Math 3210 - Numerical Analysis

Homework #4 Due Dec 5

1. Consider the boundary value problem,

$$y'' - 9y' - 10y = 0\,, \quad y(0) = 1.0001\,, y(1) = 2.57052$$

   (a) Solve the above boundary value problem using linear shooting. For the time integration use rk2 method with time step $h = 0.01$.
   To use linear shooting, we must write the boundary value problem as two initial value problems. The initial value problem will be a second order problem. All the methods, we may use on initial value problems are for first order systems. So we must write our second order problem as a first order system. We let $u = y$ and $v = y'$ then u and v satisfy,

$$\begin{aligned} u' &= v\,, \\ v' &= 9v + 10u\,, \\ u(0) = y_0\,, &\quad v(0) = y'_0\,. \end{aligned}$$

   Here is the code I used to evaluate the right hand side of this equation,

```
function y=f2d(t,u)

% f expects u to be a 2-vector here.

   y=[u(2) 9*u(2)+10*u(1)];

end
```

   This function returns a vector. Now we need to modify the midpoint method to work with a system of equations. Here is my modified code,

```
function y=rk22(f,a,b,y0,h)

     n=(b-a)/h+1;

     y=zeros(n,2);
     t=a;
     y(1,:)=y0;
     for i=2:n
       K1=h*feval(f,t,y(i-1,:));
       y(i,:)=y(i-1,:)+h*feval(f,t+h/2,y(i-1,:)+K1/2);
       t=t+h;
     end
end
```

   You will note that this code is almost unchanged from the original midpoint code. The main difference is now the solution is a $2 \times n$ array. The values $y(:,1)$ are for u and $y(:,2)$ are for v. Now all we need to do is to call the Runge-Kutta code with the right initial values and combine the two solutions as in the text.

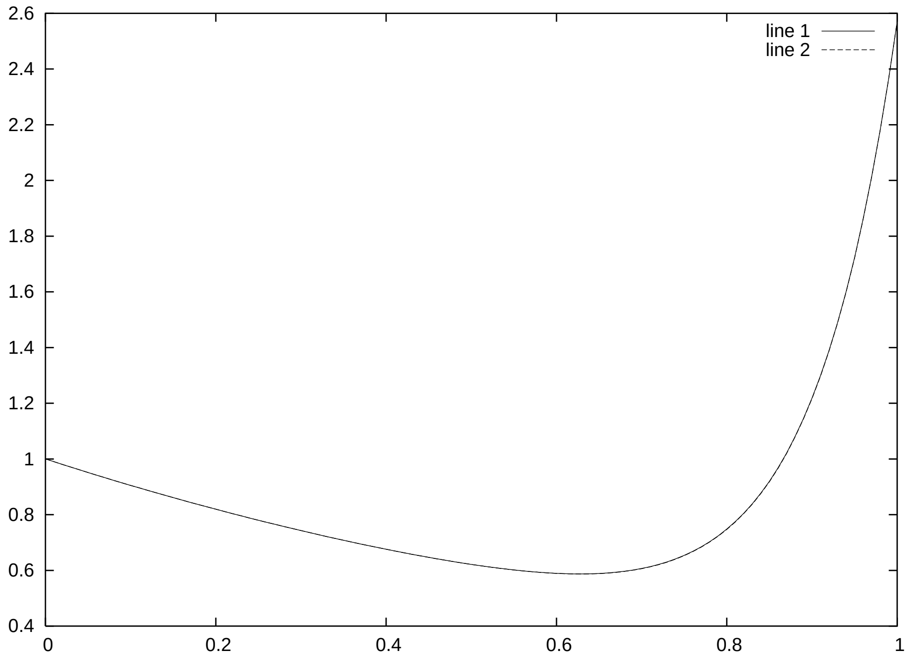
```
function y=shoot(f,a,b,alp,bet,h)

     alp0=[alp 0];

     y1=rk22(f,a,b,alp0,h);

     alp1=[0 1];

     y2=rk22(f,a,b,alp1,h);
     [n m]=size(y2)

     c=(bet-y1(n,1))/y2(n,1);

     y=y1+c*y2;

     end
```

I will give the plots of the solutions in part (c).

(b) Solve the above boundary value problem using finite differences. Use step size $h = 0.01$. For this method, all we need to do is construct the matrix and here I will let Matlab invert it. To be efficient, a tridiagonal solver should be used, but we can be a little sloppy here.

```
function y=findiff(p,q,r,a,b,alp,bet,h)

     % we are only solving for w_1 to w_{n-1}, add in w_0 and w_N later.
     N=(b-a)/h-1

     A=zeros(N);
     b=zeros(N,1);
     x=a+h;

     A(1,1)=2+h^2*feval(q,x);
     A(1,2)=-1+h/2*feval(p,x);

     b(1)=-h^2*feval(r,x)+(1+h/2*feval(p,x))*alp;

     for i=2:(N-1)
       x=a+i*h;
       A(i,i)=2+h^2*feval(q,x);
       A(i,i-1)=-1-h/2*feval(p,x);
       A(i,i+1)=-1+h/2*feval(p,x);

       b(i)=-h^2*feval(r,x);
     end

       x=b-h;
     A(N,N)=2+h^2*feval(q,x);
     A(N,N-1)=-1-h/2*feval(p,x);

     b(N)=-h^2*feval(r,x)+(1-h/2*feval(p,x))*bet;

y=A\b;
% add in the boundary conditions
y=[alp ; y ; bet];

end


function y=p1(x)

  y=9;

end

function y=q1(x)

y=10;

end

function y=r1(x)

y=0;

end
```

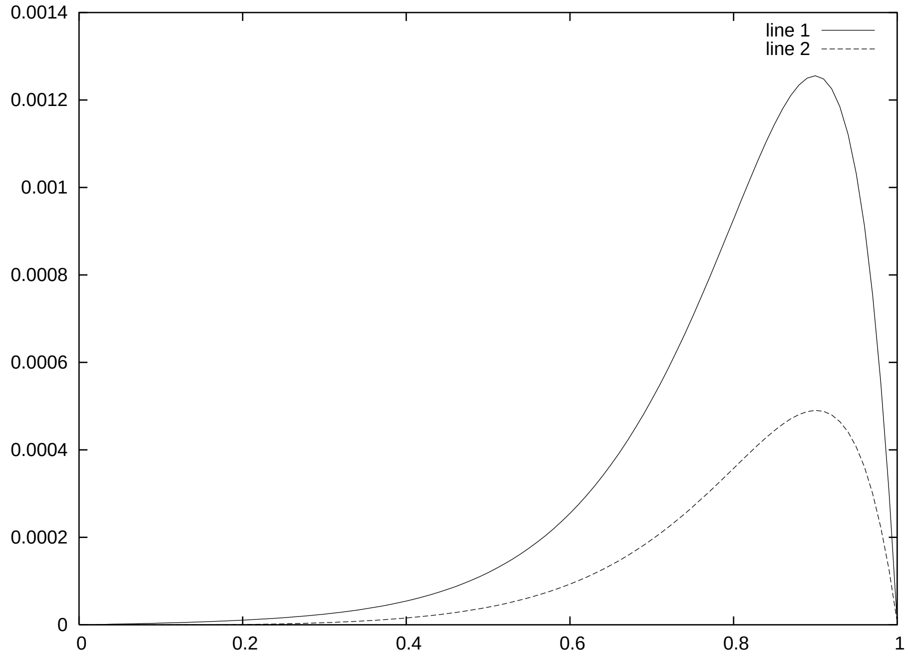
(c) Plot both solutions. Comment on the two solutions, which do think provides a more accurate approximation.
Here are both solutions on the same graph.



The two curves are almost indistinguishable. We might expect the finite difference approximation to be better because of stability. For this problem, we can find the exact solution,

$$y(x) = e^{-x} + 0.0001e^{10x}\,.$$

Now we plot the absolute error for both solutions,



The smaller curve corresponds to the finite difference method. As you can see the maximum error in this case is about $\frac{1}{3}$ that of the shooting maximum error. So in this case finite difference provides a better approximation for the same step size.
Here is the octave commands I used to make the above plots.

```
octave:2> y1=shoot('f2d',0,1,1.0001,2.57052,.01);
```

```
n = 101
m = 2
octave:3> y2=findiff('p1','q1','r1',0,1,1.0001,2.57052,0.01);
N = 101
octave:4> x=0:.01:1;
octave:5> sol=exp(-x)+0.0001*exp(10*x);
octave:9> plot(x,y1(:,1))
octave:7> hold on
octave:8> plot(x,y2)
octave:9> hold off
octave:10> plot(x,abs(y1(:,1)-sol'))
octave:11> hold on
octave:12> plot(x,abs(y2-sol'))
octave:13> hold off
octave:14> quit
```