M368K Homework 11

§ 11.5 #12[1]   § 12.1 #2, 8[2]

Hershal Bhave (hb6279)

Due April 19, 2013

1 § 11.5

1.1 12[1]

1.1.1 12[1]a

Show that the matrix given by the piecewise linear basis function is positive definite. Assume $p(x) > 0$ and $g(x) \geqslant 0$. Show $c^\intercal Ac = \int_a^b p[v']^2 + qv^2 \mathrm{d}x > 0$ where $v(x) = \sum_{i=1}^N c_i\phi_i(x)$.

$$\begin{aligned} c^\intercal Ac &= \sum_{i=1}^N \left(\sum_{j=1}^N c_i A_{i,j} c_j \right) \\ &= \sum_{i=1}^N \left(\sum_{j=1}^N c_i \left[\int_a^b pv_i'v_j'\mathrm{d}x + \int_b^a qv_iv_j\mathrm{d}x \right] c_j \right) \\ &= \int_b^a pv'^2 + qv^2\mathrm{d}x \end{aligned} \tag{1}$$

Since it was assumed that $p(x) > 0$ and $g(x) \geqslant 0$ and it was given that $v(x) = \sum_{i=1}^N c_iv_i(x)$, we know that $v(x)^2 \geqslant 0$. Since $v(x)^2$ will always be greater than or equal to zero, the integral $\int_a^b p[v']^2 + qv^2\mathrm{d}x$ will also be greater than or equal to zero based on $v(x)^2 \geqslant 0$ and the previous assumptions. This implies that $c^\intercal Ac$ will alway be greater than or equal to zero for $v(x) \in \mathbb{R}$

■

1.1.2 12[1]b

Explain each of the implications $c^\intercal Ac = 0 \implies v'(x) \equiv 0$ in each subinterval $(x_i, x_{i+1}) \implies v(x) \equiv 0$ in $[a, b] \implies c = 0$ and conclude that A is positive definite.

If $c^\intercal Ac = \int_a^b p[v']^2 + qv^2 \mathrm{d}x = 0$ then both $\int_a^b p[v']^2 \mathrm{d}x$ or $\int_a^b qv^2 \mathrm{d}x$ are zero. Given that $p(x)$ is strictly greater than zero, that must imply that $\int_a^b qv^2 \mathrm{d}x = 0$ is a possibility, leaving $\int_a^b p[v']^2 \mathrm{d}x = 0$. Since $p(x)$ cannot equal zero, we must conclude that $v'(x) \equiv 0$. If $v'(x) = \sum_{i=1}^{N} c_i v_i(x) \equiv 0$, we know that $v_i(x)$ is nonzero on (x_{i-1}, x_{i+1}), so that must mean that $c_i \equiv 0$.

Since $A = \int_a^b p[v']^2 + qv^2 \mathrm{d}x$ is symmetric and $c^\intercal Ac$ is only zero if $c_i = 0$ then $c^\intercal Ac > 0$ for $c_i \neq 0$. Thus $c^\intercal Ac$ is positive definite.

■

2 § 12.1

2.1 2

Use the Poisson Equation Finite Difference Algorithm to approximate the solution to the elliptic partial difference equation.

$$\frac{\partial^2 u}{\partial x^2} + \frac{\partial^2 u}{\partial y^2} = 0 \qquad 1 < x < 2, \quad 0 < y < 1$$

$$\begin{aligned} u(x,0) &= 2\ln x, & u(x,1) &= \ln(x^2+1), & 1 \leqslant x \leqslant 2; \\ u(1,y) &= \ln(y^2+1), & u(2,y) &= ln(y^2+4), & 0 \leqslant y \leqslant 1. \end{aligned}$$

Use $h = k = \frac{1}{3}$ and compare the results to the actual solution $u(x,y) = \ln(x^2+y^2)$.

Using the algorithm defined in listing 1 with tolerance 10^{-6}, I obtained the data in table 1.

i	j	x_i	y_i	$w_{i,j}^{(19)}$	$u(x_i, y_i)$	$\mid u(x_i, y_i) - w_{i,j}^{(19)} \mid$
1	1	1.333	0.333	0.63480	0.63599	1.1844×10^{-3}
1	2	1.333	0.667	0.79850	0.79851	7.3735×10^{-6}
2	1	1.667	0.333	1.05999	1.06087	8.7950×10^{-4}
2	2	1.667	0.667	1.16982	1.17007	2.5035×10^{-4}

Table 1: Poisson Equation Finite Difference Approximation for § 12.1 Number 2

2.2 8^2

A 6-cm by 5-cm rectangular silver plate has heat being uniformly generated at each point at the rate q. Let x represent the distance along the edge of the plate of length 6 cm and y be the distance along the edge of the plate of length 5 cm. Suppose the temperature u along the edges is kept at the following temperatures:

$$\begin{aligned} u(x,0) &= x(6-x), \quad u(x,5) = 0, \qquad 0 \leqslant x \leqslant 6; \\ u(0,y) &= y(5-y), \quad u(6,y) = 0, \qquad 0 \leqslant y \leqslant 5. \end{aligned}$$

where the origin lies at a corner of the plate with coordinates (0,0) and the edges lie along the positive x- and y-axes. The steady-state temperature $u = u(x,y)$ satisfies the Poisson's equation:

$$\frac{\partial^2 u}{\partial x^2}(x,y) + \frac{\partial^2 u}{\partial y^2}(x,y) = -\frac{q}{K}, \qquad 0 < x < 6, \quad 0 < y < 5,$$

where K, the thermal conductivity, is 1.04 cal/cm $\cdot$ deg $\cdot$ s and q, the rate heat is being generated, is 1.5 cal/cm^3 $\cdot$ s. Approximate the temperature $u(x,y)$ using the Poisson Equation Finite Difference Approximation Algorithm in listing 1 and two interior nodes in each direction.

The general centered-difference formulas are

$$\begin{aligned} \frac{\partial^2 u}{\partial x^2}(x_i,y_j) &= \frac{u(x_{i+1},y_j) - 2u(x_i,y_j) + u(x_{i-1},y_j)}{h^2} - \frac{h^2}{12}\frac{\partial^4 u}{\partial x^4}(\xi_i,y_j) \\ \frac{\partial^2 u}{\partial y^2}(x_i,y_j) &= \frac{u(x_i,y_{j+1}) - 2u(x_i,y_j) + u(x_i,y_{j-1})}{k^2} - \frac{k^2}{12}\frac{\partial^4 u}{\partial y^4}(x_i,\eta_j) \end{aligned} \tag{2}$$

where $\xi_i \in (x_{i-1}, x_{i+1})$ and $\eta_j \in (y_{j-1}, y_{j+1})$. Writing this in difference-equation form, we get

$$2\left[\left(\frac{h}{k}\right)^2 + 1\right] w_{i,j} - (w_{i+1,j} + w_{i-1,j}) - \left(\frac{h}{k}\right)^2 (w_{i,j+1} + w_{i,j-1}) = -h^2 f(x_i,y_j) \tag{3}$$

Which can simplify to

$$\beta w_{i,j} - (w_{i+1,j} + w_{i-1,j}) - \alpha(w_{i,j+1} + w_{i,j-1}) = \gamma \tag{4}$$

Where $\alpha = \left(\frac{h}{k}\right)^2$, $\beta = 2(\alpha + 1)$, and $\gamma = -h^2\frac{q}{K}$. Expressing this in terms of the relabled interior grid points $w_i = u(P_i)$, we get the equations at the points P_i.

$$\begin{aligned} P_1 : \beta w_1 - w_2 - \alpha w_3 \quad &= w_{0,2} + \cancel{\alpha w_{1,3}} + \gamma = \frac{50}{9} + \gamma \\ P_2 : \beta w_2 - w_1 - \alpha w_4 \quad &= \cancel{w_{3,2}} + \cancel{\alpha w_{2,3}} + \gamma = \gamma \\ P_3 : \beta w_3 - w_4 - \alpha w_1 \quad &= w_{0,1} + \alpha w_{1,0} + \gamma = \frac{50}{9} + 8\alpha + \gamma \\ P_4 : \beta w_4 - w_3 - \alpha w_2 \quad &= \cancel{w_{3,1}} + \alpha w_{2,0} + \gamma = 8 + \gamma \end{aligned} \tag{5}$$

In matrix form

$$\begin{pmatrix} \beta & -1 & -\alpha & 0 \\ -1 & \beta & 0 & -\alpha \\ -\alpha & 0 & \beta & -1 \\ 0 & -\alpha & -1 & \beta \end{pmatrix} \begin{pmatrix} w_1 \\ w_2 \\ w_3 \\ w_4 \end{pmatrix} = \begin{pmatrix} \frac{50}{9} + \gamma \\ \gamma \\ \frac{50}{9} + 8\alpha + \gamma \\ 8 + \gamma \end{pmatrix} \tag{6}$$

Using the algorithm defined in listing 1 with tolerance 10^{-6}, I obtained the data in table 1.

i	j	x_i	y_i	$w_{i,j}^{(19)}$
1	1	2	1.667	7.5719
1	2	2	3.333	5.4064
2	1	4	1.667	6.3206
2	2	4	3.333	4.1552

Table 2: Poisson Equation Finite Difference Approximation for § 12.1 Number 8^2

3 Minilab

3.1 Part a

Program output for S 12.1 Number 8^2 is in table 3 and is consistent with the results I obtained before.

x_i	y_i	$u_{i,j}$
0.00000	0.00000	0.00000
0.00000	1.66667	5.55556
0.00000	3.33333	5.55556
0.00000	5.00000	0.00000
2.00000	0.00000	8.00000
2.00000	1.66667	7.57186
2.00000	3.33333	5.40645
2.00000	5.00000	0.00000
4.00000	0.00000	8.00000
4.00000	1.66667	6.32061
4.00000	3.33333	4.15520
4.00000	5.00000	0.00000
6.00000	0.00000	0.00000
6.00000	1.66667	0.00000
6.00000	3.33333	0.00000
6.00000	5.00000	0.00000

Table 3: Data for Minilab part a

4 Part b

The concentration of pollutatnts at the school's location (0.8, 0.8) is approximately 6.880% (0.06880). The maximum concentration of pollutants appears at (0.36667,0.23333) and is approximately 23.171% (0.23171).

5 Part c

With the second factory at location (0.2,0.6), the concentration of pollutants at the school is approximately 0.012% (0.00012) and the maximum concentration of pollutants is approximately 27.998% (0.27998) at location (0.10000, 0.23333).

With the second factory at location (0.8,0.2), the concentration of pollutants at the school is approximately 0.09% (0.00090) and the maximum concentration of pollutants is approximately 35.709% (0.35709) at location (0.13333, 0.20000).

It would seem that having the second factory at location (0.8,0.2) would have the lowest impact on the school under the conditions we assumed.

6 Code

6.1 Poisson Finite Difference Algorithm

```
#!/usr/bin/octave
# Created by Hershal Bhave on 04/18/13
# For M368K HW11,  12.1 Number 2
# Written in GNU Octave
#
# Description: Uses the Finite-Difference algorithm to approximate the
# solution to a given Poisson equation f, its boundary conditions g, x
# boundaries a and b, y boundaries c and d, and grid dimensions m and n
#

function [x,y,w,l] = poissonFinDiff(a,b,c,d,f,g,m,n,N)

  if (m<3 || n<3)
    error("m and n must be greater than 3");
  endif

  tol = 10^-10;

  # Step 1
  h = (b-a)/n;
  k = (d-c)/m;

  # Step 2, 3
  x = a + [1:(n-1)]*h;
  y = c + [1:(m-1)]*k;

  # Step 4
  w = zeros(n-1,m-1);

  # Step 5
  lam = h^2/k^2;
  mu = 2*(1+lam);
  l=0;

  # Step 6
  do
    # Step 7
    z = (-h^2*f(x(1),y(m-1)) + g(a,y(m-1)) + lam*g(x(1),d) + lam*w(1,m-2) + w(2,m-1))/mu;
    NORM = abs(z-w(1,m-1));
    w(1,m-1) = z;

    # Step 8
    for i = 2:(n-2)
      z = (-h^2*f(x(i),y(m-1)) + lam*g(x(i),d) + w(i-1,m-1) + w(i+1,m-1) + lam*w(i,m-2))/mu;
      NORM = max(abs(w(i,m-1)-z), NORM);
      w(i,m-1) = z;
    endfor

    # Step 9
    z = (-h^2*f(x(n-1),y(m-1)) + g(b,y(m-1)) + lam*g(x(n-1),d) + w(n-2,m-1) + lam*w(n-1,m-2))/mu;
    NORM = max(abs(w(n-1,m-1)-z), NORM);
    w(n-1,m-1) = z;

    # Step 10
    for j = (m-2):-1:2

      # Step 11
      z = (-h^2*f(x(1),y(j)) + g(a,y(j)) + lam*w(1,j+1) + lam*w(1,j-1) + w(2,j))/mu;
      NORM = max(abs(w(1,j)-z), NORM);
      w(1,j) = z;

      # Step 12
      for i=2:(n-2)
  z = (-h^2*f(x(i),y(j)) + w(i-1,j) + lam*w(i,j+1) + w(i+1,j) + lam*w(i,j-1))/mu;
  NORM = max(abs(w(i,j)-z), NORM);
  w(i,j) = z;
      endfor

      # Step 13
      z = (-h^2*f(x(n-1),y(j)) + g(b,y(j)) + w(n-2,j) + lam*w(n-1,j+1) + lam*w(n-1,j-1))/mu;
      NORM = max(abs(w(n-1,j)-z), NORM);
      w(n-1,j) = z;

    endfor

    # Step 14
    z = (-h^2*f(x(1),y(1)) + g(a,y(1)) + lam*g(x(1),c) + lam*w(1,2) + w(2,1))/mu;
    NORM = max(abs(w(1,1)-z), NORM);
    w(1,1) = z;

    # Step 15
    for i=2:(n-2)
      z = (-h^2*f(x(i),y(1)) + lam*g(x(i),c) + w(i-1,1) + lam*w(i,2) + w(i+1,1))/mu;
      NORM = max(abs(w(i,1)-z), NORM);
      w(i,1) = z;
    endfor

    # Step 16
    z = (-h^2*f(x(n-1),y(1)) + g(b,y(1)) + lam*g(x(n-1),c) + w(n-2,1) + lam*w(n-1,2))/mu;
    NORM = max(abs(w(n-1,1)-z), NORM);
    w(n-1,1) = z;

    l++;
    until l>N || NORM<tol

  if(l>N)
    error("Maximum iterations exceeded\n");
  endif

endfunction
```

Listing 1: poissonFinDiff.m

6.2 Program 11

```
/*************************************************************
Program 11. Uses the central-difference method to find an
approximate solution of an elliptic BVP in a rectangular
domain of the form

 P uxx + Q uyy + p ux + q uy + r u = f, a<=x<=b, c<=y<=d
 u(a,y) = ga(y), u(b,y) = gb(y), c<=y<=d
 u(x,c) = gc(x), u(x,d) = gd(x), a<=x<=b


Inputs:
  PDEeval Function to evaluate P,Q,p,q,r,f
  BCeval Function to evaluate ga,gb,gc,gd
  a,b,c,d 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

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: u(i,j)=soln at x(i),y(j),
              i=0...N+1, j=0...M+1


Note 1: The function file linearcd2D.cpp is incomplete;
you'll need to code the A-matrix and G-vector as
indicated in that file.

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

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

Note 4: Gauss elimination is used to solve the system,
so only moderate values of N,M should be used.

Note 5: To compile this program use the command (all
on one line)

  c++ -o program11 matrix.cpp gauss_elim.cpp
                       linearcd2D.cpp program11.cpp

Note 6: 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]="program11.out" ;
ofstream prt(myfile) ;


/*** Declare external function ***/
int linearcd2D(int, int, double, double, double, double,
                                    vector&, vector&, matrix&) ;


/*** Define P(x,y), Q(x,y), p(x,y), q(x,y), r(x,y), f(x,y) ***/
void PDEeval(const double& x, const double& y,
            double& P, double& Q, double& p, double& q,
                                    double& r, double& f){
  // For 8
  // P = 1 ;
  // Q = 1 ;
  // p = 0 ;
  // q = 0 ;
  // r = 0 ;
  // f = -1.5/1.04;

  // For Part b
  // double x0 = 0.2;
  // double y0 = 0.2;
  // P = Q = 0.3 + 0.05*y;
  // p = -(10 - 10.0*x);
  // q = -5.0*y;
  // r = 0.0;
  // f = -10.0*exp(-30*pow(x-x0,2)-30*pow(y-y0,2));

  // For Part c
  double x0 = 0.2;
  double y0 = 0.2;
  double x1 = 0.8;
  double y1 = 0.2;
  P = Q = 0.3 + 0.05*y;
  p = 10-10*x;
  q = 5*y;
  r = 0;
  f = -(10*exp(-30*pow(x-x0,2)-30*pow(y-y0,2))
  + 8*exp(-30*pow(x-x1,2)-30*pow(y-y1,2)));
}

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

  // gLeft,gRight,gBottom,gTop -> ga,gb,gc,gd

  // For problem 8
  // ga = y*(5-y);
  // gb = 0;
  // gc = x*(6-x);
  // gd = 0;

  // ga=0;
  // gb=200*y;
  // gc=0;
  // gd=200*x;

  // For Parts b and c
  ga = 0;
  gb = 0;
  gc = 0;
  gd = 0;
}

int main() {
  /*** Define problem parameters ***/
  // For 8
  // int N=2, M=2, success_flag=0 ;
  // double a=0, b=6, c=0, d=5;

  // For Parts b and c
  int N=29, M=29, success_flag=0 ;
  double a=0, b=1, c=0, d=1;

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

  /*** Construct 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 ;
  }


  /*** Call central-difference method ***/
  success_flag=linearcd2D(N,M,a,b,c,d,x,y,u) ;


  /*** Print results to output file ***/
  prt.setf(ios::fixed) ;
  prt << setprecision(5) ;
  cout << "Linear-CD-2D: output written to " << myfile << endl ;
  prt << "Linear-CD-2D results" << endl ;
  prt << "Number of interior x-grid pts: N = " << N << endl ;
  prt << "Number of interior y-grid pts: M = " << M << endl ;
  prt << "Approximate solution: 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 2: `program11.cpp`

6.3 Linear Centered-Difference 2D

```
/*************************************************************
Function to implement the central-difference method to
find an approximate solution of an elliptic BVP in a
rectangular domain of the form

 P uxx + Q uyy + p ux + q uy + r u = f, a<=x<=b, c<=y<=d
 u(a,y) = ga(y), u(b,y) = gb(y), c<=y<=d
 u(x,c) = gc(x), u(x,d) = gd(x), a<=x<=b


Inputs:
  PDEeval Function to evaluate P,Q,p,q,r,f
  BCeval Function to evaluate ga,gb,gc,gd
  a,b,c,d 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

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: u(i,j)=soln at x(i),y(j),
               i=0...N+1, j=0...M+1


Note 1: This function is incomplete; you'll need to code
the A-matrix and G-vector as indicated below. The
central-difference equation system is A uint = G.

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

Note 3: The unknowns in the central-difference equation
system are uint(l) = u(i,j) where l=i+N(M-j), i=1...N,
j=1...M which gives l=1...NM. The boundary values for
u(i,j) are obtained from the BCeval function.

Note 4: Gauss elimination with partial pivoting is used to
solve the system.
*************************************************************/
#include <iostream>
#include <stdlib.h>
#include <math.h>
#include "matrix.h"
using namespace std;


/*** Ext fun: gauss_elim(A,G,uint) ***/
int gauss_elim(matrix&, vector&, vector&) ;


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

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


/*** Aux fun: build A and G for cent-diff system ***/
```

```
void AGeval(int N, int M,
           double a, double b, double c, double d,
           vector& x, vector& y, matrix& A, vector& G){

  int l, ll ;
  double P, Q, p, q, r, f ;
  double gLeft, gRight, gBottom, gTop ;
  double dx=(b-a)/(N+1), dy=(d-c)/(M+1) ; //grid parameters
  double al, bl, cl, dl, el ; //A-matrix parameters

  A=0 ; G=0 ;
  for(int i=1; i<=N; i++){ //actual i-index on grid
    for(int j=1; j<=M; j++){ //actual j-index on grid
      ll = i + N*(M-j) ; //actual l-label on grid (1...NM)
      l = ll-1 ; //C++ array index (0...NM-1)

      /*** Build the entries of the A-matrix and G-vector here.
           A few entries of A are shown as an example. We use
           the more descriptive notation gLeft,gRight,gBottom,
           gTop in place of ga,gb,gc,gd. ***/

      PDEeval(x(i),y(j),P,Q,p,q,r,f) ; //P,Q,p,q,r,f values
      BCeval(x(i),y(j),gLeft,gRight,gBottom,gTop) ; //ga,gb,gc,gd values

      al = (dy/dx)*P - (dy/2.0)*p ;
      bl = dx*dy*r - (2.0*dy/dx)*P - (2.0*dx/dy)*Q ;
      cl = (dy/dx)*P + (dy/2.0)*p ;
      dl = (dx/dy)*Q + (dx/2.0)*q ;
      el = (dx/dy)*Q - (dx/2.0)*q ;

      if( i>1 ){ A(l,l-1) = al ; }
      A(l,l) = bl ;
      if( i<N ){ A(l,l+1) = cl ; }

      if( j<M ){ A(l,l-N) = dl ; }
      if( j>1 ){ A(l,l+N) = el ; }

      G(l)=dy*dx*f;
      if(j==M) {
  G(l)=G(l)-dl*gTop;
      }
      if(j==N) {
  G(l)=G(l)-cl*gRight;
      }
      if(i==1) {
  G(l)=G(l)-al*gLeft;
      }
      if(j==1) {
  G(l)=G(l)-el*gBottom;
      }
    }
  }
}


/*** Main function: central-difference method ***/
int linearcd2D(int N, int M,
               double a, double b, double c, double d,
                           vector& x, vector& y, matrix& u){

  int l, ll, success_flag=0 ;
  matrix A(N*M,N*M) ;
  vector uint(N*M), G(N*M) ;
  double gLeft, gRight, gBottom, gTop ;
  /*** Build A-matrix and G-vector ***/
  AGeval(N,M,a,b,c,d,x,y,A,G) ;
  cout << "Linear-CD-2D: arrays assembled" << endl ;

  /*** Solve linear system ***/
  cout << "Linear-CD-2D: solving........." << endl ;
  gauss_elim(A,G,uint) ;
  cout << "Linear-CD-2D: equations solved" << endl ;

  /*** Assemble total solution: interior and boundary ***/
  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),gLeft,gRight,gBottom,gTop) ;
      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)){
         ll = i + N*(M-j) ; //actual l-label on grid (1...NM)
         l = ll-1 ; //C++ array index (0...NM-1)
         u(i,j) = uint(l) ; //interior soln
      }
    }
  }
  cout << "Linear-CD-2D: solution assembled" << endl ;

  return success_flag=1 ;
}
```

Listing 3: `linearcd2D.cpp`