M368K Homework 12

§ 12.2 #6^1, 8^1, 10^1

Hershal Bhave (hb6279)

Due April 26, 2013

1 § 12.2

1.1 6^1

Use the Forward-Difference method to approximate the solution to the following parabolic partial differential equations. Use $\Delta x = \frac{1}{4}$ and $\Delta t = \frac{1}{10}$. Explicitly write the discrete equations. Compute up to $t = \frac{2}{10}$; compare approximate and exact solutions at each node at the final time.

1.1.1 b

$$\begin{aligned} \frac{\partial u}{\partial t} - \frac{1}{\pi^2}\frac{\partial^2 u}{\partial x^2} &= 0, \quad 0 < x < 1, \quad 0 < t; \\ u(0,t) &= u(1,t) = 0, \quad 0 < t, \\ u(x,0) &= \cos\left[\pi\left(x - \frac{1}{2}\right)\right], \quad 0 \leqslant x \leqslant 1. \end{aligned} \tag{1}$$

From the given equation we have the following constants:

$$\begin{aligned} h = \Delta x = \frac{1}{4} \qquad & \alpha^2 = \frac{1}{\pi^2} \\ k = \Delta t = \frac{1}{10} \qquad & \lambda = \alpha^2(k/h^2) = \frac{8}{5\pi^2} \end{aligned} \tag{2}$$

Now we can construct the A matrix and initial $\mathbf{w}$ vector using λ and $u(x,0)$.

$$A = \begin{pmatrix} 1-2\lambda & \lambda & 0 & \cdots & 0 \\ \lambda & 1-2\lambda & \lambda & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & \\ \vdots & \ddots & \ddots & \ddots & 0 \\ & & \ddots & \lambda & 1-2\lambda & \lambda \\ 0 & \cdots & 0 & \lambda & 1-2\lambda \end{pmatrix} \tag{3}$$

The discrete equations for the Forward-Difference method is

$$\mathbf{w}_{i,j+1} = (1-2\lambda)w_{i,j} + \lambda(w_{i+1,j} + w_{i-1,j}) \tag{4}$$

Which turns out to be

$$\mathbf{w}_{i,j+1} = \left(1 - \frac{16}{5\pi^2}\right) w_{i,j} + \frac{8}{5\pi^2}(w_{i+1,j} + w_{i-1,j}) \tag{5}$$

where

$$\mathbf{w}_{i,0} = \cos\left[\pi\left(x_i - \frac{1}{2}\right)\right] \tag{6}$$

And now further approximations of $\mathbf{w}$ can be computed by

$$\mathbf{w}^{(j)} = A\mathbf{w}^{(j-1)} \tag{7}$$

As you can see, higher orders of $\mathbf{w}$ can be obtained by simple multiplication in eq. (7).

Using the algorithm given in listing 1, I obtained the data in tables 1 and 2.

x_i	$u(x_i, 1/10)$	$w(i, 1/10)$	$\lvert u(x_i, 1/10) - w(i, 1/10)\rvert$
0	0	0	
0.25	0.63982	0.63996	1.4033×10^{-4}
0.50	0.90484	0.90504	1.9846×10^{-4}
0.75	0.63982	0.63996	1.4033×10^{-4}
1.00	0	0	

Table 1: Data for Number 6b for t=1/10

x_i	$u(x_i, 2/10)$	$w(i, 2/10)$	$\lvert u(x_i, 2/10) - w(i, 2/10)\rvert$
0	0	0	
0.25	0.57893	0.57918	2.5399×10^{-4}
0.50	0.81873	0.81909	3.5919×10^{-4}
0.75	0.57893	0.57918	2.5399×10^{-4}
1.00	0	0	

Table 2: Data for Number 6b for t=2/10

1.2 8^1

Repeat the question in section 1.1 using the Backward-Difference Algorithm.

1.2.1 b

Equation (2) still applies to this problem. We can construct the A matrix and initial $\mathbf{w}$ vector using λ and $u(x,0)$.

$$A = \begin{pmatrix} 1+2\lambda & -\lambda & 0 & \cdots & & 0 \\ -\lambda & 1+2\lambda & -\lambda & \ddots & & \vdots \\ 0 & \ddots & \ddots & \ddots & \ddots & \\ & \ddots & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & -\lambda & 1+2\lambda & -\lambda \\ 0 & \cdots & & 0 & -\lambda & 1+2\lambda \end{pmatrix} \tag{8}$$

The discrete equation for the Backward-Difference Method is

$$\mathbf{w}_{i,j-1} = (1+2\lambda)w_{i,j} - \lambda(w_{i+1,j} + w_{i-1,j}) \tag{9}$$

Which turns out to be

$$\mathbf{w}_{i,j-1} = \left(1 + \frac{16}{5\pi^2}\right) w_{i,j} - \frac{8}{5\pi^2}(w_{i+1,j} + w_{i-1,j}) \tag{10}$$

where

$$\mathbf{w}_{i,0} = \cos\left[\pi\left(x_i - \frac{1}{2}\right)\right] \tag{11}$$

And now further approximations of $\mathbf{w}$ can be computed by

$$\mathbf{w}^{(j-1)} = A\mathbf{w}^{(j)} \tag{12}$$

In the Backward-Difference method, higher orders of $\mathbf{w}$ can only be obtained by solving the linear system for $\mathbf{w}^{(j)}$ in eq. (12).

Using the algorithm given in listing 2, I obtained the data in tables 3 and 4.

x_i	$u(x_i, 1/10)$	$w(i, 1/10)$	$\lvert u(x_i, 1/10) - w(i, 1/10)\rvert$
0	0	0	
0.25	0.63982	0.64578	0.0059641
0.50	0.90484	0.91327	0.0084345
0.75	0.63982	0.64578	0.0059641
1.00	0	0	

Table 3: Data for Number 8b for t=1/10

x_i	$u(x_i, 2/10)$	$w(i, 2/10)$	$\lvert u(x_i, 2/10) - w(i, 2/10)\rvert$
0	0	0	
0.25	0.57893	0.58977	0.010843
0.50	0.81873	0.83407	0.015335
0.75	0.57893	0.58977	0.010843
1.00	0	0	

Table 4: Data for Number 8b for t=2/10

1.3 10^1

Repeat the question in section 1.1 using the Crank-Nicolson Algorithm.

1.3.1 b

Equation (2) still applies to this problem. We can construct the A and B matrices and initial $\mathbf{w}$ vector using λ and $u(x, 0)$.

$$A = \begin{pmatrix} 1+\lambda & -\frac{\lambda}{2} & 0 & \cdots & 0 \\ -\frac{\lambda}{2} & 1+\lambda & -\frac{\lambda}{2} & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & \\ & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & -\frac{\lambda}{2} & 1+\lambda & -\frac{\lambda}{2} \\ 0 & \cdots & 0 & -\frac{\lambda}{2} & 1+\lambda \end{pmatrix}$$

$$B = \begin{pmatrix} 1-\lambda & \frac{\lambda}{2} & 0 & \cdots & 0 \\ \frac{\lambda}{2} & 1-\lambda & \frac{\lambda}{2} & \ddots & \vdots \\ 0 & \ddots & \ddots & \ddots & \\ & \ddots & \ddots & \ddots & 0 \\ \vdots & & \ddots & \frac{\lambda}{2} & 1-\lambda & \frac{\lambda}{2} \\ 0 & \cdots & 0 & \frac{\lambda}{2} & 1-\lambda \end{pmatrix} \tag{13}$$

The discrete equation for the Crank-Nicolson method is

$$\frac{w_{i,j+1} - w_{i,j}}{k} = \frac{\alpha^2}{2}\left[\frac{w_{i+1,j} - 2w_{i,j} + w_{i-1,j}}{h^2} + \frac{w_{i+1,j} - 2w_{i,j} + w_{i-1,j}}{h^2}\right] \tag{14}$$

Which simplifies to

$$w_{i,j+1} - w_{i,j} = \frac{\lambda}{2}[w_{i+1,j} - 2w_{i,j} + w_{i-1,j} + w_{i+1,j} - 2w_{i,j} + w_{i-1,j}] \tag{15}$$

And turns out to look like

$$w_{i,j+1} - w_{i,j} = \frac{4}{5\pi^2}[w_{i+1,j} - 2w_{i,j} + w_{i-1,j} + w_{i+1,j} - 2w_{i,j} + w_{i-1,j}] \tag{16}$$

where

$$\mathbf{w}_{i,0} = \cos\left[\pi\left(x_i - \frac{1}{2}\right)\right] \tag{17}$$

And now further approximations of $\mathbf{w}$ can be computed by

$$A\mathbf{w}^{(j+1)} = B\mathbf{w}^{(j)} \tag{18}$$

In the Crank-Nicolson method, higher orders of $\mathbf{w}$ can only be obtained by solving the linear system for $\mathbf{w}^{(j+1)}$ in eq. (18).

Using the algorithm given in listing 2, I obtained the data in tables 5 and 6.

x_i	$u(x_i, 1/10)$	$w(i, 1/10)$	$\lvert u(x_i, 1/10) - w(i, 1/10)\rvert$
0	0	0	
0.25	0.63982	0.64300	0.0031842
0.50	0.90484	0.90934	0.0045032
0.75	0.63982	0.64300	0.0031842
1.00	0	0	

Table 5: Data for Number 10b for t=1/10

x_i	$u(x_i, 2/10)$	$w(i, 2/10)$	$\lvert u(x_i, 2/10) - w(i, 2/10)\rvert$
0	0	0	
0.25	0.57893	0.58471	0.0057767
0.50	0.81873	0.82690	0.0081695
0.75	0.57893	0.58471	0.0057767
1.00	0	0	

Table 6: Data for Number 10b for t=2/10

2 Programming Minilab

2.1 b

i	$u_i(0.6, 0.6, 5)$	Δx_i	Δy_i	Δt_i
1	0.95370	$\frac{1}{10}$	$\frac{1}{10}$	$\frac{1}{20}$
2	0.95438	$\frac{1}{15}$	$\frac{1}{15}$	$\frac{1}{25}$
3	-1.76134 $\times 10^{34}$	$\frac{1}{20}$	$\frac{1}{20}$	$\frac{1}{30}$

Table 7: Data for Minilab Part c

It seems that the computation does not approach a definite value because it is unstable at higher resolutions of t. That is, the value of Δt at steps 2 and 3 is above what can be considered stable for the Forward Difference Method, i.e. $\Delta t = 0.05 \nleq \frac{\Delta x^2 \Delta y^2}{2P\Delta x^2 + 2Q\Delta y^2} < 0.083333$ for step 2 and $\Delta t = 0.04 \nleq \frac{\Delta x^2 \Delta y^2}{2P\Delta x^2 + 2Q\Delta y^2} < 0.020833$ at step 3. We are not guaranteed stability at steps 2 and 3, so the solution does not converge to a definite value.

2.2 c

i	t_i	$u_{avg}(t_i)$
1	0.5	0.9917993860561917
2	3	0.9640316857440158
3	10	0.9357088345473462
4	20	0.9248950884495316
5	40	0.9221110405827277

Table 8: Data for Minilab Part c

3 Code

```
#!/usr/bin/octave
# Created by Hershal Bhave on 04/25/13
# For M368K HW12,  12.2 Number 6
# Written in GNU Octave
#
# Description: Computes the approximate solution to the PDE using the
# Forward-Difference method, where f=u(x,0), alphasq is alpha^2 in
# du/dt-alpha^2*(d^2u/dt^2), h is deltax, k is deltat, m is the length
# of x, and n is the length of t
#

function w = forwarddiff(f,alphasq,h,k,m,n)

  i = 1:(m-1);
  x = (i*h)';
  w = f(x);

  lambda = alphasq*(k/h^2);

  A(1,1) = 1-2*lambda;
  A(1,2) = lambda;

  for i=2:m-2
    A(i,i-1) = lambda;
    A(i,i) = 1-2*lambda;
    A(i,i+1) = lambda;
  endfor

  A(m-1,m-2) = lambda;
  A(m-1,m-1) = 1-2*lambda;

  for i=1:n
    w=A*w;
```

```
  endfor

endfunction
```

Listing 1: `forwarddiff.m`

```
#!/usr/bin/octave
# Created by Hershal Bhave on 04/25/13
# For M368K HW12,  12.2 Number 8
# Written in GNU Octave
#
# Description: Computes the approximate solution to the PDE using the
# Backward-Difference method, where f=u(x,0), alphasq is alpha^2 in
# du/dt-alpha^2*(d^2u/dt^2), h is deltax, k is deltat, m is the length
# of x, and n is the length of t
#

function w = backdiff(f,alphasq,h,k,m,n)

  i = 1:(m-1);
  x = (i*h)';
  w = f(x);

  lambda = alphasq*(k/h^2);

  A(1,1) = 1+2*lambda;
  A(1,2) = -lambda;

  for i=2:m-2
    A(i,i-1) = -lambda;
    A(i,i) = 1+2*lambda;
    A(i,i+1) = -lambda;
  endfor

  A(m-1,m-2) = -lambda;
  A(m-1,m-1) = 1+2*lambda;

  for i=1:n
    w=A\w;
  endfor

endfunction
```

Listing 2: `backdiff.m`

```
#!/usr/bin/octave
# Created by Hershal Bhave on 04/25/13
# For M368K HW12,  12.2 Number 10
# Written in GNU Octave
#
# Description: Computes the approximate solution to the PDE using the
# Crank-Nicolson method, where f=u(x,0), alphasq is alpha^2 in
# du/dt-alpha^2*(d^2u/dt^2), h is deltax, k is deltat, m is the length
# of x, and n is the length of t
#

function [w,u] = crankn2(f,alphasq,h,k,m,n)

  i = 1:(m-1);
  x = (i*h)';
  w = f(x);

  lambda = alphasq*(k/h^2)

  A(1,1) = 1+lambda;
  A(1,2) = -lambda/2;
  B(1,1) = 1-lambda;
  B(1,2) = lambda/2;

  for i=2:m-2
    A(i,i-1) = -lambda/2;
    A(i,i) = 1+lambda;
    A(i,i+1) = -lambda/2;
    B(i,i-1) = lambda/2;
    B(i,i) = 1-lambda;
    B(i,i+1) = lambda/2;
  endfor

  A(m-1,m-2) = -lambda/2;
  A(m-1,m-1) = 1+lambda;
  B(m-1,m-2) = lambda/2;
  B(m-1,m-1) = 1-lambda;

  for i=1:n
    w=A\B*w;
  endfor

endfunction
```

Listing 3: `crankn.m`

```
/*********************************************************
Program 12. Uses the forward-difference method to find an
approximate solution of a parabolic IBVP in a rectangular
domain of the form

 ut = P uxx + Q uyy + p ux + q uy + r u + eta,
                                a<=x<=b, c<=y<=d, 0<=t<=T

 u(a,y,t) = ga(y,t), u(b,y,t) = gb(y,t), c<=y<=d, 0<=t<=T
 u(x,c,t) = gc(x,t), u(x,d,t) = gd(x,t), a<=x<=b, 0<=t<=T

 u(x,y,0) = f(x,y), a<=x<=b, c<=y<=d


Inputs:
  PDEeval Function to evaluate P,Q,p,q,r,eta
  BCeval Function to evaluate ga,gb,gc,gd
  ICeval Function to evaluate f
  a,b,c,d Space domain parameters
  N,M Number of interior x,y pts (N+2,M+2 total pts)
  x Grid point vector: x(i)=a+i*dx, i=0...N+1
  y Grid point vector: y(j)=c+j*dy, j=0...M+1
  T Time interval parameter (final time)
  L Number of time steps

Outputs:
  x Grid point vector: x(i)=a+i*dx, i=0...N+1
  y Grid point vector: y(j)=c+j*dy, j=0...M+1
  u Approx soln at time t=T: u(i,j) is soln
               at x(i),y(j), i=0...N+1, j=0...M+1


Note 1: The function file fwddiff2D.cpp is incomplete; you'll
need to finish coding the method as indicated in that file.

Note 2: For any given problem, the functions PDEeval, BCeval
and ICeval must be changed.

Note 3: For any given problem, the grid parameters a,b,c,d,T
and N,M,L must be specified.

Note 4: To compile this program use the command (all on one
line)

  c++ -o program12 matrix.cpp fwddiff2D.cpp program12.cpp

Note 5: The program output is written to a file.
*********************************************************/
#include <iostream>
#include <iomanip>
#include <fstream>
#include <stdlib.h>
#include <math.h>
#include "matrix.h"
using namespace std;


/*** Define output file ***/
const char myfile[20]="program12.out" ;
ofstream prt(myfile) ;


/*** Declare external function ***/
int fwddiff2D(int, int, double, double, double, double,
```

```
                vector&, vector&, double&, double&, matrix&) ;

/*** Define P(x,y,t), Q(x,y,t),
                  p(x,y,t), q(x,y,t), r(x,y,t), eta(x,y,t) ***/
void PDEeval(const double& x, const double& y, double& t,
                 double& P, double& Q, double& p, double& q,
                                    double& r, double& eta){

  double galpha = 0.03;

  double alpha[] = {0.2, 0.6, -0.5};
  double beta[] = {0.2, 0.2, 0.0};
  double gamma[] = {0.6, -0.7, -0.6};

  P = galpha;
  Q = galpha;
  p = -y;
  q = x;
  r = 0;
  eta = 0;

  // Unrolled at compile-time
  for(int i=0; i<3; i++) {
    eta += gamma[i]*exp(-30*pow(x-alpha[i],2) - 30*pow(y-beta[i],2));
  }

}

/*** Define ga(y,t), gb(y,t), gc(x,t), gd(x,t) ***/
void BCeval(const double& x, const double& y, double& t,
           double& ga, double& gb, double& gc, double& gd){

  ga = 1.0;
  gb = 1.0;
  gc = 1.0;
  gd = 1.0;

}

/*** Define f(x,y) ***/
void ICeval(const double& x, const double& y, double& f){

  f = 1.0;
}


int main() {
  /*** Define problem parameters ***/

  // Part B Parameters
  // First Grid Parameters (Part b)
  // int N=19, M=19, L=100, success_flag=0 ;
  // double T=5;

  // Second Grid Parameters (Part b)
  // int N=29, M=29, L=125, success_flag=0 ;
  // double T=5;

  // Third Grid Parameters (Part b)
  // int N=39, M=39, L=150, success_flag=0 ;
  // double T=5;

  // Part C Parameters (T<0.5*0.02083)
  // First Grid Parameters (Part c)
  int N=29, M=29, L=50, success_flag=0 ;
```

```
double T=0.5;

// Second Grid Parameters (Part c)
// int N=29, M=29, L=300, success_flag=0 ;
// double T=3;

// Third Grid Parameters (Part c)
// int N=29, M=29, L=1000, success_flag=0 ;
// double T=10;

// Fourth Grid Parameters (Part c)
// int N=29, M=29, L=2000, success_flag=0 ;
// double T=20;

// Fifth Grid Parameters (Part c)
// int N=29, M=29, L=4000, success_flag=0 ;
// double T=40;

matrix u(N+2,M+2) ;
vector x(N+2), y(M+2) ;
double a=-1, b=1, c=-1, d=1 ;
double dx=(b-a)/(N+1), dy=(d-c)/(M+1) ;
double dt=T/L ;

/*** Construct xy-grid ***/
for(int i=0; i<=N+1; i++){
  x(i) = a + i*dx ;
}
for(int j=0; j<=M+1; j++){
  y(j) = c + j*dy ;
}

/*** Load initial condition ***/
double t=0 ;
double gLeft, gRight, gBottom, gTop, f ;
for(int i=0; i<=N+1; i++){ //actual i-index on grid
  for(int j=0; j<=M+1; j++){ //actual j-index on grid
    BCeval(x(i),y(j),t,gLeft,gRight,gBottom,gTop) ;
    ICeval(x(i),y(j),f) ;
    if( j==M+1 ){ u(i,j) = gTop ; }
    if( i==N+1 ){ u(i,j) = gRight ; }
    if( i==0 ){ u(i,j) = gLeft ; }
    if( j==0 ){ u(i,j) = gBottom ; }
    if((i>0)&&(i<N+1)&&(j>0)&&(j<M+1)){ u(i,j) = f ; }
  }
}

/*** Call fwd-diff method at each time step (overwrites u) ***/
for(int n=0; n<L; n++){
  success_flag = fwddiff2D(N,M,a,b,c,d,x,y,t,dt,u) ;
  t = t + dt ;
}

/*** Print results at final time to output file ***/
prt.setf(ios::fixed) ;
prt << setprecision(5) ;
cout << "Fwd-Diff-2D: output written to " << myfile << endl ;
prt << "Fwd-Diff-2D results" << endl ;
prt << "Number of interior x-grid pts: N = " << N << endl ;
prt << "Number of interior y-grid pts: M = " << M << endl ;
prt << "Number of time steps: L = " << L << endl ;
prt << "Final time of simulation: t = " << t << endl ;
prt << "Approximate solution at time t: x_i, y_j, u_ij" << endl ;
for(int i=0; i<=N+1; i++){
  for(int j=0; j<=M+1; j++){
```

```
      prt << setw(8) << x(i) ;
      prt << " " ;
      prt << setw(8) << y(j) ;
      prt << " " ;
      prt << setw(8) << u(i,j) ;
      prt << endl;
    }
  }

  return 0 ; //terminate main program
}
```

Listing 4: `program12.cpp`

```
/*************************************************************
Function to implement the forward-difference method to find
an approximate solution of a parabolic IBVP in a rectangular
domain of the form

 ut = P uxx + Q uyy + p ux + q uy + r u + eta,
                                a<=x<=b, c<=y<=d, 0<=t<=T

 u(a,y,t) = ga(y,t), u(b,y,t) = gb(y,t), c<=y<=d, 0<=t<=T
 u(x,c,t) = gc(x,t), u(x,d,t) = gd(x,t), a<=x<=b, 0<=t<=T

 u(x,y,0) = f(x,y), a<=x<=b, c<=y<=d


Inputs:
  PDEeval Function to evaluate P,Q,p,q,r,eta
  BCeval Function to evaluate ga,gb,gc,gd
  a,b,c,d Space domain parameters
  N,M Number of interior x,y pts (N+2,M+2 total pts)
  x Grid point vector: x(i)=a+i*dx, i=0...N+1
  y Grid point vector: y(j)=c+j*dy, j=0...M+1
  t,dt Current time t and time step dt
  u Approx soln at time t: u(i,j) is soln
               at x(i),y(j), i=0...N+1, j=0...M+1

Outputs:
  x Grid point vector: x(i)=a+i*dx, i=0...N+1
  y Grid point vector: y(j)=c+j*dy, j=0...M+1
  u Approx soln at time t+dt: u(i,j) is soln
               at x(i),y(j), i=0...N+1, j=0...M+1


Note 1: This function is incomplete; you'll need to finish
coding the method as indicated below.

Note 2: The functions PDEeval and BCeval are assumed
to be defined externally (e.g. by calling program).

Note 3: Rather than use the single-label index l=1...NM
for the interior xy-grid, it is convenient to use the
double-label indices i=0...N+1, j=0...M+1 for the entire
xy-grid.
*************************************************************/
#include <iostream>
#include <stdlib.h>
#include <math.h>
#include "matrix.h"
#include <stdio.h>
using namespace std;


/*** Ext fun: PDEeval(x,y,t,P,Q,p,q,r,eta) ***/
void PDEeval(const double&, const double&, double&,
                         double&, double&, double&,
                           double&, double&, double&) ;

/*** Ext fun: BCeval(x,y,t,gLeft,gRight,gBottom,gTop) ***/
void BCeval(const double&, const double&, double&,
                   double&, double&, double&, double&) ;


/*** Main function: forward-difference method ***/
int fwddiff2D(int N, int M,
             double a, double b, double c, double d,
```

```
           vector& x, vector& y, double& t, double& dt,
                                        matrix& u){

int success_flag=0 ;
double P, Q, p, q, r, eta, tn, tnn ;
double gLeft, gRight, gBottom, gTop ;
double uLeft=0, uRight=0, uBottom=0, uTop=0, uCenter=0 ;
double dx=(b-a)/(N+1), dy=(d-c)/(M+1) ; //grid params
double ahatij=0,bhatij=0,chatij=0,dhatij=0,ehatij=0 ; //fwd-diff params
matrix unew(N+2,M+2) ; //temporary variable


/*** Compute u at t_{n+1} at interior grid points ***/
for(int i=1; i<=N; i++){ //actual i-index on grid
  for(int j=1; j<=M; j++){ //actual j-index on grid

    tn = t ;
    PDEeval(x(i),y(j),tn,P,Q,p,q,r,eta) ;
    BCeval(x(i),y(j),tn,gLeft,gRight,gBottom,gTop) ;

    /*** Build the ahat_{i,j},...,dhat_{i,j} params here
    and compute u^{n+1} in terms of u^{n}, eta-values
    and g-values. A few lines are given as an example.
    We use the more description notation uCenter in
    place of u_{i,j}, uTop in place of u_{i,j+1}, etc. ***/

    ahatij = P/(dx*dx) - p/(2.0*dx) ;
    bhatij = r - 2.0*P/(dx*dx) - 2.0*Q/(dy*dy) ;
    chatij = P/(dx*dx) + p/(2.0*dx) ;
    dhatij = Q/(dy*dy) + q/(2.0*dy);
    ehatij = Q/(dy*dy) - q/(2.0*dy);

    if(i>1){
      uLeft = u(i-1,j) ; //interior value
    } else {
      uLeft = gLeft ; //boundary value
    }

    if(i<N){
      uRight = u(i+1,j) ; //interior value
    } else {
      uRight = gRight ; //boundary value
    }

    if(j>1) {
uBottom = u(i,j-1); // interior value
    } else {
uBottom = gBottom; // boundary value
    }

    if(j<M) {
uTop = u(i,j+1); // interior value
    } else {
uTop = gTop;  // boundary value
    }

    uCenter = u(i,j);

    unew(i,j) = u(i,j) + dt*dhatij*uTop
                       + dt*ahatij*uLeft
                       + dt*bhatij*uCenter
                       + dt*chatij*uRight
                       + dt*ehatij*uBottom
                       + dt*eta ; //fwd-diff formula
  }
```

```
  }

  /*** Compute u at t_{n+1} at boundary grid points ***/
  for(int i=0; i<=N+1; i++){ //actual i-index on grid
    tnn = t + dt ;
    BCeval(x(i),y(M+1),tnn,gLeft,gRight,gBottom,gTop) ;
    unew(i,M+1) = gTop ;
    BCeval(x(i),y(0),tnn,gLeft,gRight,gBottom,gTop) ;
    unew(i,0) = gBottom ;
  }
  for(int j=0; j<=M+1; j++){ //actual j-index on grid
    tnn = t + dt ;
    BCeval(x(N+1),y(j),tnn,gLeft,gRight,gBottom,gTop) ;
    unew(N+1,j) = gRight ;
    BCeval(x(0),y(j),tnn,gLeft,gRight,gBottom,gTop) ;
    unew(0,j) = gLeft ;
  }

  u = unew ; //update u-matrix with new values

  return success_flag=1 ;
}
```

Listing 5: `fwddiff2D.cpp`