind(9/2,5.855,9/2,0,1);

setwind(1);
  _pdate = ''''; _ptitlht = 1.25; _pnum = 0; _paxes = 0;
  title(''Gourlay and McKee @[1977@] - Example 2 page 203'');
  draw;

graphset;
  _pdate = ''''; _pnum = 2; _ptitlht = 0.25; _paxht = 0.20; _pnumht = 0.15;
  xlabel(''x'');
  ylabel(''y'');

setwind(2);
  title(''Numerical solution'')
  surface(x',y',U');

setwind(3);
  title(''Exact solution'')
  surface(x',y',sol');
  graphprt(''-c=1 -cf=ex2.eps'');

endwind;
```

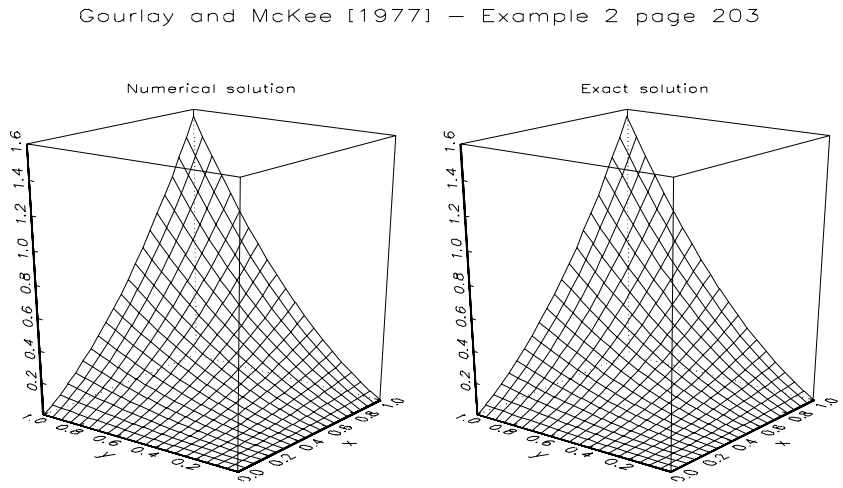

Figure 4-4

4.5.3 4th example of Gourlay and McKee [1977]

Remark 16 *I have done this example, because I think that it illustrates the difference between the ordered odd-even and the line methods.*

```
/*
**> ex3.prg
**
** Gourlay, A.R. and S. McKee [1977], The construction of hopscotch methods
** for parabolic and elliptic equations in two space dimensions with a mixed
** derivative, Journal of Computational and Applied Mathematics, 3, 201-205
**
** page 205, example 4
**
** a(t,x,y) = (x - 0.5)^2
** b(t,x,y) = 0.5 * (x - 0.5) * (y - 0.5)
```

```
** c(t,x,y) = (y - 0.5)^2
** d(t,x,y) = 0
** e(t,x,y) = 0
** f(t,x,y) = 0
** g(t,x,y) = -(5*x*y*(x+y) + (x+y) - (6*x*y + x^2 + y^2))*exp(-t)
**
** u(0,x,y) = x*y*(x+y)
** u(t,0,y) = 0
** u(t,1,y) = (y + y^2) * exp(-t)
** u(t,x,0) = 0
** u(t,x,1) = (x + x^2) * exp(-t)
**
** h(t,x,y,u) = u
*/

new;
library PDE2D,pgraph;
PDE2Dset;


proc aProc(t,x,y);
  retp( ((x - 0.5)^2) .* ones(1,cols(y)) );
endp;

proc bProc(t,x,y);
  retp(0.5*(x - 0.5).*(y - 0.5));
endp;

proc cProc(t,x,y);
  retp( ((y - 0.5)^2) .* ones(rows(x),1) );
endp;

proc dProc(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

proc eProc(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

proc fProc(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

proc gProc(t,x,y);
  retp( -(5*x.*y.*(x+y) + (x+y) - (6*x.*y + x^2 + y^2)).*exp(-t) );
endp;

proc tminBound(t,x,y);
  retp( (x^2).*y + x.*(y^2) );
endp;

proc xminBound(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

proc xmaxBound(t,x,y);
  retp( (y + y^2) .* exp(-t) );
endp;

proc yminBound(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

proc ymaxBound(t,x,y);
  retp( (x + x^2) .* exp(-t) );
endp;

call PDE2D(&aProc,&bProc,&cProc,&dProc,&eProc,&fProc,&gProc,0,
               &tminBound,&xminBound,&xmaxBound,&yminBound,&ymaxBound,
```

```
              0,0,0,0);

xmin = 0; xmax = 1; Nx = 21;
ymin = 0; ymax = 1; Ny = 21;
tmin = 0; tmax = 0.25; Nt = 101;

_Hopscotch_PrintIters = 1;

_Hopscotch_Methods = 1|2;
call Hopscotch(tmin,tmax,Nt,xmin,xmax,Nx,ymin,ymax,Ny,''ex3'');

x = ExtractSolution(''ex3'',''x'');
y = ExtractSolution(''ex3'',''y'');
t = ExtractSolution(''ex3'',''t'');

indx = FindIndex(x,0.25|0.5|0.75);
indy = FindIndex(y',0.5);

U = ExtractSolution(''ex3'',''y''|y[indy]);
U1 = U[indx,.];

_Hopscotch_Methods = 2|3;
call Hopscotch(tmin,tmax,Nt,xmin,xmax,Nx,ymin,ymax,Ny,''ex3'');

x = ExtractSolution(''ex3'',''x'');
y = ExtractSolution(''ex3'',''y'');
t = ExtractSolution(''ex3'',''t'');

indx = FindIndex(x,0.25|0.5|0.75);
indy = FindIndex(y',0.5);

U = ExtractSolution(''ex3'',''y''|y[indy]);
U2 = U[indx,.];


/* The exact solution is
**
**          u(t,x,y) = (x^2*y + x*y^2) * exp(-t)
**
*/

proc solutionProc(t,x,y);
  retp( ( (x^2).*y + x.*(y^2) ) * exp(-t) );
endp;

sol = solutionProc(t',0.25|0.5|0.75,0.5);

graphset;
  begwind;
  window(1,2,0);
  _pdate = ''''; _pnum = 2; _pnumht = 0.25; _paxht = 0.25; _ptitlht = 0.25;
  _plctrl = -3; _pstype = 8|9|10; _pysci = 1; _psymsiz = 7;
  xtics(0,0.25,0.05,5);
  xlabel(''t'');
  ylabel(''Numerical error for y = 0.5'');
  _plegstr = ''x = 0.25\000x = 0.50\000x = 0.75'';

  setwind(1);
    title(''ORDERED ODD-EVEN Hopscotch'');
    _plegctl = {2 7 7.75 0.75};
    xy(t,sol'-U1');
  setwind(2);
    _plegctl = {2 7 -1.25 0.75};
    title(''LINE Hopscotch'');
    xy(t,sol'-U2');
    graphprt(''-c=1 -cf=ex3.eps'');
  endwind;
```

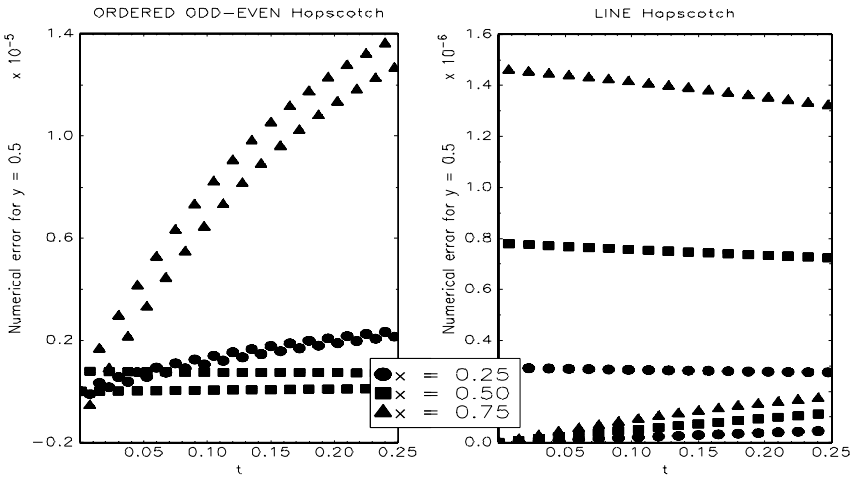

Figure 4-5

4.5.4 Stochastic volatility option model

The two programs below have been used to generate the following table of KURPIEL and RONCALLI [1998].

$\rho = -0.5$					
S_0	C_{EU}	$C^{\mathrm{He}}_{\mathrm{EU,SV}}$	$C^{\mathrm{H}}_{\mathrm{EU,SV}}$	$C^{\mathrm{BAW}}_{\mathrm{AM}}$	$C^{\mathrm{H}}_{\mathrm{AM,SV}}$
80	0.0291	0.0169	0.0168	0.0322	0.0170
85	0.1545	0.1170	0.1166	0.1624	0.1185
90	0.5700	0.5040	0.5034	0.5896	0.5133
95	1.5689	1.5030	1.5026	1.6151	1.5401
100	3.4211	3.3961	3.3958	3.5249	3.5038
105	6.2204	6.2496	6.2492	6.4448	6.5016
110	9.8470	9.9094	9.9087	10.3146	10.4123
115	14.0596	14.1251	14.1242	15	15.0156
120	18.6180	18.6682	18.6675	20	20

4.5.4.1 European options

```
/*
**> ex4.prg
**
** Kurpiel, A. and T. Roncalli [1998], Hopscotch methods for two-state
** financial models, FERC working paper, City University Business School
*/

new;
library option,PDE2D;
PDE2Dset;

K = 100;
S0 = seqa(80,5,9);
b = -0.04;
r =  0.08;
V0 = 0.20^2;
tau = 0.25;

rho = -0.5;
Kappa = 0.9;
theta = 0.20^2;
```

```
lambda = 0.0;
sigmaV = 0.10;
kappaStar = kappa + lambda;
thetaStar = kappa*theta/(kappa+lambda);

proc aProc(t,x,y);
  retp( 0.5 * y .* (x^2) );
endp;

proc bProc(t,x,y);
  retp( 0.5* rho * sigmaV * y .* x );
endp;

proc cProc(t,x,y);
  retp( 0.5 * (sigmaV^2).*y .* ones(rows(x),1) );
endp;

proc dProc(t,x,y);
  retp( b.*x.*ones(1,cols(y)) );
endp;

proc eProc(t,x,y);
  retp( ( kappaStar*(thetaStar-y) ) .* ones(rows(x),1) );
endp;

proc fProc(t,x,y);
  retp( r .* ones(rows(x),cols(y)) );
endp;

proc gProc(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

proc tminBound(t,x,y);
  x = ones(1,cols(y)) .* x;
  retp( (x .> K) .* ( x - K) );
endp;

proc xminBound(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

proc DxmaxBound(t,x,y);
  retp( ones(rows(x),cols(y)) );
endp;

proc DyminBound(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

proc DymaxBound(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

call PDE2D(&aProc,&bProc,&cProc,&dProc,&eProc,&fProc,&gProc,0,
                  &tminBound,&xminBound,0,0,0,
                  0,&DxmaxBound,0,0);

xmin = 50; xmax = 150; Nx = 201;
ymin = 0.002; ymax = 0.122; Ny = 61;
tmin = 0; tmax = tau; Nt = floor(1825*tau);

_Hopscotch_SaveLastIteration = 1;
_Hopscotch_PrintIters = 1;

call Hopscotch(tmin,tmax,Nt,xmin,xmax,Nx,ymin,ymax,Ny,''tmp'');

x = ExtractSolution(''tmp'',''x'');
y = ExtractSolution(''tmp'',''y'');
tmax = ExtractSolution(''tmp'',''t'');
```

```
U = ExtractSolution(''tmp'',''t''|tmax);

_fcmptol = 0.1;
indxX = FindIndex(x,S0); indxY = FindIndex(y',V0);
S0 = x[indxX]; V0 = y[indxY];

Ceu1 = EuropeanBS(S0,K,sqrt(V0),tau,b,r);
Ceu2 = zeros(rows(S0),1);
i = 1;
do until i > rows(S0);
  {Ceu2[i],retcode} = EuropeanHeston(S0[i],K,V0,tau,b,r,
                                     kappa,theta,sigmaV,rho,lambda);
  i = i + 1;
endo;

output file = ex4.out reset;

print S0~Ceu1~Ceu2~U[indxX,indxY];

output off;


       80.000000      0.029090207      0.016931435      0.016758608
       85.000000       0.15446696       0.11698944       0.11664725
       90.000000       0.56998560       0.50399368       0.50342106
       95.000000        1.5689110        1.5030913        1.5025541
       100.00000        3.4211088        3.3961055        3.3957527
       105.00000        6.2204481        6.2495834        6.2491900
       110.00000        9.8469572        9.9093726        9.9087376
       115.00000        14.059550        14.125064        14.124180
       120.00000        18.618023        18.668220        18.667531
```

4.5.4.2 American options

```
/*
**> ex5.prg
**
** Kurpiel, A. and T. Roncalli [1998], Hopscotch methods for two-state
** financial models, FERC working paper, City University Business School
*/

new;
library option,PDE2D;
PDE2Dset;

K = 100;
S0 = seqa(80,5,9);
b = -0.04;
r =  0.08;
V0 = 0.20^2;
tau = 0.25;

rho = -0.5;
Kappa = 0.9;
theta = 0.20^2;
lambda = 0.0;
sigmaV = 0.10;
kappaStar = kappa + lambda;
thetaStar = kappa*theta/(kappa+lambda);

proc aProc(t,x,y);
  retp( 0.5 * y .* (x^2) );
endp;

proc bProc(t,x,y);
  retp( 0.5* rho * sigmaV * y .* x );
endp;

proc cProc(t,x,y);
  retp( 0.5 * (sigmaV^2).*y .* ones(rows(x),1) );
```

```
endp;

proc dProc(t,x,y);
  retp( b.*x.*ones(1,cols(y)) );
endp;

proc eProc(t,x,y);
  retp( ( kappaStar*(thetaStar-y) ) .* ones(rows(x),1) );
endp;

proc fProc(t,x,y);
  retp( r .* ones(rows(x),cols(y)) );
endp;

proc gProc(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

@<------------------ h(t,x,y,U) ------------------>@

proc hProc(t,x,y,U);
  local PayOff,cnd;
  PayOff = ( x .> K ) .* ( x - K ) .* ones(1,cols(y));
  cnd = PayOff .> U;
  retp( cnd.*PayOff + (1-cnd).*U );
endp;

@<------------------------------------------------>@

proc tminBound(t,x,y);
  x = ones(1,cols(y)) .* x;
  retp( (x .> K) .* ( x - K) );
endp;

proc xminBound(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

proc DxmaxBound(t,x,y);
  retp( ones(rows(x),cols(y)) );
endp;

proc DyminBound(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

proc DymaxBound(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

call PDE2D(&aProc,&bProc,&cProc,&dProc,&eProc,&fProc,&gProc,&hProc,
                  &tminBound,&xminBound,0,0,0,
                  0,&DxmaxBound,0,0);

xmin = 50; xmax = 150; Nx = 201;
ymin = 0.002; ymax = 0.122; Ny = 61;
tmin = 0; tmax = tau; Nt = floor(1825*tau);

_Hopscotch_SaveLastIteration = 1;
_Hopscotch_PrintIters = 1;
call Hopscotch(tmin,tmax,Nt,xmin,xmax,Nx,ymin,ymax,Ny,''tmp'');

x = ExtractSolution(''tmp'',''x'');
y = ExtractSolution(''tmp'',''y'');
tmax = ExtractSolution(''tmp'',''t'');
U = ExtractSolution(''tmp'',''t''|tmax);

_fcmptol = 0.1;
indxX = FindIndex(x,S0); indxY = FindIndex(y',V0);
S0 = x[indxX]; V0 = y[indxY];
```

```
Cam = AmericanBS(S0,K,sqrt(V0),tau,b,r);

output file = ex5.out reset;

print S0~Cam~U[indxX,indxY];

output off;
```

```
       80.000000      0.032151197      0.016965155
       85.000000       0.16244037       0.11845361
       90.000000       0.58964881       0.51329422
       95.000000        1.6150921        1.5400671
       100.00000        3.5249212        3.5037915
       105.00000        6.4447677        6.5015876
       110.00000        10.314602        10.412318
       115.00000        15.000000        15.015637
       120.00000        20.000000        20.000000
```

4.5.5 Greeks computing

The following program numerically solves the Black-Scholes model. Then, we use the `Derive` procedure to compute the Δ, Γ and Θ coefficients of the call options. We remark that the differences between these numerical values and these obtained by exact formulas are very small.

```
/*
**> ex6.prg
**
*/

new;
library PDE2D,option,pgraph;
PDE2Dset;

K = 100;
sigma = 0.20;
tau = 0.25;
r = 0.08;
b = 0.08;

proc aProc(t,x,y);
  retp( 0.5*(sigma^2) .* (x^2) .* ones(1,cols(y)) );
endp;

proc bProc(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

proc cProc(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

proc dProc(t,x,y);
  retp( b .* x .* ones(1,cols(y)) );
endp;

proc eProc(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

proc fProc(t,x,y);
  retp( r .* ones(rows(x),cols(y)) );
endp;
```

```
proc gProc(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

proc tminBound(t,x,y);
  local PayOff;
  PayOff = x - K;
  PayOff = PayOff .* (PayOff .> 0);
  retp( PayOff .* ones(1,cols(y)) );
endp;

proc xminBound(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

proc DxmaxBound(t,x,y);
  retp( ones(rows(x),cols(y)) );
endp;

call PDE2D(&aProc,&bProc,&cProc,&dProc,&eProc,&fProc,&gProc,0,
               &tminBound,&xminBound,0,0,0,
                          0,&DxmaxBound,0,0);

xmin = 50; xmax = 150; Nx = 201;
ymin = 0.10; ymax = 0.30; Ny = 5;
tmin = 0; tmax = tau; Nt = floor(1825*tau);

_Hopscotch_PrintIters = 10;
call Hopscotch(tmin,tmax,Nt,xmin,xmax,Nx,ymin,ymax,Ny,''ex6'');

dt = vread(_Hopscotch_MeshRatios,''k'');
dS = vread(_Hopscotch_MeshRatios,''hx'');

indxT = seqa(10,15,30);
indxS0 = seqa(1,10,21);

graphset;

  begwind;
  window(2,2,0);

  _pdate = ''''; _pnum = 2; _pnumht = 0.25; _ptitlht = 0.25; _paxht = 0.25;
  fonts(''simplex simgrma'');

  setwind(1);

    S0 = ExtractSolution(''ex6'',''x'');
    U = ExtractSolution(''ex6'',''t''|tau);
    delta = Derive(U,dS,0);

    title(''Delta'');
    xlabel(''S0'');
    ytics(0,1,0.25,0);
    xy(S0[indxS0],submat(delta~EuropeanBS_Delta(S0,K,sigma,tau,b,r),indxS0,0));

  setwind(2);

    gamma_ = Derive(Derive(U,dS,-1),dS,1);

    title(''Gamma'');
    ytics(0,0.05,0.01,0);
    xy(S0[indxS0],submat(gamma_~EuropeanBS_Gamma(S0,K,sigma,tau,b,r),indxS0,0));

  setwind(3);

    t = ExtractSolution(''ex6'',''t'');
    y = ExtractSolution(''ex6'',''y'');
    U = ExtractSolution(''ex6'',''y''|y[3]);
    theta = Derive(U',dt,0);
```

```
    title(''Theta (Hopscotch)'');
    xlabel(''\202t\201'');
    ylabel(''S0'');
    xtics(0,0.20,0.10,0);
    ytics(80,120,10,0);
    ztics(0,50,10,0);
    surface(submat(t',0,indxT),submat(S0,indxS0,0),
            submat(theta',indxS0,indxT));

  setwind(4);

    title(''Theta'');
    surface(submat(t',0,indxT),submat(S0,indxS0,0),
            submat(EuropeanBS_Theta(S0,K,sigma,t',b,r),indxS0,indxT));

    graphprt(''-c=1 -cf=ex6.eps'');

  endwind;
```

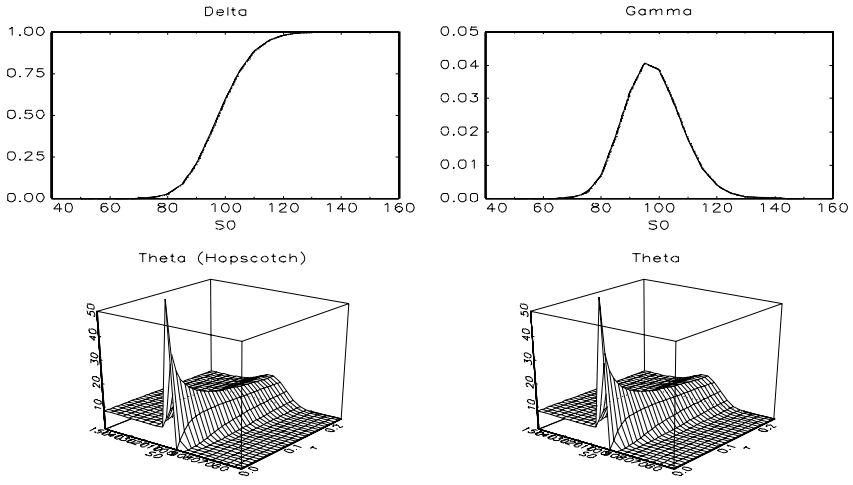

Figure 4-6

4.5.6 Laplace equation

```
/*
**> ex7.prg
**
** Laplace Equation
*/

new;
library PDE2D,pgraph;
PDE2Dset;

proc aProc(t,x,y);
  retp( 2*ones(rows(x),cols(y)) );
endp;

proc bProc(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

proc cProc(t,x,y);
```

```
  retp( 2*ones(rows(x),cols(y)) );
endp;

proc dProc(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

proc eProc(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

proc fProc(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

proc gProc(t,x,y);
  retp( zeros(rows(x),cols(y)) );
endp;

proc InitialGuessSolution(t,x,y);
  retp( rndu(rows(x),cols(y)) );
endp;

proc xminBound(t,x,y);
  retp( ones(rows(x),cols(y)) .* (1 - y^2) );
endp;

proc xmaxBound(t,x,y);
  retp( ones(rows(x),cols(y)) .* (1 - y^2) );
endp;

proc yminBound(t,x,y);
  retp( ones(rows(x),cols(y)) .* (x^2 - 1) );
endp;

proc ymaxBound(t,x,y);
  retp( ones(rows(x),cols(y)) .* (x^2 - 1) );
endp;

call PDE2D(&aProc,&bProc,&cProc,&dProc,&eProc,&fProc,&gProc,0,
               &InitialGuessSolution,&xminBound,&xmaxBound,&yminBound,&ymaxBound,
               0,0,0,0);

xmin = -1; xmax = 1; Nx = 21;
ymin = -1; ymax = 1; Ny = 21;
tmin = 0; tmax = 0.25; Nt = 101;

_Hopscotch_PrintIters = 2;
_Hopscotch_Elliptic = 1;
_Hopscotch_SaveLastIter = 1;
_fcmptol = 1e-5;

_Hopscotch_Methods[1] = 2;  /* Center Filling Method */

_Hopscotch_Methods[2] = 0.5; /* Crank-Nicholson Theta Method */

call Hopscotch(tmin,tmax,Nt,xmin,xmax,Nx,ymin,ymax,Ny,''tmp'');
x = ExtractSolution(''tmp'',''x'');
y = ExtractSolution(''tmp'',''y'');
t = ExtractSolution(''tmp'',''t'');
U1 = ExtractSolution(''tmp'',''t''|t);

_Hopscotch_Methods[2] = 1; /* Implicit Theta Method */

call Hopscotch(tmin,tmax,Nt,xmin,xmax,Nx,ymin,ymax,Ny,''tmp'');
t = ExtractSolution(''tmp'',''t'');
U2 = ExtractSolution(''tmp'',''t''|t);

_Hopscotch_Methods[2] = 2; /* Ordered Odd-Even Theta Method */
```

```
call Hopscotch(tmin,tmax,Nt,xmin,xmax,Nx,ymin,ymax,Ny,''tmp'');
t = ExtractSolution(''tmp'',''t'');
U3 = ExtractSolution(''tmp'',''t''|t);

_Hopscotch_Methods[2] = 3; /* Line Theta Method */

call Hopscotch(tmin,tmax,Nt,xmin,xmax,Nx,ymin,ymax,Ny,''tmp'');
t = ExtractSolution(''tmp'',''t'');
U4 = ExtractSolution(''tmp'',''t''|t);

proc SolutionProc(x,y);
  retp(x^2-y^2);
endp;

sol = SolutionProc(x,y);


graphset;
  begwind;
  makewind(9,6.855,0,0,1);
  makewind(2.5,2.5,0,0,1);
  makewind(2.5,2.5,0,6.855-2.5,1);
  makewind(2.5,2.5,9-2.5,0,1);
  makewind(2.5,2.5,9-2.5,6.855-2.5,1);

  _pnum = 2;

  setwind(1);

    title(''Laplace Equation'');
    xtics(-1,1,0.5,0);
    ytics(-1,1,0.5,0);
    surface(x',y',U1');

  graphset;
  _ptitlht = 0.30; _pnumht = 0.70; _paxes = 0;

  setwind(2);
   title(''Crank-Nicholson'');
    contour(x',y',(U1-sol)');

  setwind(3);
    title(''Implicit'');
    contour(x',y',(U2-sol)');

  setwind(4);
    title(''Ordered Odd-Even'');
    contour(x',y',(U3-sol)');

  setwind(5);
    title(''Line'');
    contour(x',y',(U4-sol)');

  graphprt(''-c=1 -cf=ex7.eps'');

  endwind;
```

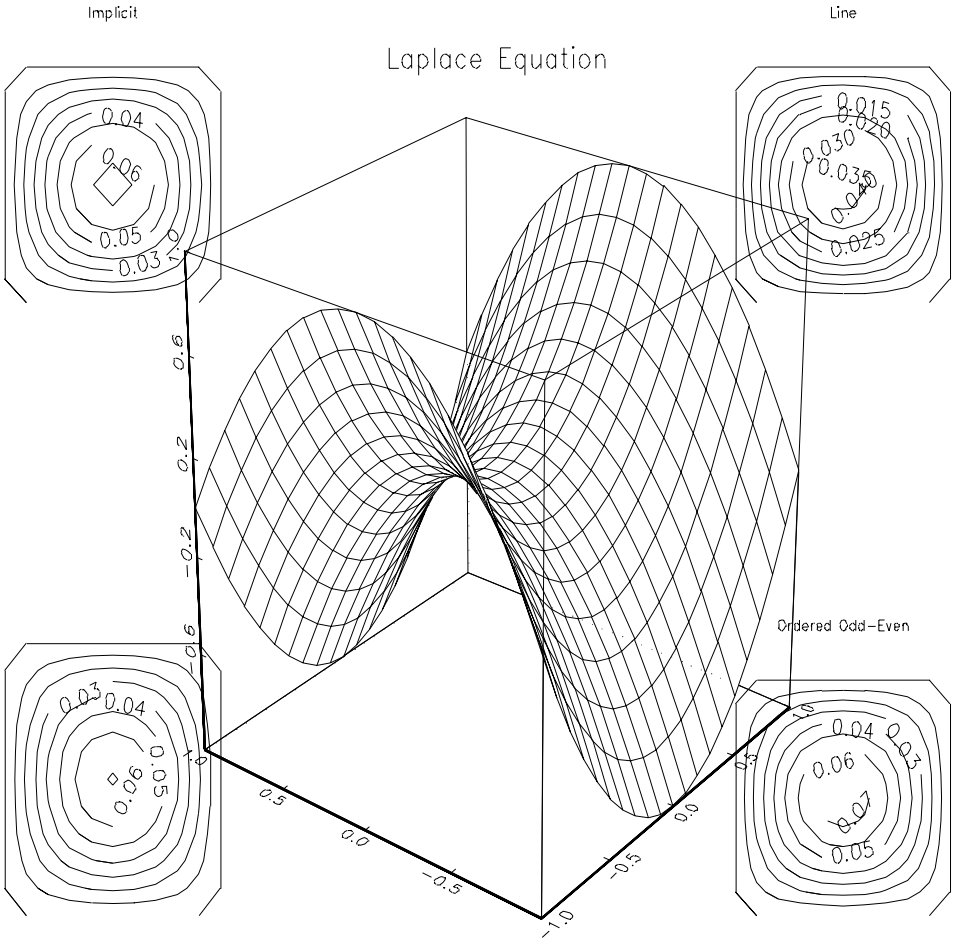
Implicit
Line
Laplace Equation
Ordered Odd-Even
0.04
0.06
0.05
0.03
0.015
0.020
0.030
0.035
0.040
0.025
0.6
0.2
−0.2
−0.6
1.0
0.5
0.0
−0.5
−1.0

Figure 4-7

Bibliography

[1] Gourlay, A.R. and S. McKee [1977], The construction of hopscotch methods for parabolic and elliptic equations in two space dimensions with a mixed derivative, *Journal of Computational and Applied Mathematics*, **3**, 201-205

[2] Kurpiel, A. and T. Roncalli [1998], Hopscotch methods for two-state financial models, FERC Working Paper, City Univesity Business School

Index