MA2915 Linear and Numerical Methods
SECOND MATLAB ASSIGNMENT, 2003/2004

Deadline: 1700 on Monday 19 January 2004

Electronic submission required, see below.

This is the second of two Matlab assignments which forms part of your assessment for MA2915 Linear and Numerical Methods.

Description

This assignment is concerned with implementing various methods for approximately solving a second order ordinary differential equation of the form

$$-u''(x) + \sigma(x)u(x) = f(x), \quad 0 < x < 1, \tag{1}$$

subject to the following conditions.

1.
$$u(0) = 0, \quad u(1) = 0. \tag{2}$$
This is the two point boundary value problem.

2.
$$u(0) = 0, \quad u'(0) = \alpha_0. \tag{3}$$
This is an initial value problem.

For a specific problem the functions $\sigma : [0,1] \to \mathbb{R}$ and $f : [0,1] \to \mathbb{R}$ are specified and the aim in both cases is to approximately determine $u : [0,1] \to \mathbb{R}$ given that it satisfies equation (1) together with the boundary or initial conditions. The methods to be used are briefly as follows.

The two point boundary value problem

We consider (1) with the boundary conditions (2).

For an integer $n \geq 1$ define $h = 1/(n+1)$ and let $x_i = ih$, $i = 0, 1, \cdots, n+1$ denote $n+2$ equally spaced points in $[0,1]$. From the observation that

$$-u''(x_i) = \frac{-u(x_{i-1}) + 2u(x_i) - u(x_{i+1})}{h^2} + \mathcal{O}(h^2) \tag{4}$$

for sufficiently smooth u (in fact u has to be 4 times continuously differentiable) this suggests the difference equations

$$\frac{-U_{i-1} + 2U_i - U_{i+1}}{h^2} + \sigma(x_i)U_i = f(x_i), \quad i = 1, 2, \cdots, n, \quad U_0 = U_{n+1} = 0, \tag{5}$$

as an approximation to the differential equation (1) with U_i being an approximation to $u(x_i)$. We can re-write (5) as

$$-U_{i-1} + d_i U_i - U_{i+1} = h^2 f(x_i), \quad i = 1, 2, \cdots, n, \qquad U_0 = U_{n+1} = 0, \quad d_i = 2 + h^2\sigma(x_i) \tag{6}$$

which in matrix-vector form is

$$T\underline{U} = h^2 \underline{f} \tag{7}$$

where

$$T = \begin{pmatrix} d_1 & -1 & & & & \\ -1 & d_2 & -1 & & & \\ & -1 & d_3 & -1 & & \\ & & \ddots & \ddots & \ddots & \\ & & & -1 & d_{n-1} & -1 \\ & & & & -1 & d_n \end{pmatrix}, \quad \underline{U} = \begin{pmatrix} U_1 \\ U_2 \\ U_3 \\ \vdots \\ U_{n-1} \\ U_n \end{pmatrix}, \quad \underline{f} = \begin{pmatrix} f(x_1) \\ f(x_2) \\ f(x_3) \\ \vdots \\ f(x_{n-1}) \\ f(x_n) \end{pmatrix}. \tag{8}$$

This way of obtaining an approximation is referred to here as **scheme A**.

A more accurate scheme than the one just given is to replace (6) by

$$\frac{-U_{i-1} + 2U_i - U_{i+1}}{h^2} + \frac{\sigma(x_{i-1})U_{i-1} + 10\sigma(x_i)U_i + \sigma(x_{i+1})U_{i+1}}{12} = \frac{f(x_{i-1}) + 10 f(x_i) + f(x_{i+1})}{12}, \tag{9}$$

for $i = 1, 2, \cdots, n$, and with again $U_0 = U_{n+1} = 0$. This gives a slightly different tri-diagonal system for $U_1, U_2, \cdots, U_n$. This way of obtaining an approximation is referred to here as **scheme B**.

The initial value problem

We consider (1) with the initial conditions (3).

If we let $y_1 = u$ and $y_2 = u'$ then (1) gives a system of first order ODEs

$$\begin{pmatrix} y_1 \\ y_2 \end{pmatrix}' = \begin{pmatrix} y_2 \\ \sigma y_1 - f \end{pmatrix}. \tag{10}$$

Let $\underline{y} = (y_1, y_2)^T$. Then in vector form we have

$$\underline{y}' = \underline{g}(x, \underline{y}), \quad \underline{y}(0) = \begin{pmatrix} 0 \\ \alpha_0 \end{pmatrix}, \quad \text{where} \quad \underline{g}(x, \underline{y}) = \begin{pmatrix} y_2 \\ \sigma(x) y_1 - f(x) \end{pmatrix}. \tag{11}$$

In the following $\underline{Y}_i$ denotes an approximation for $\underline{y}(x_i)$ for $i = 1, 2, \cdots, n+1$. A popular initial value solver is the Runge Kutta order 4 scheme which involves the following.

$$\underline{Y}_{i+1} = \underline{Y}_i + \frac{1}{6}(\underline{k}_1 + 2\underline{k}_2 + 2\underline{k}_3 + \underline{k}_4), \quad i = 0, 1, 2, \cdots, n-1, \tag{12}$$

where

$$\begin{aligned} \underline{k}_1 &= h\underline{g}(x_i, \underline{Y}_i), \\ \underline{k}_2 &= h\underline{g}(x_i + 0.5h, \underline{Y}_i + 0.5\underline{k}_1), \\ \underline{k}_3 &= h\underline{g}(x_i + 0.5h, \underline{Y}_i + 0.5\underline{k}_2), \\ \underline{k}_4 &= h\underline{g}(x_i + h, \underline{Y}_i + \underline{k}_3). \end{aligned}$$

(Observe that here $\underline{k}_1$, $\underline{k}_2$, $\underline{k}_3$ and $\underline{k}_4$ are all column vectors of length 2.)

A shooting method

The numerical solution that we obtain with the Runge Kutta scheme just described generates $\underline{Y}_1, \underline{Y}_2, \cdots, \underline{Y}_{n+1}$ with $(\underline{Y}_{n+1})_1$ giving an approximation to $u(1)$. The value that we will get here depends on the initial value α_0 used. We have a an approximate solution to the boundary value problem if we can determine the value of α_0 which leads to $(\underline{Y}_{n+1})_1 = 0$ and this we can attempt to determine this using a Newton type iteration. This way of obtaining an approximation is referred to here as **scheme C**.

The Tasks

These are concerned with implementing schemes A and B to approximately solve the boundary value problem, with implementing the Runge Kutta method to approximately solve the initial value problem and with using your Runge Kutta scheme together with the Newton type solver that we give you below to implement the shooting method.

1. Create Matlab function files called `testsig.m` and `testf.m` containing respectively

   ```
   function y=testsig(x)
   y=1+x.^4;
   ```

   and

   ```
   function y=testf(x)
   y=exp(-x.^2).*sin(20*x);
   ```

2. Create a Matlab function file called `methA.m` with a first line of the form

   ```
   function [x,U] = methA(n,sig,f)
   ```

   This has 3 input arguments and 2 output argument. The first input argument `n` is such that `n+1` is the number of equal subdivisions into which $[0, 1]$ is divided and the second and third input arguments are strings which give the names of the function files describing the function $\sigma(x)$ and $f(x)$ in (1). The first output argument is the row vector of length `n+2` of the equally spaced points in $[0, 1]$. The second output argument U created is a row vector of length `n+2` containing the approximate solution using scheme A with `U(1)`=0 and `U(n+2)`=0, with `U(`k`)` being the approximation to $u(x_{k-1})$.

3. Create a Matlab function file called `methB.m` with a first line of the form

   ```
   function [x,U] = methB(n,sig,f)
   ```

   This has 3 input arguments and 2 output argument which are as described in 2 except that here the output argument U created should be done using scheme B.

4. Create a Matlab function file called `RK4.m` with a first line of the form

   ```
   function [x,U] = RK4(n,sig,f,alpha0)
   ```

to implement the Runge Kutta scheme. This has 4 input arguments the first 3 of which are as in 2 and 3. The 4th input argument is the value of α_0 to be used at $x = 0$ for $y_2(0) = u'(0)$ and the vector `U` generated is to be, as before, a row vector of length `n+2` containing the values approximating $(u(x_0), u(x_1), \cdots, u(x_{n+1}))$.

5. Create the function file given below which implements a Newton type iteration.

```
function [x,U] = methC(n,sig,f,alpha0)
% Shooting method routine for the second assignment
% On input: n is the number of equally spaced interior points in [0,1],
%           sig and f are the names of the function files in
%           -u''+sig(x)*u=f(x)
%           alpha0 is the derivative of u at 0 to start the iteration
% On output: x is a row vector of the n+2 equally spaced points from 0 to 1.
%            U is a row vector of length n+2 containing the approximations
%            corresponding to the boundary conditions u(0)=0, u(1)=0.

% set the parameters in the Newton type iteration
delta=sqrt(eps); tol=1e-12; nit=10;

% perform the newton type iteration
for it=1:nit
  [x,U]=RK4(n,sig,f,alpha0);
  [x,Ud]=RK4(n,sig,f,alpha0+delta);
  phi=U(1,n+2);
  phid=(Ud(1,n+2)-U(1,n+2))/delta;
  alpha0=alpha0-phi/phid;
  if abs(phi)<tol
    fprintf('The iteration has converged in %d iterations\n',it);
    return;
  end
end
fprintf('Newton type iteration has not converged in %d iterations\n',nit);
```

6. Create a Matlab script file called `astest2.m` which uses scheme A and scheme B with $n = 39$. Once both approximations have been obtained you should create a single plot which contains the graphs of x against U using both scheme A and scheme B as implemented by your files `methA.m` and `methB.m`. The plot should contain appropriate labelling and a title.

   `astest2.m` should also contain statements to test `RK4.m` with $n = 39$ and with the two different values of α_0 given by $\alpha_0 = 0.045$ and $\alpha_0 = 0.05$. You should create another plot which contains the graphs of both these approximate solutions.

   Now, using the results obtained with scheme A we have

$$u'(0) \approx \frac{U_1 - U_0}{h} = \frac{U_1}{h}$$

which can be used for the starting point `alpha0` for scheme C. By using this value for `alpha0` use scheme C as implemented by `methC.m` and compare the results of scheme C with scheme A and scheme C with scheme B.

A summary of the tasks

To do the complete assignment you need to create the **6** function m-files `testsig.m`, `testf.m`, `methA.m`, `methB.m`, `RK4.m`, `methC.m` and the script m-file `astest2.m`. Apart from the first two functions which can be just be two lines each as indicated in 1, and apart from the function file `methC.m` which you just need to copy, we expect that the other files contain suitable comments describing what they do and in each of these files **you should put your name in a comment line at the beginning of the file.** The only output that `astest2` needs to create when run is the suitably labelled plot of the approximate solutions using schemes A and B, the suitably labelled plot of the Runge Kutta solutions corresponding to $\alpha_0 = 0.045$ and $\alpha_0 = 0.05$, together with the numbers comparing the results of scheme C with schemes A and B.

What you have to submit

For the submission **we only require an electronic version of your work** and this can be done using either of the following ways.

By email

(i) Once you have the versions of `testsig.m`, `testf.m`, `methA.m`, `methB.m`, `RK4.m`, `methC.m` and `astest2.m` that you wish to submit you need to create a `zip` file which contains these files. You can do this by typing the following at the command line in a command window.

```
zip ma2915as2 testsig.m testf.m methA.m methB.m RK4.m methC.m astest2.m
```

If you are unable to complete everything then you include the files that you have completed.

(ii) You should send the `zip` file `ma2915as2.zip` as an attachment to `Mike.Warby@brunel.ac.uk` using your preferred mail program.

The reason for asking you to create the `zip` file here is that some mail programs corrupt attachments which only involve plain text.

You will receive a reply by email indicating whether or not the submission was successful. You should not consider the submission as complete until you receive a satisfactory reply.

By allowing us to copy the files across the network

The following steps will enable us to retrieve the final versions of `testsig.m`, `testf.m`, `methA.m`, `methB.m`, `RK4.m`, `methC.m` and `astest2.m` that you wish to submit and attempts to ensure that nobody else on the network can obtain useable versions of your work.

(i) Check that you have a `u:` drive. This should usually be mounted as part of your Matlab session. If it is not mounted then in a command window you will need to type

```
net use u: \\pinn.brunel.ac.uk\ma02xyz /user:ma02xyz
```

with `ma02xyz` replaced by your login id.

(ii) Login to `molnir` using putty. You get this by typing in a command window

```
putty molnir
```

You will need a `putty` window and a command window at the same time.

(iii) In the command window on windows you should type at the prompt

```
copy testsig.m u:\
copy testf.m u:\
copy methA.m u:\
copy methB.m u:\
copy RK4.m u:\
copy methC.m u:\
copy astest2.m u:\
```

If you get any error here then you have not done this successfully.

(iv) At the command prompt on `molnir` (i.e. at the prompt `%`) in the `putty` window you should type

`~icstmkw/bin/ma2915_as2` *your_8_character_encode_key*

This will adjust the permissions on your m-files to ensure that nobody else can copy these files and it will create encrypted versions of these files that we can copy and decode. Your 8 character key is as with the first assignment.

(v) To get an acknowledgement to confirm that everything has been done successfully you should send an email to `Mike.Warby@brunel.ac.uk` indicating that your files are ready to be collected. You will receive a reply indicating whether or not the files could be collected. You should not consider the submission as complete until you receive a satisfactory reply.

The marking scheme

This is the longer of the two Matlab assignments. The first assignment due on 24 November will be marked out of 30 with this assignment marked out of 70. The breakdown of these 70 marks is roughly as follows. There are about 20 marks associated with implementing scheme A, there are about 20 marks associated with implementing scheme B and there are about 20 marks associated with implementing the Runge Kutta method. Here implementing scheme A means creating `methA.m` and having sufficient lines in `astest2.m` to test it. Similar remarks apply to the marks associated with scheme B. The remaining 10 marks will be associated with correctly creating `methC.m` and with using it in `astest2.m` to get the results for scheme C to do the comparisons between the various schemes.

Some remarks, comments and tips

1. As has already been stated the files `testsig.m` and `testf.m` can be just 2 lines long. Also, to create `methC.m` you only need to put what is given in the listing given earlier.

2. Once `methA.m`, `methB.m` and `methC.m` have been created the script file `astest2.m` contain files of the form

```
...
n=39;
[x,UA]=methA(n,'testsig','testf');
[x,UB]=methB(n,'testsig','testf');

...
alpha0= the starting value for the iteration
[x,UC]=methC(n,'testsig','testf',alpha0);
```

   It is sufficient to compare the methods by just computing

```
ca=norm(UC-UA,inf);
cb=norm(UC-UB,inf);
```

   where `UA`, `UB` and `UC` are each vectors of length `n+2`.

3. In `methA.m` you can create the tridiagonal matrix T with the statements

```
h=1/(n+1);
xx=linspace(h,1-h,n);
sig=feval('testsig',xx)';
d=2+h^2*sig;
o=ones(n,1);
T=spdiags([-o d -o],-1:1,n,n);
```

   Once you have also set the vector `b` you can solve the linear equations with the statement

```
UU=T\b;
```

4. In `methB.m` you similarly have a tridiagonal system as in `methA.m` but with a different matrix and a different right hand side vector. If `a`, `d` and `b` are vectors of length `n` containing respectively the subdiagonal, diagonal and superdiagonal entries then the statement

```
T=spdiags([a d b],-1:1,n,n);
```

   creates the matrix.

   The subdiagonal and superdiagonal of course only have $n-1$ entries and in `methB.m` you need to know which $n-1$ values are used. This is indicated by the following set of lines together with the output that it creates.

```
n=5;
z=linspace(1,n,n)';
T=spdiags([-z 5*z z],-1:1,n,n)
FT=full(T)
whos
```

This produces the output given below.

```
T =
   (1,1)        5
   (2,1)       -1
   (1,2)        2
   (2,2)       10
   (3,2)       -2
   (2,3)        3
   (3,3)       15
   (4,3)       -3
   (3,4)        4
   (4,4)       20
   (5,4)       -4
   (4,5)        5
   (5,5)       25
FT =
     5     2     0     0     0
    -1    10     3     0     0
     0    -2    15     4     0
     0     0    -3    20     5
     0     0     0    -4    25
  Name      Size           Bytes  Class

  T         5x5              180  sparse array
  FT        5x5              200  double array
  n         1x1                8  double array
  z         5x1               40  double array

Grand total is 44 elements using 428 bytes
```

The sub-diagonal uses the first $n-1$ entries whilst the super-diagonal uses the last $n-1$ entries, i.e. positions $2, 3, \cdots, n$.

The tridiagonal matrix is scheme B is of the form

$$T = \begin{pmatrix} d_1 & a_2 & & & & \\ a_1 & d_2 & a_3 & & & \\ & a_2 & d_3 & a_4 & & \\ & & \ddots & \ddots & \ddots & \\ & & & a_{n-2} & d_{n-1} & a_n \\ & & & & a_{n-1} & d_n \end{pmatrix} \tag{13}$$

where

$$d_i = 2 + \frac{10}{12}h^2\sigma(x_i), \quad a_i = -1 + \frac{1}{12}h^2\sigma(x_i), \quad i = 1, 2, \cdots, n. \tag{14}$$

As a check that this has been created correctly you may wish to check that the $(i-1, i)$ and $(i+1, i)$ entries agree for $i = 2, 3, \cdots, n-1$.

5. As a tip to creating `RK4.m` you may wish to have statements of the form

```
function [x,U] = RK4(n,sig,f,alpha0)
...
... appropriate statements
...
  k2=h*g(x(j)+0.5*h,VV+0.5*k1,sig,f);
...
... appropriate statements
...

function y=g(x,u,sig,f)
y(2,1)=feval(sig,x)*u(1)-feval(f,x);
y(1,1)=u(2);
```

   Here the function `g` is an internal function of `RK4`.

6. If you want to check that your scripts are producing the correct results then there are few things you can do. First, as schemes A and B are approximately solving the same problem then are 2 approximations should be 'close' to each other. In your testing you can check that the different approximations are getting closer together as `n` increases. You could for example try `n=79` and `n=159` etc.. You can also test you program with different choices for σ and the right hand side function f. If you take $\sigma = 0$ and $f(x) = 2$ then the exact solution is $u(x) = x(1-x)$ and, up to rounding error, all these schemes give the exact solution for any $n \geq 1$. If you take $\sigma = 0$ and $f(x) = 6x(2x-1)$ then the exact solution is $u(x) = x^3(1-x)$ and, up to rounding error, scheme B gives the exact solution for any $n \geq 1$ but scheme A does not.