A variable time step method

We construct and test a variable time step method. We will use Euler's method and the Midpoint method for our two approximations. This code bounds the relative error. The derivation in the text is for the absolute error. The code for our implementation is given below:

```matlab
function [t,y]=varh1(f,y0,t0,tf,tol)
    t(1)=t0;
    h=0.001;
    y(1,:)=y0;
    i=1;
    while(t(i)<tf)
        k1=h*feval(f,t(i),y(i,:));
        k2=h*feval(f,t(i)+h/2,y(i,:)+k1/2);
        y1=y(i,:)+k1;
        y2=y(i,:)+k2;
        err=norm(y2-y1);
        if (err/norm(y(i,:))<tol)
            y(i+1,:)=y2;
            t(i+1)=t(i)+h;
            i=i+1;
        end
        h=.8*(tol*norm(y(i,:))/err)^.5*h;
    end
    t=t';
end
```

In the above program, \( y_1 \) and \( y_2 \) are the \( O(h) \) and \( O(h^2) \) approximations respectively. We use the `norm` command since these quantities may be vectors and `norm` returns the standard Euclidean norm. Notice the `if` block. This set of instruction is executed only if the relative error is less then the tolerance. So a time-step is accepted only if it is good enough. In any case we find a new time-step. The last statement is to make the time vector a column vector. This make plotting easier.

We can test our code on the Lotka-Volterra system:

```matlab
function yp=volt(t,y)
    a=4;
    c=1;
    yp(1)=a*(y(1)-y(1)*y(2));
    yp(2)=-c*(y(2)-y(1)*y(2));
end
```

Here is the matlab session:

```
octave:2> [t,y]=varh1('volt',[3,1],0,10,.001);
octave:3> plot(t,y,'x')
octave:4> print("vartimesteptest.eps")
octave:5> size(t)
an =
    462   1

octave:6> [t,y]=varh1('volt',[3,1],0,10,.0001);
octave:7> size(t)
an =
    1456   1
```

octave:8> diary off
When plotting the solution, I chose to plot an $x$ at each point. This allows us to see what the time steps are. As you can see, there are regions which appear denser. In these regions, the time step is reduced to capture the behaviour of the system. At the end of the session I check the number of time steps used then reran the program with a smaller tolerance. As you can see that resulted many more points and thus a much smaller time step.

![Figure 1: The numerical solution to the Lotka-Volterra system using a variable time step method.](image-url)