Numerical Methods in Octave

Justin M. Ryan

In this notebook I show how to perform Euler's method, Imrpoved Euler's method, and the Runge-Kutta method to solve first order initial value problems in `Octave`.

`Octave` is free mathematical software that is designed to be very similar to `MATLAB`. The syntax is exactly the same, and most of the functions are the same. You can download `Octave` for free at `http://www.gnu.org` (https://www.gnu.org/software/octave/).

We begin by defining the algorithms. Each algorithm takes in an anonymous function handle representing the right-hand side of a first order differential equation, an initial t-value, an initial y-value, a step-size h, and a positive integer N representing the number of step.

In [1]:
```
% Euler's Method
function [tt, yy] = euler(f,t0,y0,h,N)
    k = N+1;
    tt = zeros(k,1);
    yy = zeros(k,1);
    tt(1) = t0;
    yy(1) = y0;
    for i = 2:k
        tt(i) = tt(i-1) + h;
        m = f(tt(i-1),yy(i-1));
        yy(i) = yy(i-1) + h*m;
    end
end
```

In [2]:
```
% Improved Euler's Method
function [tt, yy] = improved_euler(f,t0,y0,h,N)
    k = N+1;
    tt = zeros(k,1);
    yy = zeros(k,1);
    tt(1) = t0;
    yy(1) = y0;
    for i = 2:k
        tt(i) = tt(i-1) + h;
        m1 = f(tt(i-1),yy(i-1));
        m2 = f(tt(i),yy(i-1) + h*m1);
        yy(i) = yy(i-1) + h*(m1 + m2)/2;
    end
end
```

In [3]:
```
% Runge-Kutta Method
function [tt, yy] = runge_kutta(f,t0,y0,h,N)
    k = N+1;
    tt = zeros(k,1);
    yy = zeros(k,1);
    tt(1) = t0;
    yy(1) = y0;
    for i = 2:k
        tt(i) = tt(i-1) + h;
        m1 = f(tt(i-1),yy(i-1));
        m2 = f(tt(i-1) + (h/2),yy(i-1) + (h/2)*m1);
        m3 = f(tt(i-1) + (h/2),yy(i-1) + (h/2)*m2);
        m4 = f(tt(i),yy(i-1) + h*m3);
        yy(i) = yy(i-1) + h*(m1 + 2*m2 + 2*m3 + m4)/6;
    end
end
```

We consider a first order differential equation to see the usage.

Consider the initial value problem,

$$\begin{cases} y' = \dfrac{1-2ty}{y^2}, \\ y(0) = 2. \end{cases}$$

The FEUT applies, so a solution exists in an interval containing the initial t-value.

The first step is to define the inputs.

In [4]:
```
f = @(t,y) (1 - 2.*t.*y)./(y.^2);
t0 = 0;
y0 = 2;
```

Now apply each of the algorithms to approximate $\varphi(2)$ with a step-size of $h = 0.1$.

In [5]:
```
h = 0.1;
T = 2;
N = ceil((T - t0)/h);
```

In [6]:
```
[tt ye] = euler(f,t0,y0,h,N);
```

Euler's method yields the points:

In [7]:
```
[tt ye]
```

```
ans =

   0.00000   2.00000
   0.10000   2.02500
   0.20000   2.03951
   0.30000   2.04394
   0.40000   2.03852
   0.50000   2.02334
   0.60000   1.99834
   0.70000   1.96333
   0.80000   1.91797
   0.90000   1.86173
   1.00000   1.79390
   1.10000   1.71349
   1.20000   1.61915
   1.30000   1.50907
   1.40000   1.38069
   1.50000   1.23035
   1.60000   1.05258
   1.70000   0.83882
   1.80000   0.57561
   1.90000   0.25201
   2.00000   0.31873
```

In [8]:
```
[tt yi] = improved_euler(f,t0,y0,h,N);
```

The Improved Euler method yeilds:

In [9]:
```
[tt yi]
```

```
ans =

   0.00000   2.00000
   0.10000   2.01975
   0.20000   2.02931
   0.30000   2.02893
   0.40000   2.01874
   0.50000   1.99869
   0.60000   1.96863
   0.70000   1.92822
   0.80000   1.87698
   0.90000   1.81419
   1.00000   1.73890
   1.10000   1.64975
   1.20000   1.54491
   1.30000   1.42177
   1.40000   1.27652
   1.50000   1.10340
   1.60000   0.89330
   1.70000   0.63407
   1.80000   0.38722
   1.90000   1.95738
   2.00000   1.77720
```

In [10]:
```
[tt, yrk] = runge_kutta(f,t0,y0,h,N);
```

And the Runge-Kutta method yields:

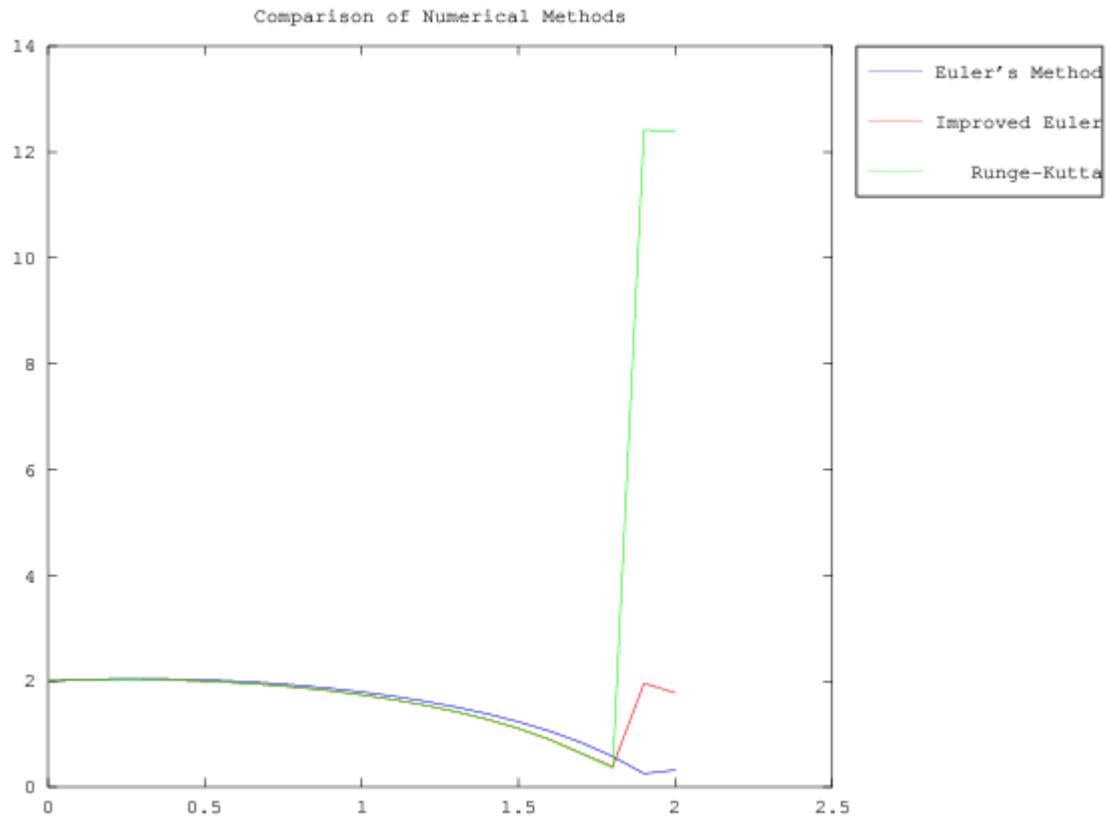
In [11]:
```
[tt yrk]
```

```
ans =

    0.00000    2.00000
    0.10000    2.01977
    0.20000    2.02934
    0.30000    2.02897
    0.40000    2.01879
    0.50000    1.99876
    0.60000    1.96871
    0.70000    1.92832
    0.80000    1.87710
    0.90000    1.81433
    1.00000    1.73906
    1.10000    1.64993
    1.20000    1.54513
    1.30000    1.42202
    1.40000    1.27682
    1.50000    1.10373
    1.60000    0.89360
    1.70000    0.63364
    1.80000    0.35630
    1.90000   12.41465
    2.00000   12.38385
```

We plot the graphs of each approximation on the same set of axes.

In [12]:
```
figure
    plot(tt,ye,'b')
    title("Comparison of Numerical Methods")
    hold on
    plot(tt,yi,'r')
    plot(tt,yrk,'g')
    legend("Euler's Method","Improved Euler","Runge-Kutta",
    'location','northeastoutside')
```



We see that the approximations behave similarly for t-values from 0 up until about $t = 1.8$. At this point, the approximation curves become erratic. This is due to the fact that the nonlinear FEUT only guarantees a solution to the IVP on "some interval" containing $t = 0$. For this example, it appears that the domain of definition of the solution curve does not extend past $t = 1.8$.

In order to better approximate that t-value at the end of the interval of definition, we could take smaller steps and keep an eye on where the smooth behavior of the curve appears to break down.