M368K Homework 9

§ 11.1 #2a[1], 9 § 11.2 #4a[2]

Hershal Bhave (hb6279)

April 5, 2013

1 § 11.1

1.1 2a[1]

For the boundary value problem

$$y'' = y' + 2y + \cos x, \quad 0 \leqslant x \leqslant 1, \quad y(0) = -0.3, \quad y(1) = -0.1 \tag{1}$$

find an approximate solution using one step of Secant-Euler with $N = 2$, $t_0 = \frac{\beta - \alpha}{b - a}$, and $t_1 = 1.1t_0$. The actual solution is has the solution $y(x) = -\frac{1}{10}(\sin x + 3\cos x)$. Compare the approximate solution associated with t_2 to the exact solution at each node.

Using the code specified in listings 1 and 2, I obtained the following data.

x	y_{approx}	y_{actual}
0	-0.30000	-0.30000
0	-0.25192	-0.24624
0	-0.10000	-0.10000

Table 1: Approximation of $y(x)$ for eq. (1)

1.2 9

Consider the boundary value problem

$$y'' + y = 0 \quad , 0 \leqslant x \leqslant b, \quad y(0) = 0, \quad y(b) = B. \tag{2}$$

Find choices for b and B so the boundary value problem has no solution, exactly one solution, or infinitely many solutions.

Equation (1) is a second-order linear differential equation, and can be solved as such. Assume $y = e^{rx}$, then we can see that

$$\begin{aligned} y'' + y &= 0 \\ r^2 e^{rx} + e^{rx} &= 0 \\ r^2 + 1 &= 0 \end{aligned}$$

Which implies that

$$y(x) = c_1 \cos x + c_2 \sin x \tag{3}$$

From eq. (3) we can evaluate the first boundary condition $y(0) = 0$.

$$\begin{aligned} y(x) &= c_1 \cos x + c_2 \sin x \\ y(0) &= c_1 \cos 0 + c_2 \sin 0 \\ \implies c_1 &= 0 \end{aligned} \tag{4}$$

Now we can find values of b and B for the second boundary condition for which the boundary value problem has the end results specified in eq. (4). We will first consider the case when there is no solution. Assuming $y(0) = 0$, consider the second condition to be $y(2\pi) = 3$. Then we will obtain the following.

$$\begin{aligned} y(2\pi) &= c_1 \cos 2\pi + c_2 \sin 2\pi \\ 3 &= c_1 \cos 2\pi + c_2 \sin 2\pi \\ \implies c_1 &= 3 \end{aligned} \tag{5}$$

We have already mentioned from eq. (4) that $c_1 = 0$, but we have obtained that $c_1 = 3$ from eq. (5). This discrepancy means that for the boundary condition $y(2\pi) = 3$ there is no solution for the system.

No Solution: $b = 2\pi,\ B = 3$

Now examine the case where there are infintely many solutions. Assuming $y(0) = 0$, consider the second condition to be $y(2\pi) = 0$. Then we will obtain the following.

$$\begin{aligned} y(2\pi) &= c_1 \cos 2\pi + c_2 \sin 2\pi \\ 0 &= c_1 \cos 2\pi + c_2 \sin 2\pi \\ \implies c_1 &= 0 \end{aligned} \tag{6}$$

We have already mentioned form eq. (4) that $c_1 = 0$, so for the boundary condition $y(2\pi) = 0$ there are infinitely many solutions since we only get the same information about c_1 from the computation involved in eq. (6).

Infinitely Many Solutions: $b = 2\pi,\ B = 0$

Finally, lets consider the case where there is a solution. If we consider the second condition to be $y(\pi) = 0$, then the following will result.

$$\begin{aligned} y(\pi) &= c_1 \cos\pi + c_2 \sin\pi \\ 0 &= c_1 \cos\pi + c_2 \sin\pi \\ \implies c_2 &= 0 \end{aligned} \tag{7}$$

Since we have a value for c_1 from eq. (4) and we have found a unique c_2 from eq. (7), then we are sure that there is only one solution to the boundary value problem which can satisfy both of these conditions.

One Solution:
$b = \pi,\ B = 0$

2 § 11.2

2.1 4a[2]

Find an approximate solution using one step of Secant-Euler with $N = 2$, $t_0 = \frac{\beta-\alpha}{b-a}$, $t_1 = 1.1t_0$ for eq. (8). The actual solution is $y(x) = (x+1)^{-1}$. Compare the approximate solution associated with t_2 to the exact solution at each node.

$$y'' = y^3 - y\,y', \quad 1 \leqslant x \leqslant 2, \quad y(1) = \frac{1}{2}, \quad y(2) = \frac{1}{3} \tag{8}$$

Using the code specified in listings 1 and 2, I obtained the following data in table 2.

x	y_{approx}	y_{actual}
1.0000	0.50000	0.5
1.5000	0.38690	0.4
2.0000	0.33333	0.3

Table 2: Approximation of $y(x)$ for eq. (8)

3 Programming Minilab

Consider the case...

Let $z = Ax + By + C$. Then $p = A$, $q = B$, $u = 0$, $v = 0$, $w = 0$. Solving y'', we obtain

$$y'' = \frac{(Ay' - B) * 0}{1 + A^2 + B^2} \quad = 0 \tag{9}$$

This implies

$$y' = K \implies y = Kx + J \tag{10}$$

So y is straight line since the second derivative of y is zero. Then z is a line since the projection of a line onto a plane is a line.

The output for part (c) is listed below in table 3 using the code written in listing 3.

x	y_{approx}
-3	-2
-2.7	-1.661
-2.4	-1.315
-2.1	-0.9471
-1.8	-0.5977
-1.5	-0.3355
-1.2	-0.1551
-0.9	-0.02526
-0.6	0.08581
-0.3	0.2049
0.0	0.3401
0.3	0.4813
0.6	0.6215
0.9	0.7653
1.2	0.9237
1.5	1.099
1.8	1.282
2.1	1.464
2.4	1.643
2.7	1.822
3	2

Table 3: Approximation of $y(x)$ for the programming minilab

4 Code

```
#!/usr/bin/octave
# Created by Hershal Bhave on 04/04/13
# For M368K HW8,  11.1 Number 2a, 4a
# Written in GNU Octave
#
# Description: Uses a Second-Order Euler's Method to solve the IVP
# given in f in n steps between a and b given initial conditions on y0
# and yp0.
#

function [x,y] = eulersivp(f,n,a,b,y0,yp0)

  h = (b-a)/n;
  x = (a:h:b)';
  z(1,1) = y0;
  z(1,2) = yp0;

  for i=1:n
    z(i+1,1) = z(i,1) + h*z(i,2);
    z(i+1,2) = z(i,2) + h*f(x(i),z(i,1),z(i,2));
  endfor

  y = z(:,1);

endfunction
```

Listing 1: `eulersivp.m`

```
#!/usr/bin/octave
# Created by Hershal Bhave on 04/04/13
# For M368K HW8,  11.1 Number 2a and 4a
# Written in GNU Octave
#
# Description: Uses Secant-Euler to approximate the solution to the
# second-order differential equation in f using n steps between a and
# b with values at a and b aa and bb respectively. Guesses for the
# slope are given in t0 and t1 and the number of iterations to guess
# another t in m.
#

function [x,y] = shootseceuler(f,n,a,b,aa,bb,t0,t1,m)

  [x,y] = eulersivp(f,n,a,b,aa,t0);
  ybo = y(n+1);
  tko = t0;
  tk = t1;
  [tmp,y] = eulersivp(f,n,a,b,aa,t1);

  for k=3:m
    ttk = tk;
    tyb = y(n+1);
    tk = tk- ((y(n+1)-bb)*(tk-tko))/(y(n+1) - ybo);
    [tmp,y] = eulersivp(f,n,a,b,aa,tk);

    tko = ttk;
    ybo = tyb;
  endfor

endfunction
```

Listing 2: `shootsecular.m`

```
/*****************************************************
Program 9. Uses the Newton-RK4 shooting method to find
an approximate solution of a two-point BVP of the form

            y'' = f(x,y,y'), a<=x<=b
            y(a) = alpha, y(b) = beta

Inputs:
  feval Function to evaluate f(x,y,y')
  a,b Interval params
  alpha,beta Boundary value params
  t Initial guess for y'(a)
  N Number of grid points for RK4
  maxIter,tol Control params for Newton

Outputs:
  iter Number of Newton steps taken
  t Approx value of y'(a)
  x Grid point vector: x(j)=a+jh, j=0...N
  y Approx soln vector: y(j)=soln at x(j), j=0...N

Note 1: For any given problem, the function feval must
be changed. This function computes f for any given x, y
and y'. (Below we use the symbol yp in place of y').

Note 2: For any given problem the parameters a, b, alpha,
beta, t, N, maxIter and tol must also be specified.

Note 3: To compile this program use the command (all on
one line)

  c++ -o program9 matrix.cpp shootRK4.cpp program9.cpp

*****************************************************/
#include <iostream>
#include <iomanip>
#include <stdlib.h>
#include <math.h>
#include "matrix.h"
using namespace std;

/*** Declare external function ***/
int shootRK4(int&, double&, double&, double&, double&,
              double&, int&, double&, vector&, vector&) ;

double p(double x,double y);
double q(double x, double y);
double u(double x, double y);
double v(double x, double y);
double w(double x, double y);

/*** Define f(x,y,yp) function ***/
void feval(const double& x, const double& y,
                            const double& yp, double& f){
  double pi=4.0*atan(1.0) ; //the number pi
  f = ((p(x,y)*yp - q(x,y))*(u(x,y) + 2*v(x,y)*yp + w(x,y)*yp*yp))/(1+pow(p(x,y),2) + pow(q(x,y),2));
}

double p(double x,double y) {
  return (-2*x-2)*exp(-pow(x+1,2) - pow(y+1,2)) + 0.5*(-2*x+2)*exp(-pow(x-1,2)-pow(y-1,2));
}
double q(double x, double y) {
  return p(y,x);
}
```

```
double u(double x, double y) {
  return -2*exp(-pow(x+1,2)-pow(y+1,2)) + pow(-2*x-2,2)*exp(-pow(x+1,2) - pow(y+1,2))
    - exp(-pow(x-1,2)-pow(y-1,2)) + 0.5*pow(-2*x+2,2)*exp(-pow(x-1,2)-pow(y-1,2));
}
double v(double x, double y) {
  return (-2*x-2)*(-2*y-2)*exp(-pow(x+1,2)-pow(y+1,2))
    + 0.5*(-2*x + 2)*(-2*y+2)*exp(-pow(x-1,2)-pow(y-1,2));
}
double w(double x, double y) {
  return u(y,x);
}

int main() {
  /*** Define problem parameters ***/
  double tol=1e-6 ;
  int N=20, maxIter=10, itFer ;
  double a=-3.0, b=3.0, alpha=-2.0, beta=2.0, t ;

  vector x(N+1), y(N+1) ;
  t=2.0 ; // initial guess of slope

  /*** Call Newton-RK4 method ***/
  cout << setprecision(8) ;
  iter=shootRK4(N,a,b,alpha,beta,t,maxIter,tol,x,y) ;

  /*** Print results to screen ***/
  cout << setprecision(4) ;
  cout << "Number of Newton iterations: " << iter << endl ;
  cout << "Approx solution: t = " << t << endl ;
  cout << "Approx solution: x_j, y_j = " << endl ;
  for(int j=0; j<N+1; j++){
    cout << "x =" << setw(6) << x(j) ;
    cout << " " ;
    cout << "y =" << setw(10) << y(j) ;
    cout << " " ;
    cout << endl;
  }

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

Listing 3: `program9.cpp`