https://www.mathstat.dal.ca/~iron/math3210/backwardeuler.pdf

Implementation of Backward Euler Method Solving the Nonlinear System using Newtons Method.

As I showed in class the Backward Euler method has better stability properties than the normal Euler method. Specifically errors won't grow when approximating the solution to problems with rapidly decaying solutions. The price to pay for this stability is a nonlinear system must be solved at each time step. We consider the differential equation

$$\dot{\vec{x}} = \vec{f}(\vec{x})\,,$$

where $\vec{x} \in \mathbb{R}^n$ and $\vec{f} : \mathbb{R}^n \to \mathbb{R}^n$.

We will use Newton's method to approximate a solution at each time step. Recall that when solving $\vec{f}(\vec{x}) = 0$, the Newton iterations are given by

$$\vec{x}_{i+1} = \vec{x}_i \overset{-}{\cancel{+}} Df(\vec{x}_i)^{-1}\vec{f}(\vec{x}_i)\,, \longrightarrow \underbrace{x_{i+1} = x_i - DF(x_i)^{-1}\,F(x_i)}_{\text{Solve with the Newton's method}}$$

where $Df(\vec{x}_i)^{-1}$ is the inverse of the Jacobian of $\vec{f}$ evaluated at $\vec{x}_i$. For the Backward Euler method, we must solve

$$\vec{x}_{i+1} = \vec{x}_i + h\vec{f}(\vec{x}_{i+1})\,,$$

for $\vec{x}_{i+1}$ or

$$\vec{F}(\vec{x}_{i+1}) = \vec{x}_{i+1} - \vec{x}_i - h\vec{f}(\vec{x}_{i+1}) = 0\,.$$

The Jacobian of $\vec{F}$ is then given by,

$$DF(\vec{x}) = I - hDf(\vec{x})\,.$$

Putting all these ideas together we have the following Backward Euler code.

```
function [t,y]=beul(f,df,y0,t0,tf,n)

    h=(tf-t0)/n;
    t=linspace(t0,tf,n+1);
    y=zeros(n+1,length(y0));
    y(1,:)=y0;
    for i=1:n
        x0=y(i,:)';
        x1=x0-inv(eye(length(y0))-h*feval(df,t(i),x0))*(x0-h*feval(f,t(i),x0)'-y(i,:)');
        while (norm(x1-x0)>0.0001)
                                                                          F(x0)
            x0=x1;                  DF(x0)^-1
            x1=x0-inv(eye(length(y0))-h*feval(df,t(i),x0))*(x0-h*feval(f,t(i),x0)'-y(i,:)');
        end                 I         - h Df(x0)
        y(i+1,:)=x1';
    end
    end
```

The command `eye(lenght(y0))` results in an identity matrix of the correct size for our system. To test the code I ran the simulation of the Lottka-Volterra system we used earlier. Here are the two subroutines to evaluate $\vec{f}$ and Df:

```
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

function y=dvol(t,x)

    a=4;
    c=1;
    y(1,1)=a*(1-x(2));
    y(1,2)=-a*x(1);
    y(2,1)=c*x(2);
    y(2,2)=-c*(1-x(1));
```

Math 3210
Introduction to Numerical Computing
Dalhousie University

Instructor: David Iron
Email: iron@mathstat.dal.ca

Here is the session:

```
octave:1> [t,y]=beul('volt','dvol',[2,1],0,10,1000);
octave:2> plot(t,y)
octave:3> print("beulout.eps")
```

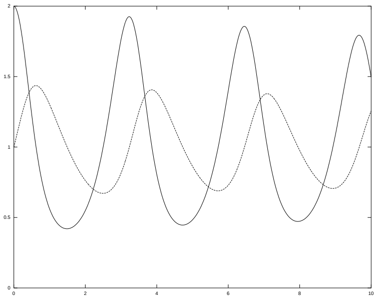

Figure 1: The Backward Euler approximation to the Lotka-Volterra system

It is important to note that implicit methods such as this one require significantly more computational effort than explicit methods. They should only be used when required.