WATERLOO ENGINEERING

4^{th}-order Runge Kutta and the Dormand-Prince Methods

Douglas Wilhelm Harder, M.Math. LEL
Department of Electrical and Computer Engineering
University of Waterloo
Waterloo, Ontario, Canada

ece.uwaterloo.ca
dwharder@alumni.uwaterloo.ca

Outline

This topic discusses advanced numerical solutions to initial value problems:

- Weighted averages and integration techniques
- Runge-Kutta methods
- 4^{th}-order Runge Kutta
- Adaptive methods
- The Dormand-Prince method
  - The Matlab `ode45` function

Outcomes Based Learning Objectives

By the end of this laboratory, you will:

- Understand the 4th-order Runge-Kutta method
- Comprehend why adaptive methods are required to reduce the error but also reduce the effort
- Understand the algorithm for the Dormand-Prince method

Weighted Averages

The average of five numbers x_1, x_2, x_3, x_4, and x_5 is:

$$\bar{x} = \frac{x_1 + x_2 + x_3 + x_4 + x_5}{5} = \tfrac{1}{5}x_1 + \tfrac{1}{5}x_2 + \tfrac{1}{5}x_3 + \tfrac{1}{5}x_4 + \tfrac{1}{5}x_5$$

Suppose these were project grades and the last two projects had twice the weight of the other projects

- We can calculate the following *weighted average*:

$$\tfrac{1}{7}x_1 + \tfrac{1}{7}x_2 + \tfrac{1}{7}x_3 + \tfrac{2}{7}x_4 + \tfrac{2}{7}x_5$$

Weighted Averages

In fact, any combination scalars of a_1, a_2, a_3, a_4, and a_5 such that

$$a_1 + a_2 + a_3 + a_4 + a_5 = 1$$

allows us to calculate the weighted average

$$a_1 x_1 + a_2 x_2 + a_3 x_3 + a_4 x_4 + a_5 x_5$$

It is also possible to have negative weights:

$$\tfrac{1}{3} + \tfrac{1}{3} + \tfrac{1}{3} + \tfrac{1}{3} - \tfrac{1}{3} = 1$$

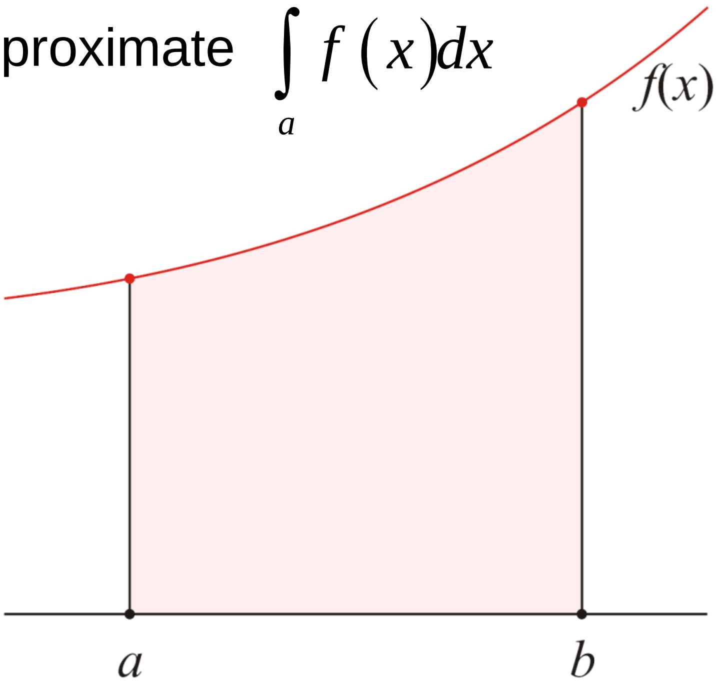
$$\tfrac{1}{3} x_1 + \tfrac{1}{3} x_2 + \tfrac{1}{3} x_3 + \tfrac{1}{3} x_4 - \tfrac{1}{3} x_5$$

Richardson extrapolation weights: $\frac{4}{3} - \frac{1}{3} = 1, \frac{16}{15} - \frac{1}{15} = 1$

Integration

We will motivate this next idea by looking at approximating integrals

- We wish to approximate $\int_a^b f(x)dx$

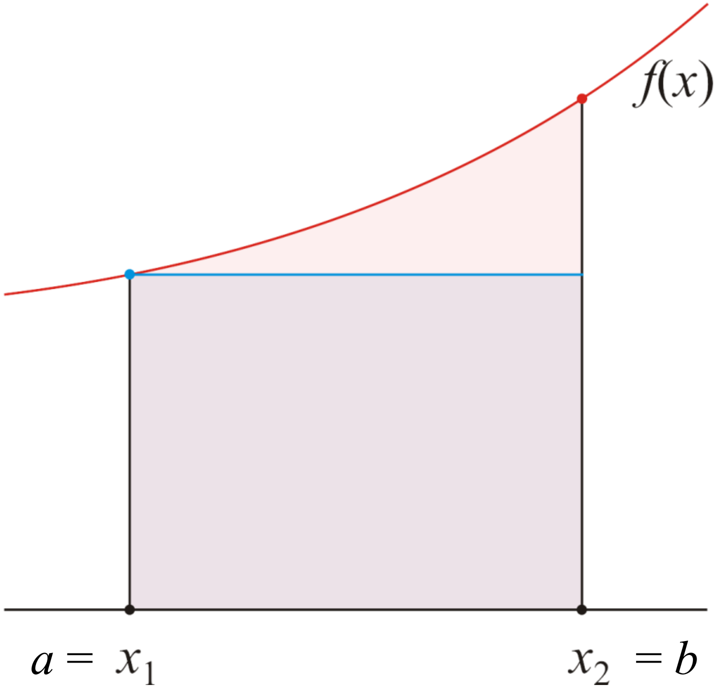


Integration

In first year, you would have seen the approximation:

- Approximate the integral by calculating the area of the square

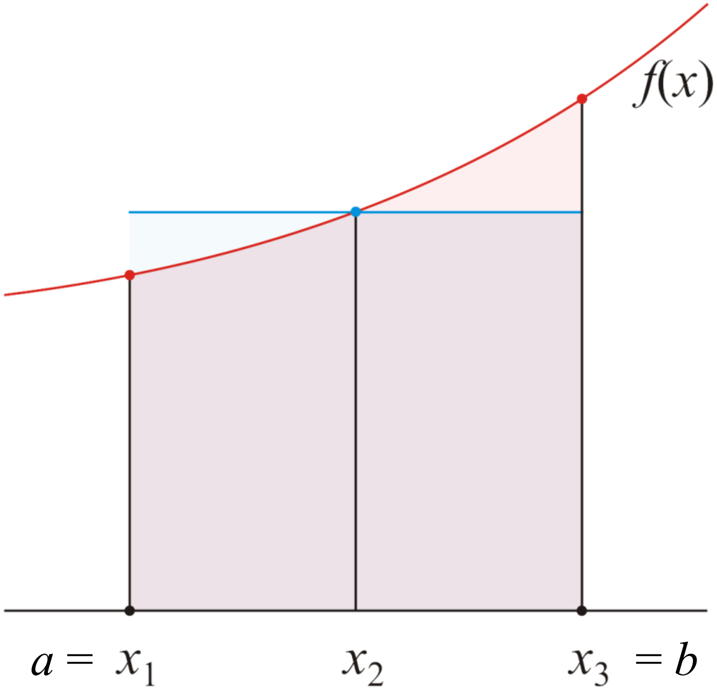
$$\int_a^b f(x)dx \approx f(x_1)(x_2 - x_1)$$



Integration

Alternatively, you could use the mid-point:

$$\int_a^b f(x)dx \approx f(x_2)(x_3 - x_1)$$

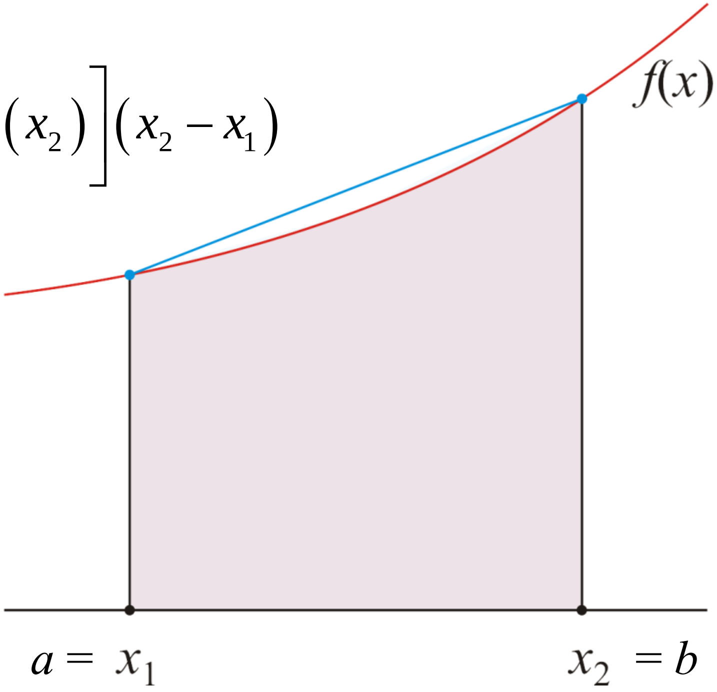


Integration

Or, take a weighted average of the two end points

- This weighted average calculates the area of the trapezoid

$$\int_a^b f(x)dx \approx \left[\frac{1}{2} f(x_1) + \frac{1}{2} f(x_2)\right](x_2 - x_1)$$



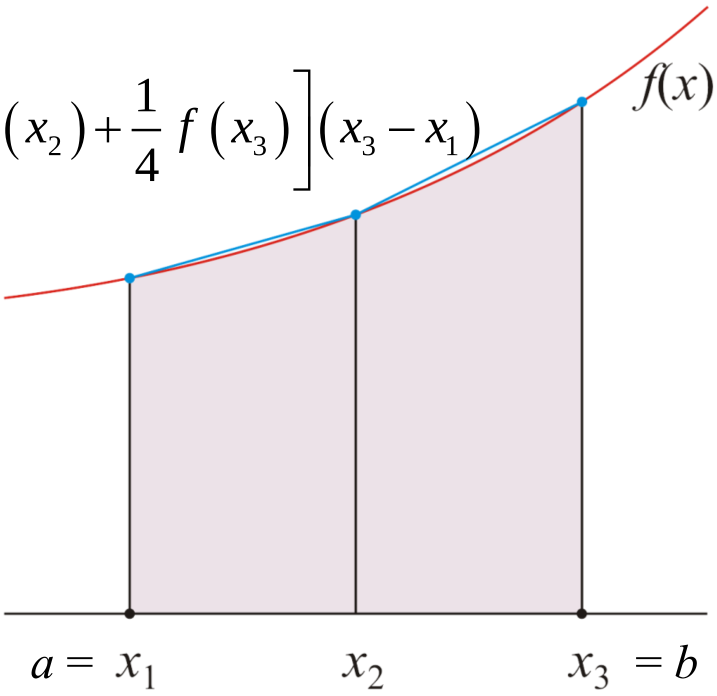
The *trapezoidal* rule

Integration

We could take a weighted average of three points

- This calculates the area of two trapezoids

$$\int_a^b f(x)dx \approx \left[\frac{1}{4}f(x_1) + \frac{1}{2}f(x_2) + \frac{1}{4}f(x_3)\right](x_3 - x_1)$$

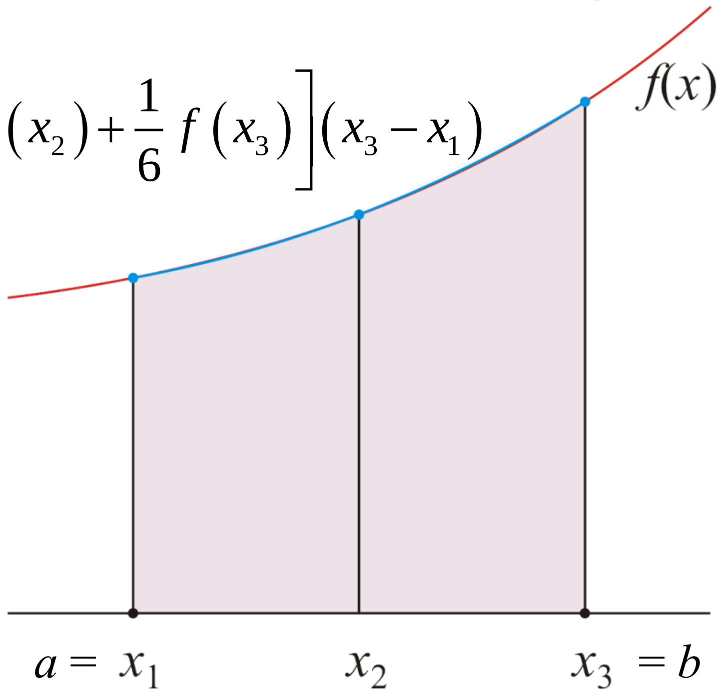


The *composite* trapezoidal rule

Integration

- A better approximation is to give more weight to the mid point
  - This calculates the area under the interpolating quadratic function

$$\int_a^b f(x)dx \approx \left[\frac{1}{6}f(x_1)+\frac{2}{3}f(x_2)+\frac{1}{6}f(x_3)\right](x_3-x_1)$$



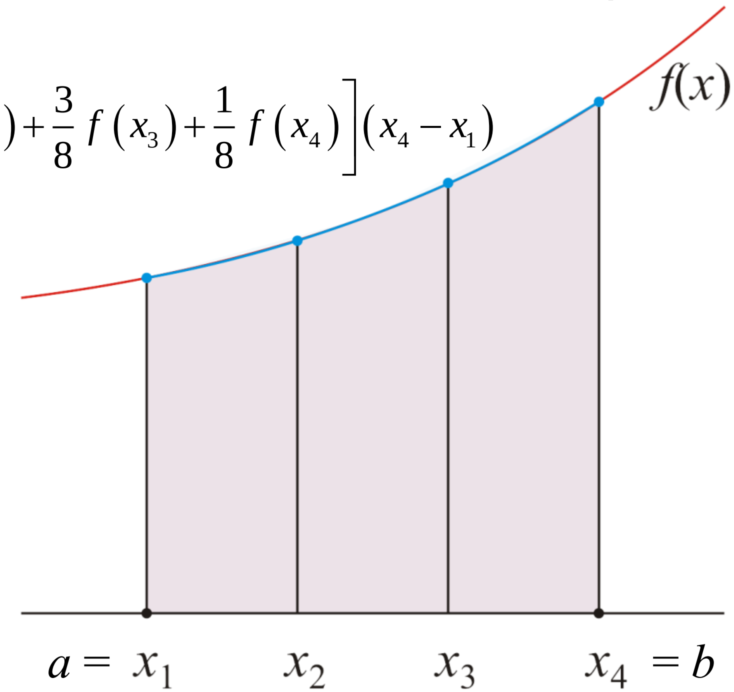
Simpson's rule

Integration

We can increase the number of points and use other weights

– This calculates the area under the interpolating quadratic function

$$\int_a^b f(x)dx \approx \left[\frac{1}{8}f(x_1)+\frac{3}{8}f(x_2)+\frac{3}{8}f(x_3)+\frac{1}{8}f(x_4)\right](x_4-x_1)$$



Simpson's $^3/_8$ rule

Initial-value Problems

We will use the same weighted average idea to find better approximations of an initial-value problem

In the last laboratory, we saw

- Euler's method
- Heun's method

In this laboratory, we will see:

- The 4^{th}-order Runge Kutta method
- The Dormand-Prince method

Both use weighted averages

Initial-value Problems

Recall that given a 1st-order ordinary-differential equation and an initial condition

$$y^{(1)}(t) = f\left(t, y(t)\right)$$

Then, given an initial condition

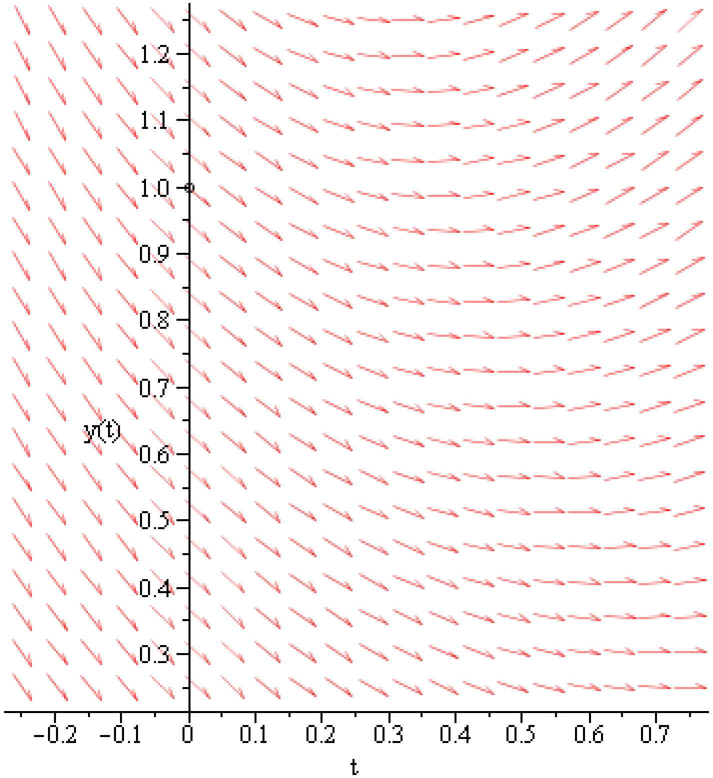
$$y(t_0) = y_0$$

we would like to approximate a solution

Initial-value Problems

For example, consider

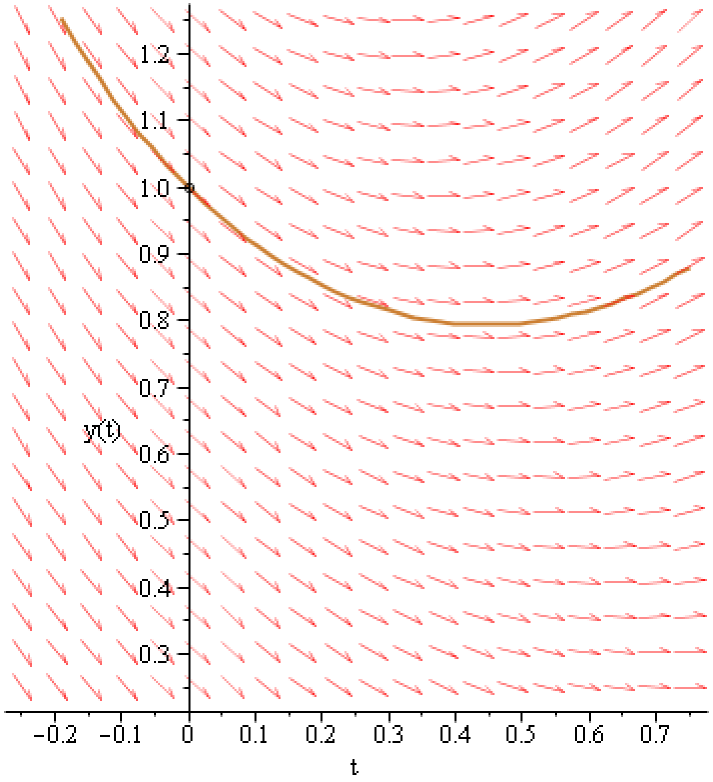
$$y^{(1)}(t) = y(t)(2-t)t + t - 1$$
$$y(0) = 1$$



Initial-value Problems

For example, consider

$$y^{(1)}(t) = y(t)(2-t)t + t - 1$$
$$y(0) = 1$$



Euler's Method

Euler's method approximates the slope by taking one sample: $K_1 = f\left(t_k, y_k\right)$

This slope is then used to approximate the next point:

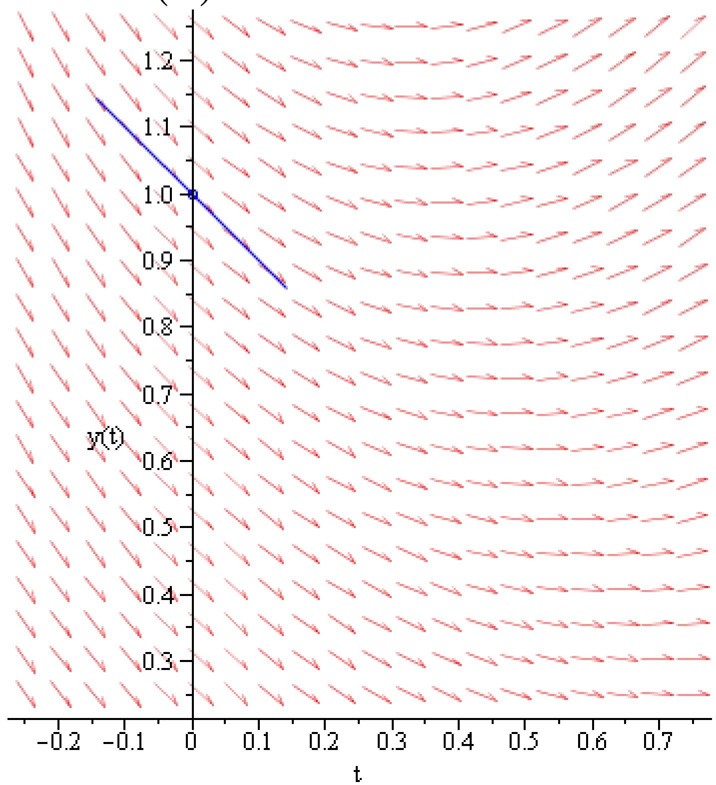
$$y_{k+1} = y_k + hK_1$$

Euler's Method

In our example, if $h = 0.5$, we would calculate this slope

$$y^{(1)}(t) = y(t)(2-t)t + t - 1$$

$$y(0) = 1$$



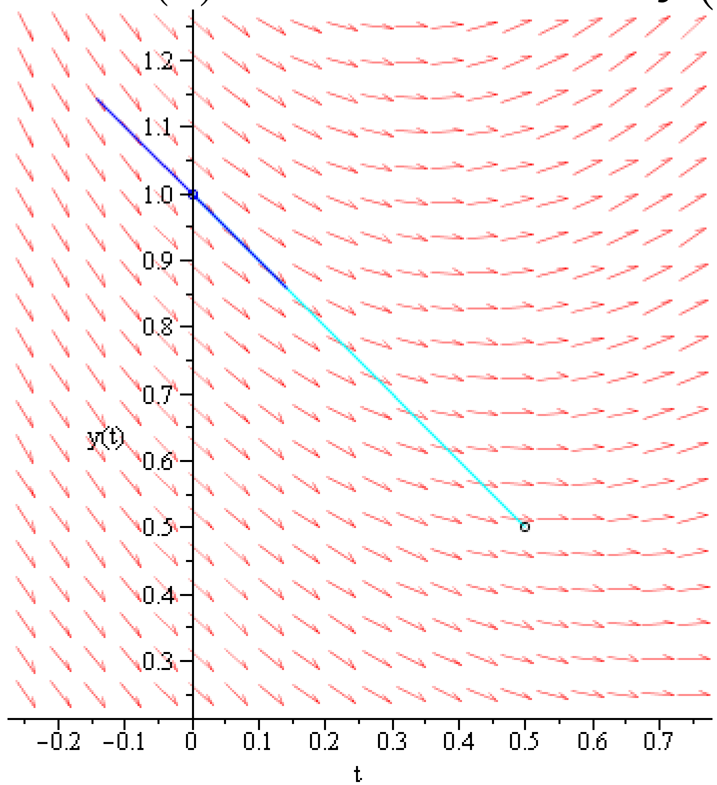
Euler's Method

We follow this slope a distance $h = 0.5$ out:

$$y^{(1)}(t) = y(t)(2-t)t + t - 1$$

$$y(0) = 1$$

$$y(0.5) \approx y_1 = y_0 + 0.5 \cdot (-1)$$
$$= 1 - 0.5 = 0.5$$

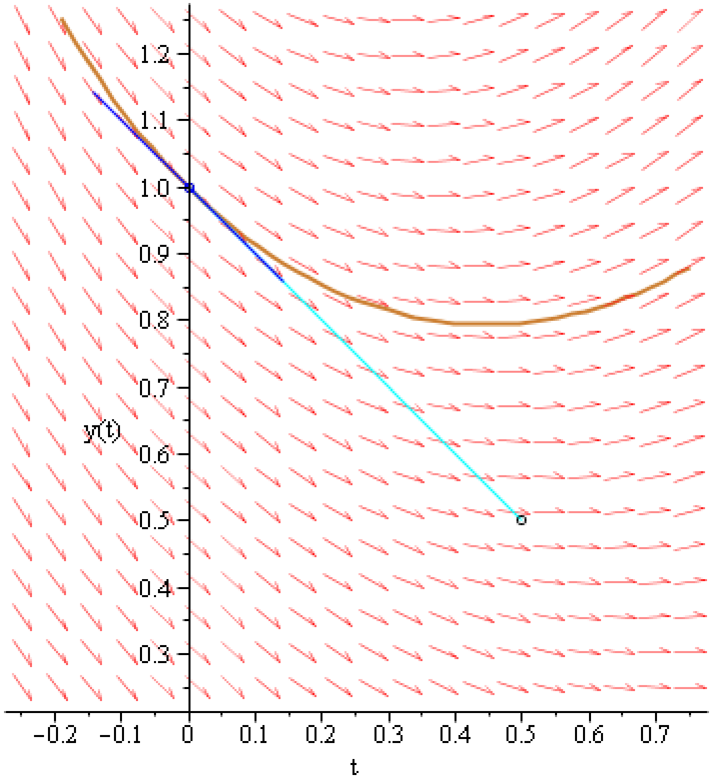


Euler’s Method

The approximation is not great if h is too large:

$$y^{(1)}(t) = y(t)(2-t)t + t - 1$$

$$y(0) = 1$$



Heun's Method

Heun's method approximates the slope by taking two samples:

$$K_1 = f\left(t_k, y_k\right)$$
$$K_2 = f\left(t_k + h, y_k + hK_1\right)$$

The average of the two slopes is used to approximate the next point:

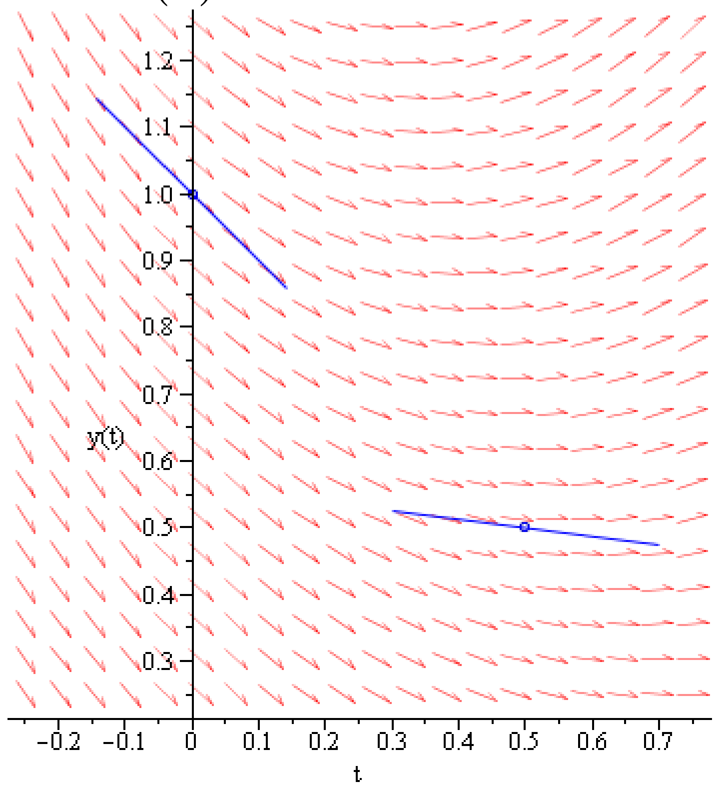
$$y_{k+1} = y_k + h\frac{K_1 + K_2}{2}$$

Heun’s Method

For Heun’s method, we calculate a second slope:

$$y^{(1)}(t) = y(t)(2-t)t + t - 1$$

$$y(0) = 1$$



$$K_1 = -1$$

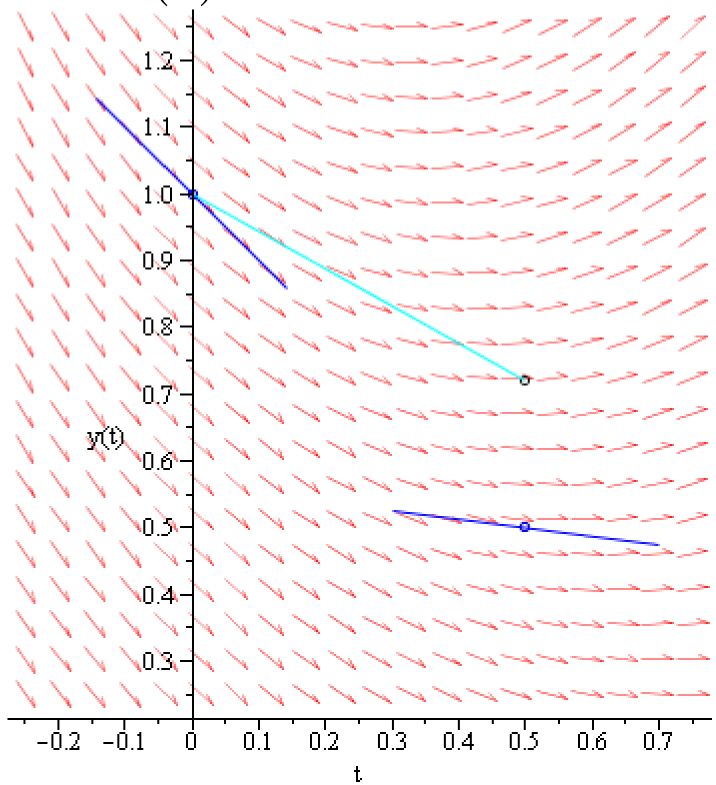
$$\begin{aligned} K_2 &= f(0.5, 0.5) \\ &= 0.5 \cdot 1.5 \cdot 0.5 + 0.5 - 1 \\ &= -0.125 \end{aligned}$$

Heun's Method

Take the average, and follow this average slope out:

$$y^{(1)}(t) = y(t)(2-t)t + t - 1$$

$$y(0) = 1$$



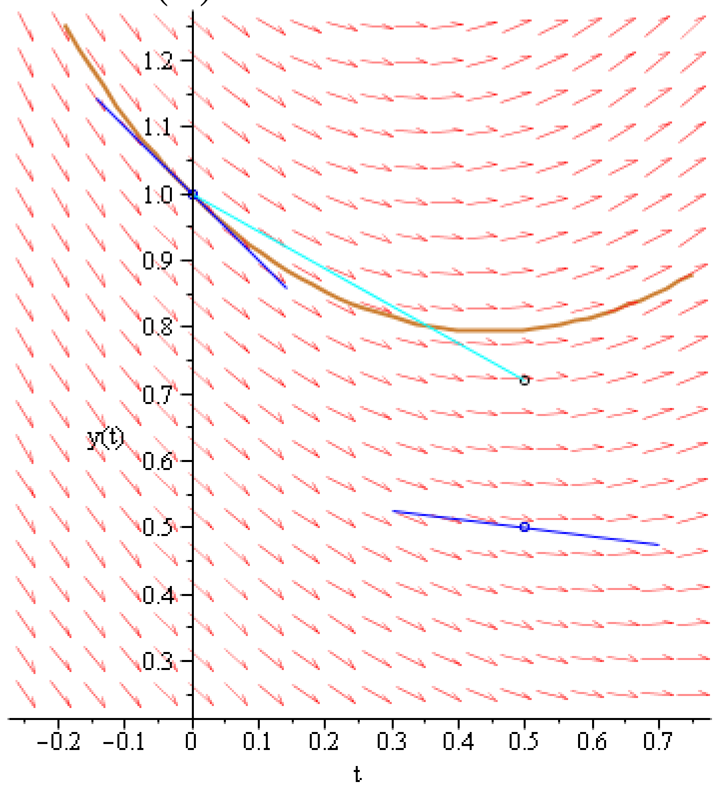
$$\frac{K_1 + K_1}{2} = \frac{-1 + (-0.125)}{2}$$
$$= -0.5625$$

$$y(0.5) \approx$$
$$y_1 = 1 + 0.5 \cdot (-0.5625)$$
$$= 0.71875$$

Heun's Method

The approximation is better than Euler's method

$$y^{(1)}(t) = y(t)(2-t)t + t - 1$$
$$y(0) = 1$$



$$\frac{K_1 + K_1}{2} = \frac{-1 + (-0.125)}{2}$$
$$= -0.5625$$

Mid-point Method

One idea we did not look at was the midpoint method:

– Use Euler's method to find in the slope in the middle with $h/2$:

$$K_1 = f\left(t_k, y_k\right)$$

$$K_2 = f\left(t_k + \left(\frac{1}{2}h\right), y_k + \left(\frac{1}{2}h\right)\left(K_1\right)\right)$$

This second slope is then used to approximate the next point:

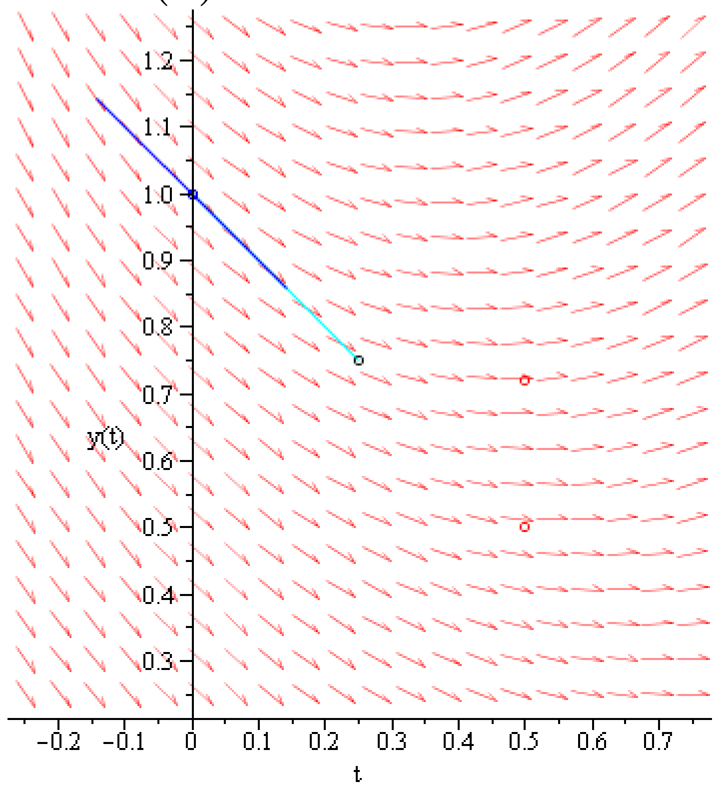
$$y_{k+1} = y_k + hK_2$$

Mid-point Method

Use Euler's method to find a point going out $h/2$:

$$y^{(1)}(t) = y(t)(2-t)t + t - 1$$

$$y(0) = 1$$



$$\begin{aligned} K_1 &= f(0,1) \\ K_2 &= f\left(0.25, 1 + 0.25(-1)\right) \\ &= f(0.25, 0.75) \\ &= -0.421875 \end{aligned}$$

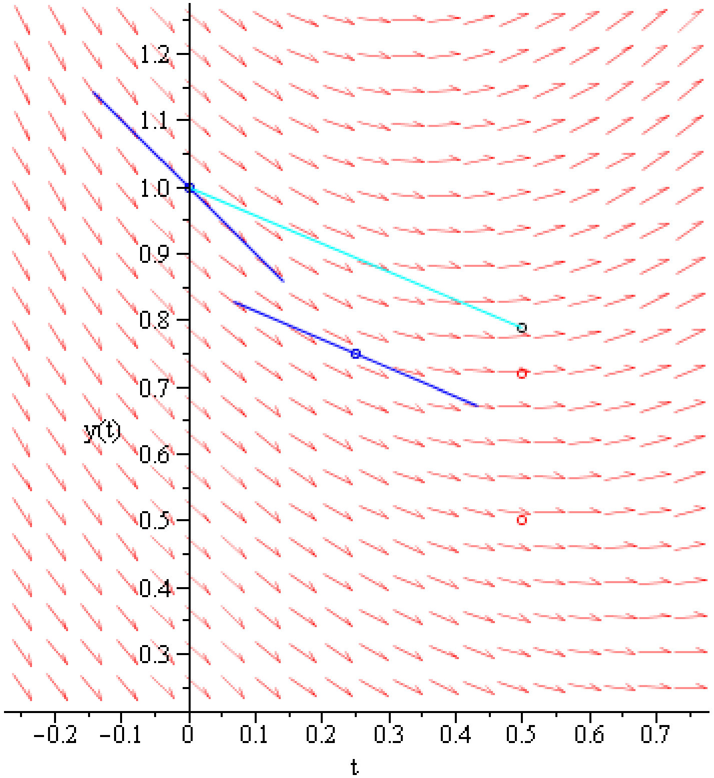
Mid-point Method

Calculate the slope and use this to approximate $y(0.5)$:

$$y^{(1)}(t) = y(t)(2-t)t + t - 1$$

$$y(0) = 1$$

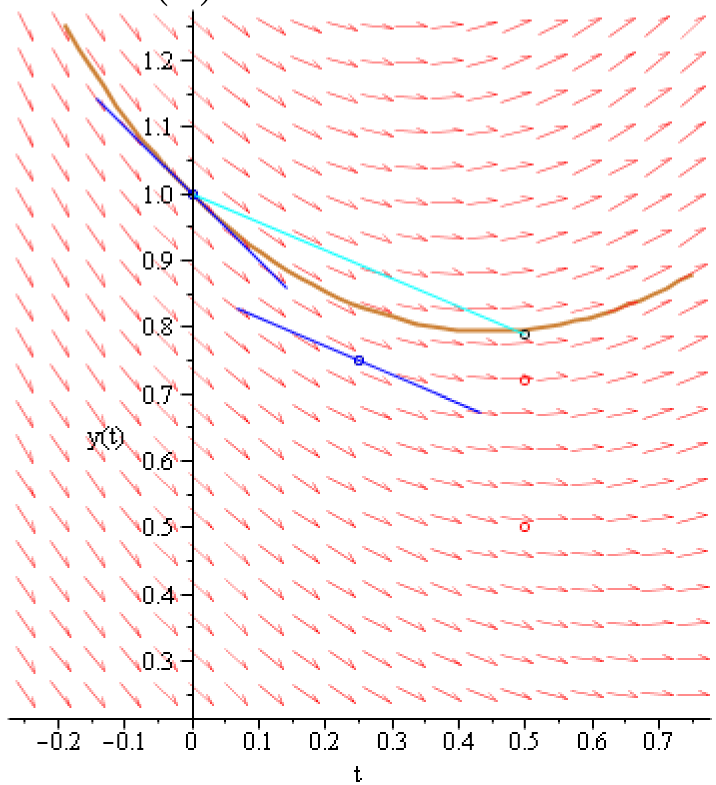
$$y(0.5) \approx y_1 = 1 + 0.5 \cdot (-0.421875)$$
$$= 0.7890625$$



Mid-point Method

The approximation is better than Heun's

$$y^{(1)}(t) = y(t)(2-t)t + t - 1$$
$$y(0) = 1$$



4th-order Runge-Kutta Method

The 4th-order Runge Kutta method is similar; again, starting at the midpoint $t_k + h/2$:

$$K_1 = f\left(t_k, y_k\right)$$

$$K_2 = f\left(t_k + \tfrac{1}{2}h, y_k + \tfrac{1}{2}h \cdot K_1\right)$$

4th-order Runge-Kutta Method

However, we then sample the mid-point again:

$$K_1 = f\left(t_k, y_k\right)$$
$$K_2 = f\left(t_k + \tfrac{1}{2}h, y_k + \tfrac{1}{2}h \cdot K_1\right)$$
$$K_3 = f\left(t_k + \tfrac{1}{2}h, y_k + \tfrac{1}{2}h \cdot K_2\right)$$

4th-order Runge-Kutta Method

We use this 3rd slope to find a point at $t_k + h$:

$$K_1 = f\left(t_k, y_k\right)$$
$$K_2 = f\left(t_k + \tfrac{1}{2}h, y_k + \tfrac{1}{2}h \cdot K_1\right)$$
$$K_3 = f\left(t_k + \tfrac{1}{2}h, y_k + \tfrac{1}{2}h \cdot K_2\right)$$
$$K_4 = f\left(t_k + h, y_k + h \cdot K_3\right)$$

4th-order Runge-Kutta Method

We then use a weighted average of these four slopes

$$K_1 = f\left(t_k, y_k\right)$$
$$K_2 = f\left(t_k + \tfrac{1}{2}h, y_k + \tfrac{1}{2}h \cdot K_1\right)$$
$$K_3 = f\left(t_k + \tfrac{1}{2}h, y_k + \tfrac{1}{2}h \cdot K_2\right)$$
$$K_4 = f\left(t_k + h, y_k + h \cdot K_3\right)$$

and approximate $y_{k+1} = y_k + h\left(\tfrac{1}{6}K_1 + \tfrac{1}{3}K_2 + \tfrac{1}{3}K_3 + \tfrac{1}{6}K_4\right)$

Compare with Heun's method:

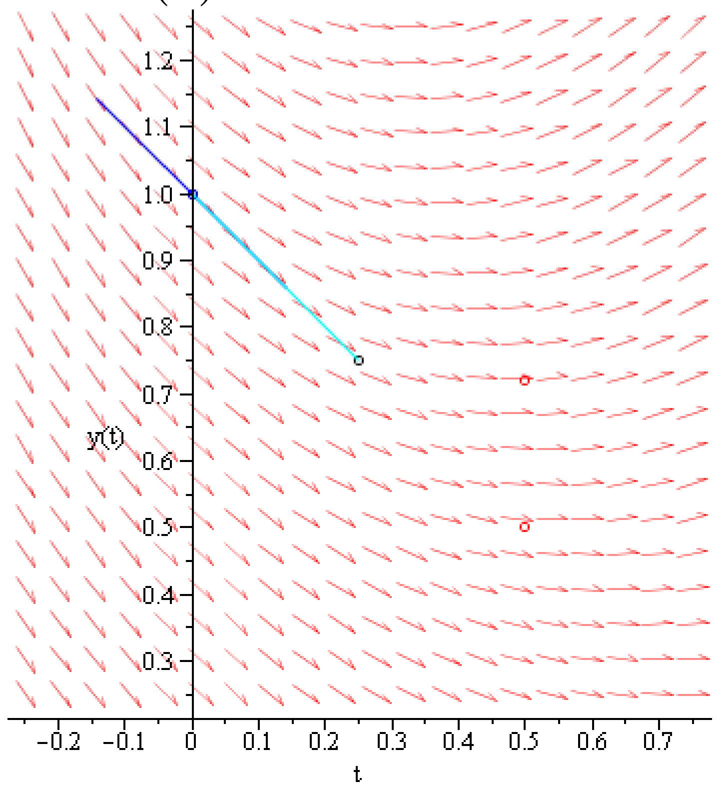
$$K_1 = f\left(t_k, y_k\right)$$
$$K_2 = f\left(t_k + h, y_k + hK_1\right) \qquad y_{k+1} = y_k + h\left(\tfrac{1}{2}K_1 + \tfrac{1}{2}K_2\right)$$

4th-order Runge-Kutta Method

Follow the slope to the mid-point

$$y^{(1)}(t) = y(t)(2-t)t + t - 1$$

$$y(0) = 1$$

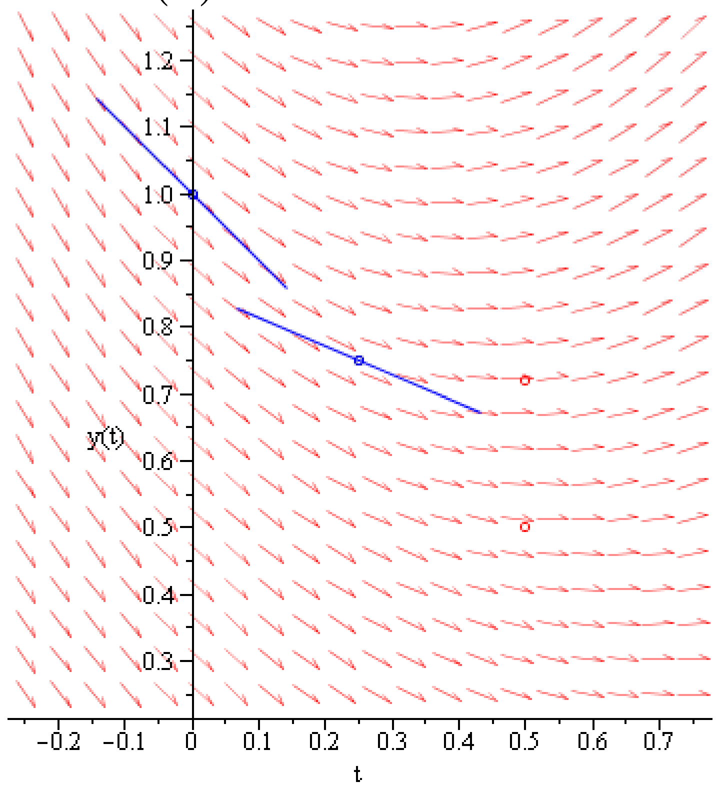


4th-order Runge-Kutta Method

Determine this slope, K_2:

$$y^{(1)}(t) = y(t)(2-t)t + t - 1$$

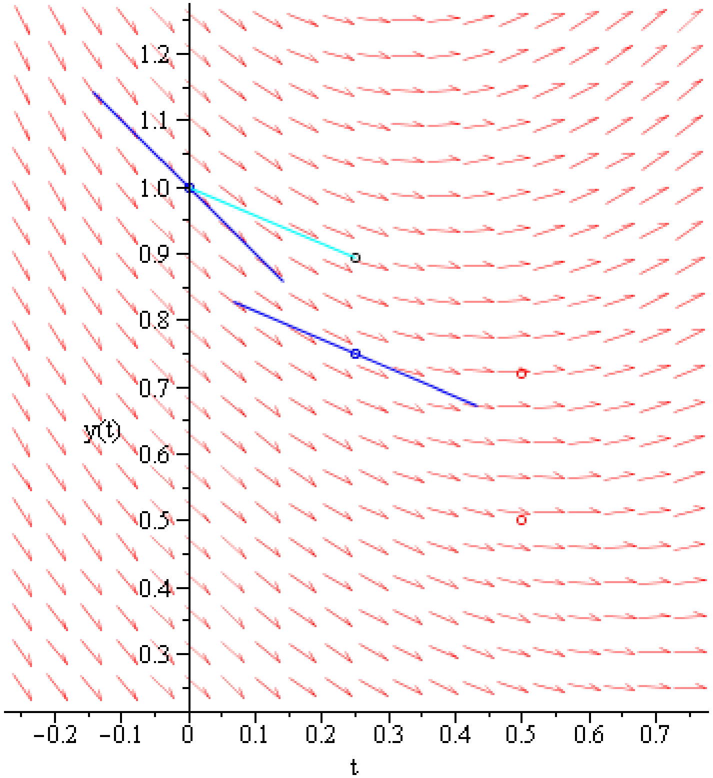
$$y(0) = 1$$



4th-order Runge-Kutta Method

Follow the slope K_2 out a distance $h/2$:

$$y^{(1)}(t) = y(t)(2-t)t + t - 1$$
$$y(0) = 1$$

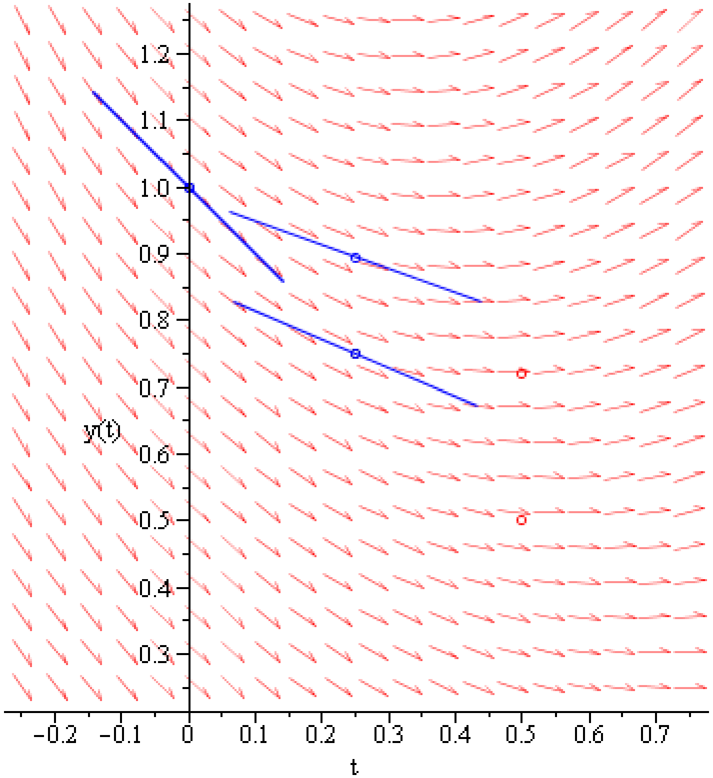


4^{th}-order Runge-Kutta Method

Determine this slope, K_3:

$$y^{(1)}(t) = y(t)(2-t)t + t - 1$$

$$y(0) = 1$$

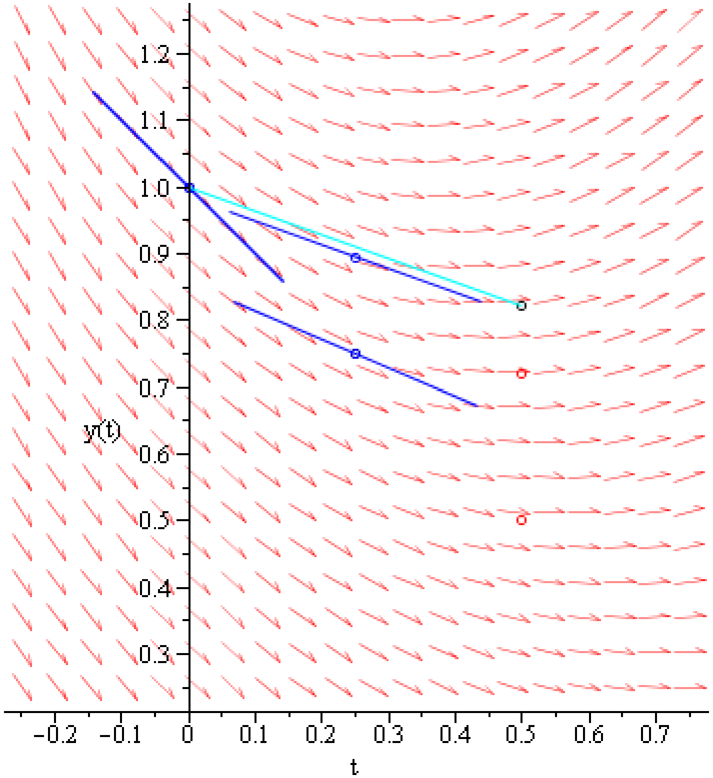


4^{th}-order Runge-Kutta Method

Follow the slope K_3 out a distance h:

$$y^{(1)}(t) = y(t)(2-t)t + t - 1$$

$$y(0) = 1$$

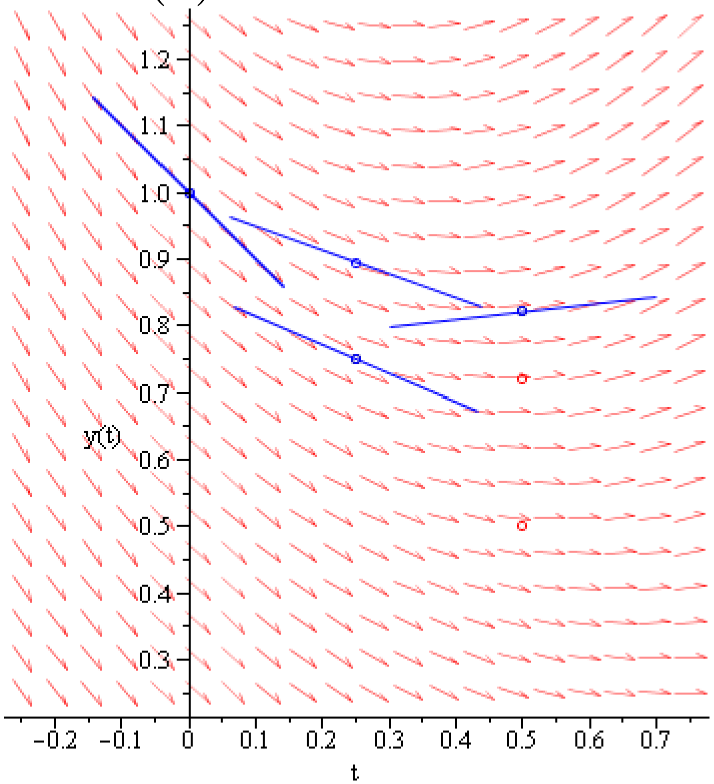


4^{th}-order Runge-Kutta Method

Determine this slope, K_4:

$$y^{(1)}(t) = y(t)(2-t)t + t - 1$$

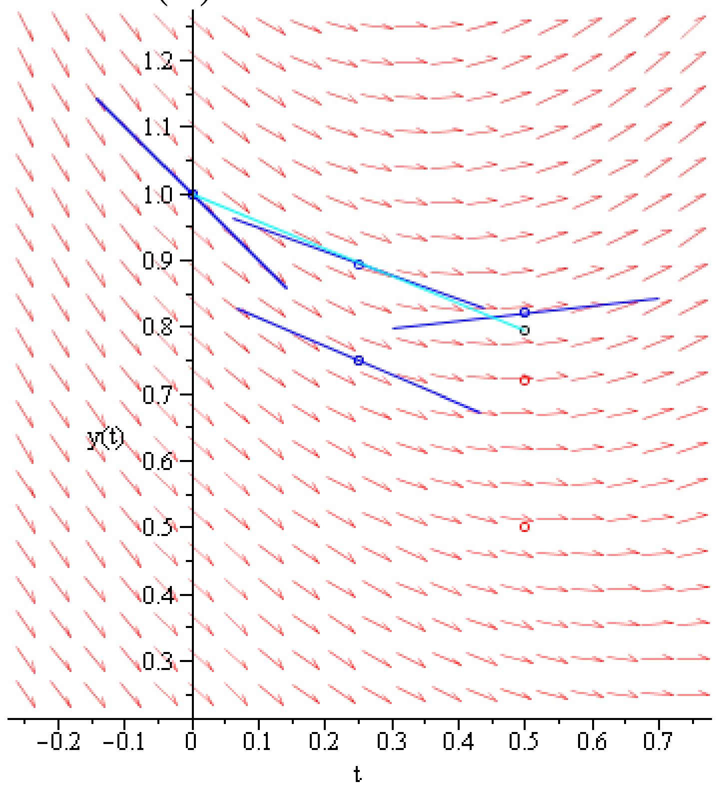
$$y(0) = 1$$



4th-order Runge-Kutta Method

Take a weighted average of the four slopes:

$$y^{(1)}(t) = y(t)(2-t)t + t - 1$$
$$y(0) = 1$$

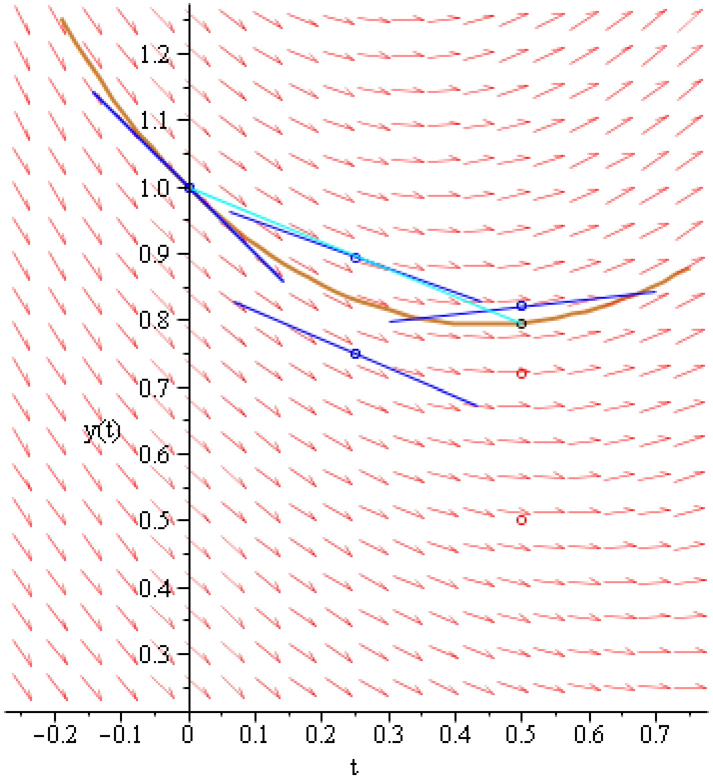


4th-order Runge-Kutta Method

The approximation looks pretty good:

$$y^{(1)}(t) = y(t)(2-t)t + t - 1$$

$$y(0) = 1$$



4^{th}-order Runge-Kutta Method

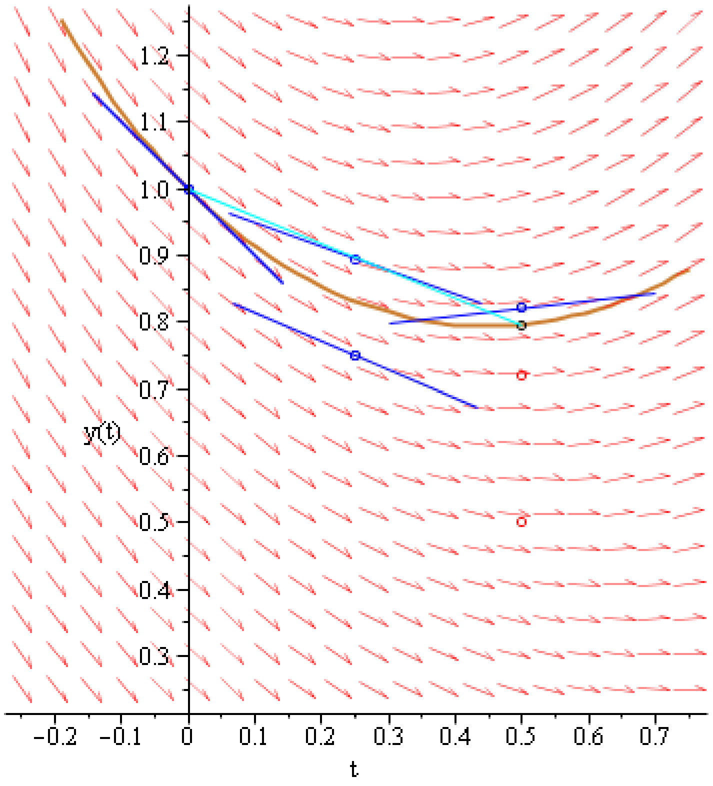
Let's compare the approximations:

$$y^{(1)}(t) = y(t)(2-t)t + t - 1$$

$$y(0) = 1$$

$$y(0.5) = 0.7963901345956429\cdots$$

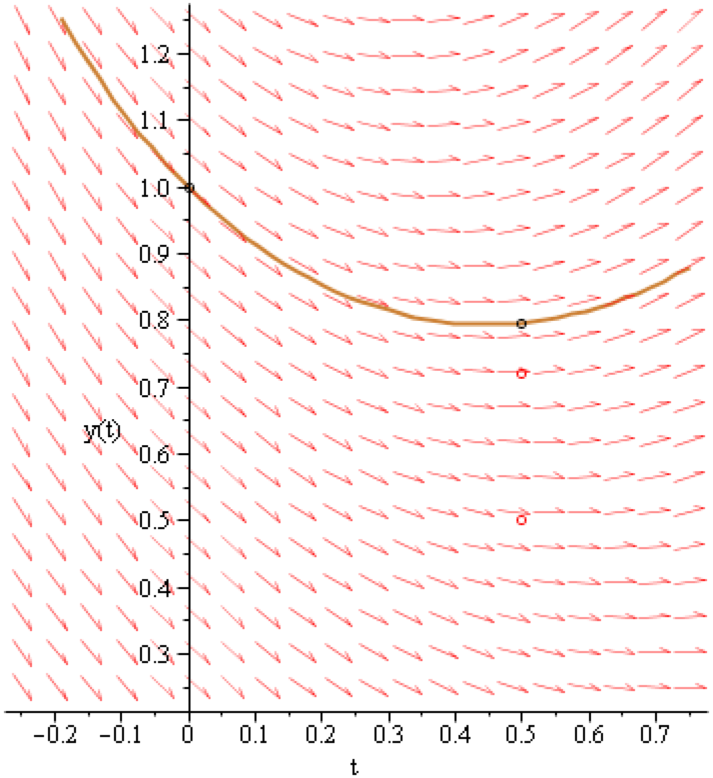
Euler:	0.5
Heun:	0.71875
Mid-point:	0.7890625
4^{th}-order R-K:	$0.79620615641666\cdots$



4th-order Runge-Kutta Method

Remember, however, this took four function evaluations

$$y^{(1)}(t) = y(t)(2-t)t + t - 1$$
$$y(0) = 1$$

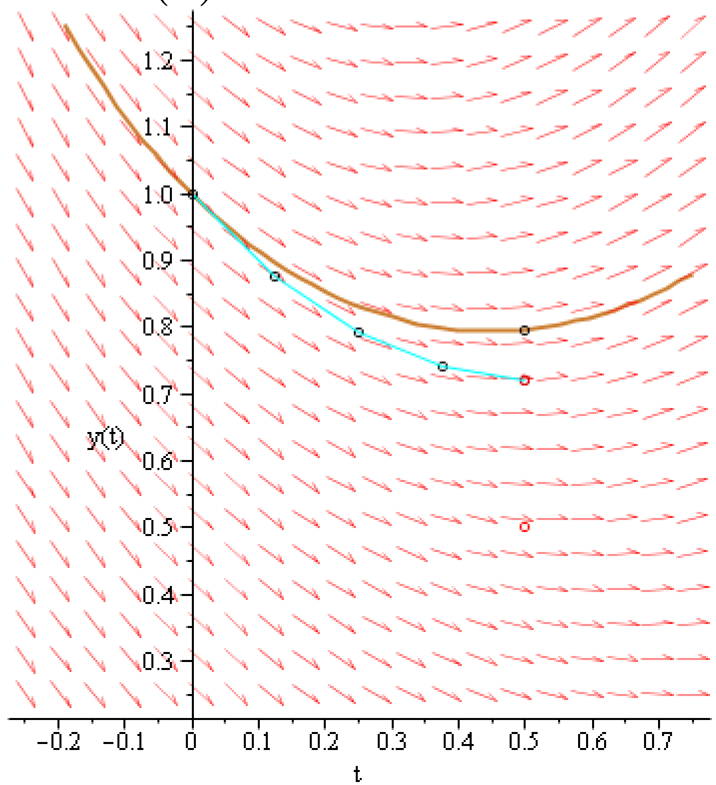


4th-order Runge-Kutta Method

Four steps of Euler's method is about as good as Heun's

$$y^{(1)}(t) = y(t)(2-t)t + t - 1$$

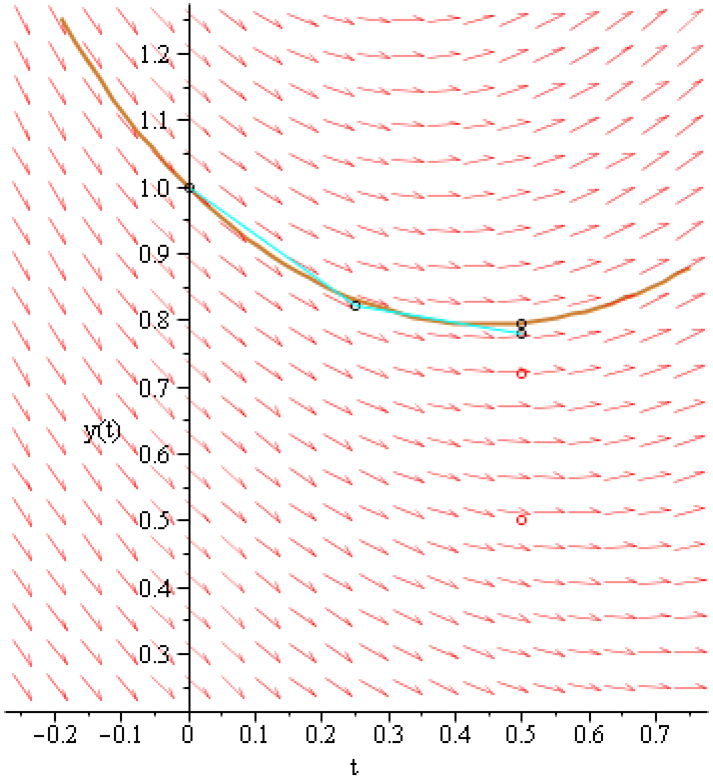
$$y(0) = 1$$



4th-order Runge-Kutta Method

Even two steps of Heun's method isn't as accurate

$$y^{(1)}(t) = y(t)(2-t)t + t - 1$$
$$y(0) = 1$$



4th-order Runge-Kutta Method

As an example,

```
>> format long
>> [t2a, y2a] = rk4( @f2a, [0, 1], 0, 2 )
t2a =
     0   1
y2a =
     0   0.265706380208333
>> [t2a, y2a] = rk4( @f2a, [0, 1], 0, 3 )
t2a =
     0   0.500000000000000   1.000000000000000
y2a =
     0   0.227653163407674   0.251787629335613
>> [t2a, y2a] = rk4( @f2a, [0, 1], 0, 4 )
t2a =
     0   0.333333333333333   0.666666666666667   1.000000000000000
y2a =
     0   0.190364751129471   0.243328110416578   0.250335465183716
```

4th-order Runge-Kutta Method

This is easily enough implemented in Matlab:

$$K_1 = f\left(t_k, y_k\right)$$
$$K_2 = f\left(t_k + \tfrac{1}{2}h, y_k + \tfrac{1}{2}h \cdot K_1\right)$$
$$K_3 = f\left(t_k + \tfrac{1}{2}h, y_k + \tfrac{1}{2}h \cdot K_2\right)$$
$$K_4 = f\left(t_k + h, y_k + h \cdot K_3\right)$$

$$y_{k+1} = y_k + h\left(\tfrac{1}{6}K_1 + \tfrac{1}{3}K_2 + \tfrac{1}{3}K_3 + \tfrac{1}{6}K_4\right)$$

```
for k = 1:(n – 1)
    % Find the four slopes K1, K2, K3, K4

    % Given y(k), find y(k + 1) by adding
    % h times the weighted average
end
```

Runge-Kutta Methods

Carl Runge and Martin Kutta observed that, given the first slope

$$K_1 = f\left(t_k, y_k\right)$$

we can approximate a second slope:

$$K_2 = f\left(t_k + c_2 h, y_k + c_2 h K_1\right)$$

where usually $0 < c_2 \le 1$, although we will allow $c_2 > 1$

Runge-Kutta Methods

Given these two slopes, we can approximate a third:

$$K_3 = f\left(t_k + c_3 h, y_k + c_3 h\left(a_{3,1}K_1 + a_{3,2}K_2\right)\right)$$

where $a_{3,1} + a_{3,2} = 1$ defines a weighted average

At this point, we could take a weighted average of K_1, K_2 and K_3 to approximate the next point:

$$y_{k+1} = y_k + h\left(b_1 K_1 + b_2 K_2 + b_3 K_3\right)$$

where $b_1 + b_2 + b_3 = 1$

Runge-Kutta Methods

The values of these coefficients for 4th-order R-K are:

$$K_1 = f\left(t_k, y_k\right)$$
$$K_2 = f\left(t_k + \tfrac{1}{2}h, y_k + \tfrac{1}{2}h \cdot K_1\right)$$
$$K_3 = f\left(t_k + \tfrac{1}{2}h, y_k + \tfrac{1}{2}h \cdot K_2\right)$$
$$K_4 = f\left(t_k + h, y_k + h \cdot K_3\right)$$

```
A = [0 0 0 0
     1 0 0 0
     0 1 0 0
     0 0 1 0]';
c = [0 1/2 1/2 1]';
b = [1/6 1/3 1/3 1/6];
```

$$y_{k+1} = y_k + h\left(\tfrac{1}{6}K_1 + \tfrac{1}{3}K_2 + \tfrac{1}{3}K_3 + \tfrac{1}{6}K_4\right)$$

$$\begin{array}{c|c} \mathbf{c} & \mathbf{A} \\ \hline & \mathbf{b} \end{array}$$

$$\begin{array}{c|cccc} 0 & & & & \\ \frac{1}{2} & 1 & & & \\ \frac{1}{2} & 0 & 1 & & \\ 1 & 0 & 0 & 1 & \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array}$$

Runge-Kutta Methods

Using these vectors, we simply access them...

```
A = [0 0 0 0
     1 0 0 0
     0 1 0 0
     0 0 1 0]';
c = [0 1/2 1/2 1]';
b = [1/6 1/3 1/3 1/6]';
```

$$K_1 = f\left(t_k, y_k\right)$$

$$K_2 = f\left(t_k + \tfrac{1}{2}h, y_k + \tfrac{1}{2}h \cdot K_1\right)$$

$$K_3 = f\left(t_k + \tfrac{1}{2}h, y_k + \tfrac{1}{2}h \cdot K_2\right)$$

$$K_4 = f\left(t_k + h, y_k + h \cdot K_3\right)$$

$$y_{k+1} = y_k + h\left(\tfrac{1}{6}K_1 + \tfrac{1}{3}K_2 + \tfrac{1}{3}K_3 + \tfrac{1}{6}K_4\right)$$

```
t0 = t_rng(1);
tf = t_rng(2);
h = (tf - t0)/(n - 1);
t_out = linspace( t0, tf, n );
y_out = zeros( 1, n );
y_out(1) = y0;
k_n = 4;

for k = 1:(n - 1)
    K = zeros( 1, k_n );

    for m = 1:k_n
        K(m) = f( t_out(k) + h*c(m), ...
                  y_out(k) + h*c(m)*K*A(:,m) );
    end

    y_out(k + 1) = y_out(k) + h*K*b;
end
```

Butcher Tableau

These tables are referred to as *Butcher tableaus*:

For Heun's method:

$$\begin{array}{c|cc} 0 & & \\ 1 & 1 & \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array}$$

For the 4th-order Runge-Kutta method:

$$\begin{array}{c|cccc} 0 & & & & \\ \frac{1}{2} & 1 & & & \\ \frac{1}{2} & 0 & 1 & & \\ 1 & 0 & 0 & 1 & \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array}$$

Butcher Tableau

Some Butcher tableaus multiply out the scalars:

$$
\begin{array}{c|cccc}
0 & & & & \\
\frac{1}{2} & 1 & & & \\
\frac{1}{2} & 0 & 1 & & \\
1 & 0 & 0 & 1 & \\
\hline
 & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6}
\end{array}
\qquad\qquad
\begin{array}{c|cccc}
0 & & & & \\
\frac{1}{2} & \frac{1}{2} & & & \\
\frac{1}{2} & 0 & \frac{1}{2} & & \\
1 & 0 & 0 & 1 & \\
\hline
 & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6}
\end{array}
$$

Comparison

Let's approximate the solutions to last weeks IVPs:

– The initial-value problem

$$y^{(1)}(t) = \left(y(t) - 1\right)^2 (t-1)^2$$
$$y(0) = 0$$

has the solution

$$y(t) = \frac{t^3 - 3t^2 + 3t}{t^3 - 3t^2 + 3t + 3}$$

```
function [dy] = f2a(t, y)
    dy = (y - 1).^2 .* (t - 1).^2;
end

function [y] = y2a( t )
    y = (t.^3 - 3*t.^2 + 3*t)./(t.^3 - 3*t.^2 + 3*t + 3);
end
```

Comparison

Being fair, let's keep the number of function evaluations the same:

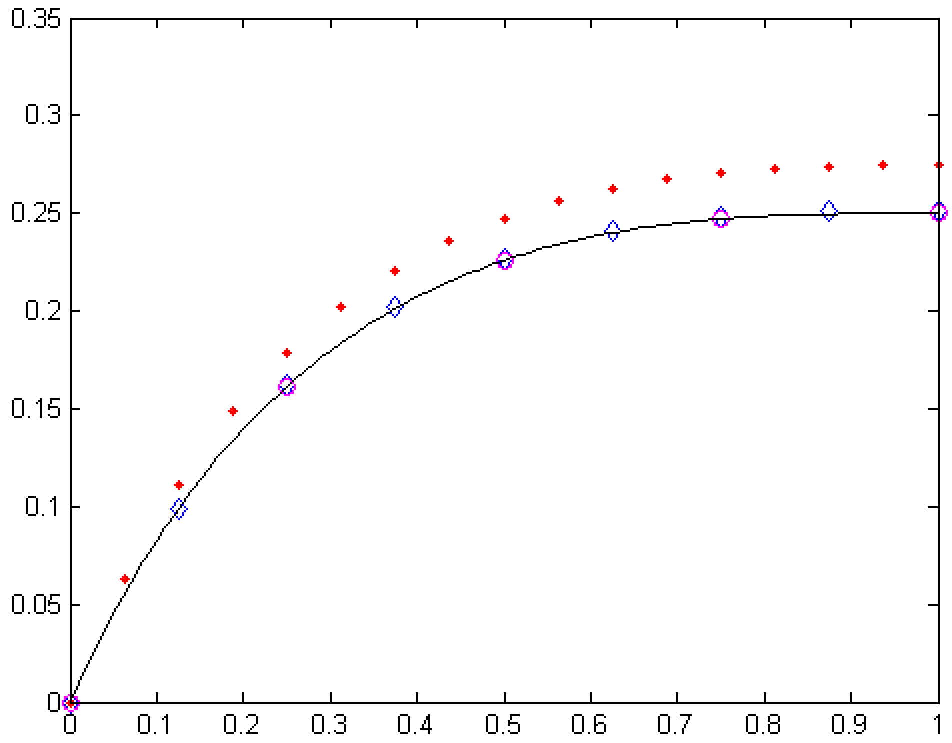
- Euler: $n = 17$
- Heun: $n = 9$
- 4^{th}-order Runge Kutta $n = 5$

```
n = 4;
[t2e, y2e] = euler( @f2a, [0, 1], 0, 4*n+1 );
[t2h, y2h] = heun( @f2a, [0, 1], 0, 2*n+1 );
[t2r, y2r] = rk4( @f2a, [0, 1], 0, n+1 );
t2s = linspace( 0, 1, 101 );

plot( t2e, y2e, 'r.' )
hold on
plot( t2h, y2h, 'd' )
plot( t2r, y2r, 'mo' )
plot( t2s, y2a( t2s ), 'k' )
```

Comparison

With this IVP, it is difficult to tell which is better at such a scale:

Euler ●	4th-order Runge Kutta o
Heun ◊	ode45 —



Comparison

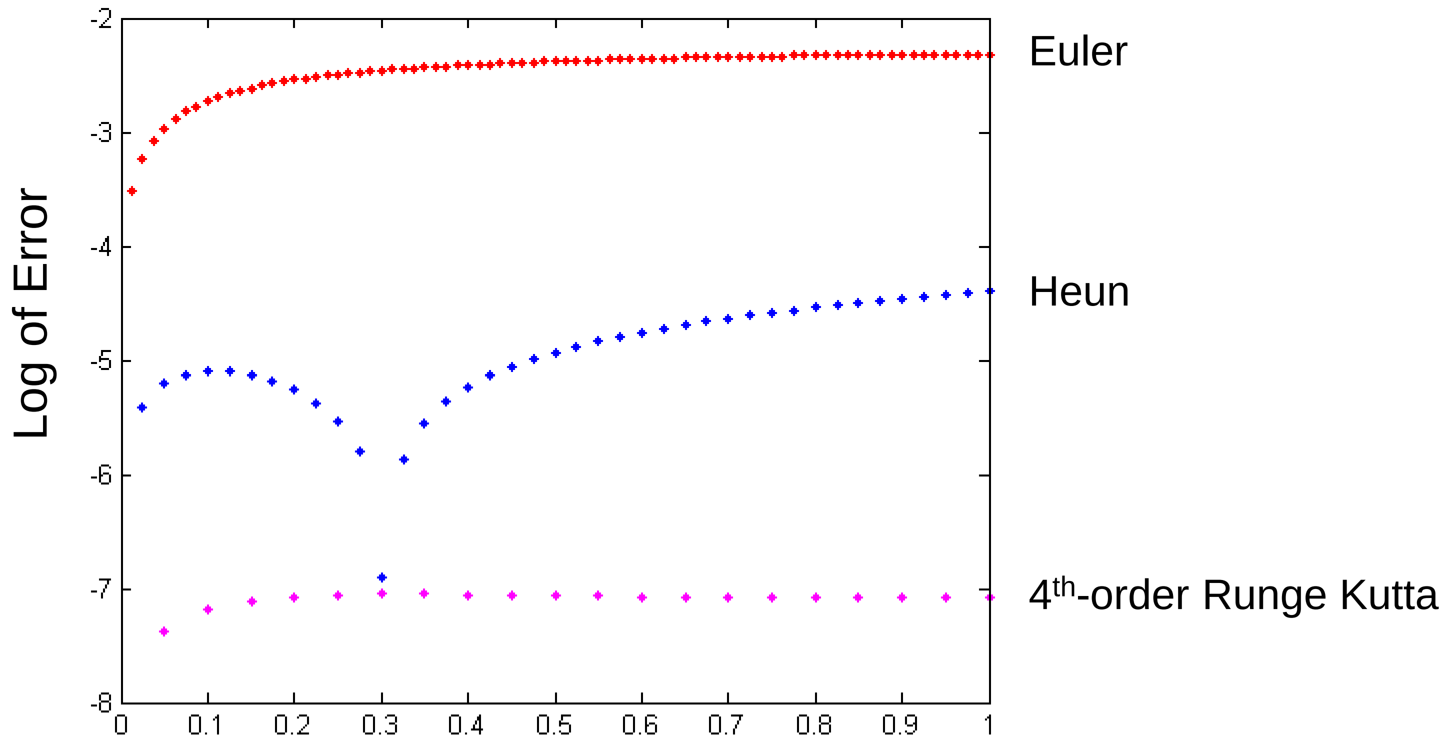
Instead, let’s plot the logarithm base 10 of the absolute errors for a greater number of points:

```
n = 20;
[t2e, y2e] = euler( @f2a, [0, 1], 0, 4*n+1 );
[t2h, y2h] = heun( @f2a, [0, 1], 0, 2*n+1 );
[t2r, y2r] = rk4( @f2a, [0, 1], 0, n+1 );
t2s = linspace( 0, 1, 101 );

plot( t2e, log10(abs(y2e - y2a( t2e ))), '.r' )
hold on
plot( t2h, log10(abs(y2h - y2a( t2h ))), '.b' )
plot( t2r, log10(abs(y2r - y2a( t2r ))), '.m' )
```

Comparison

Comparing the errors:

- The error for Euler's method is around 10^{-2}
- Heun's method has an error around 10^{-5}
- The 4th-order Runge-Kutta method has an error around 10^{-7}



4^{th}-order Runge Kutta

The other initial value problem we looked at was:

$$y^{(1)}(t) = -t \cdot y(t) + y(t) + t - \cos\left(y(t)\right)$$
$$y(0) = 1$$

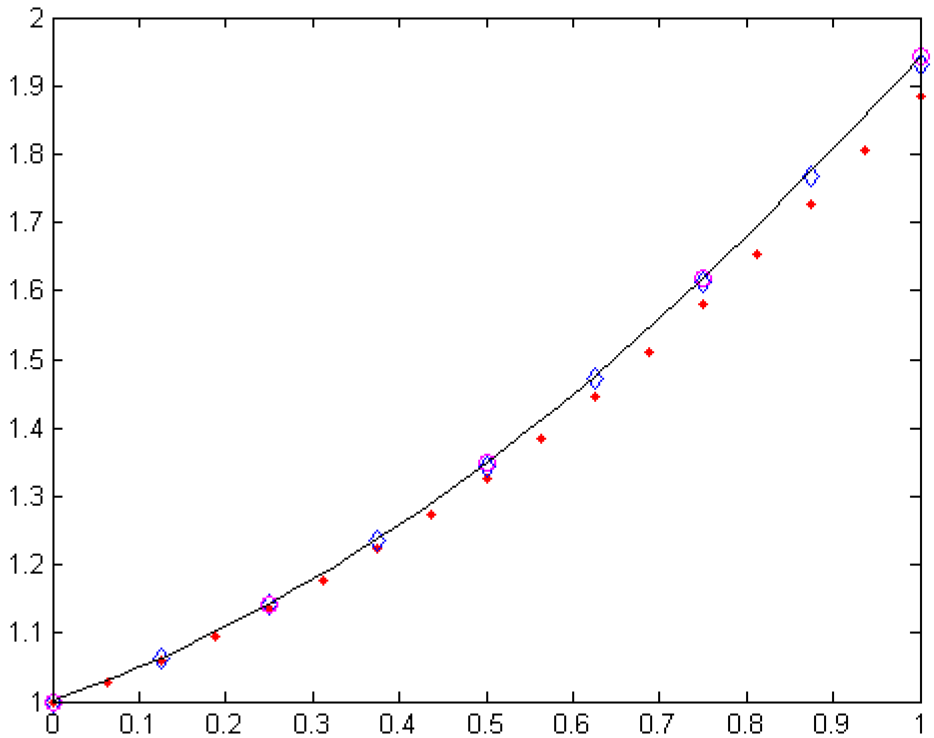
There is no analytic solution, so we had to use ode45:

```
n = 4;
[t2e, y2e] = euler( @f2b, [0, 1], 1, 4*n+1 );
[t2h, y2h] =  heun( @f2b, [0, 1], 1, 2*n+1 );
[t2r, y2r] =   rk4( @f2b, [0, 1], 1, n+1 );
[t2o, y2o] = ode45( @f2b, [0, 1], 1 );

plot( t2e, y2e, 'r.' )
hold on
plot( t2h, y2h, 'bd' )
plot( t2r, y2r, 'mo' )
plot( t2o, y2o, 'k' )
```

Comparison

With this IVP, it is difficult to tell which is better at such a scale:

Euler	●	4th-order Runge Kutta	o
Heun	◊	ode45	—



Comparison

Instead, let's again look at the logarithms of the absolute errors:

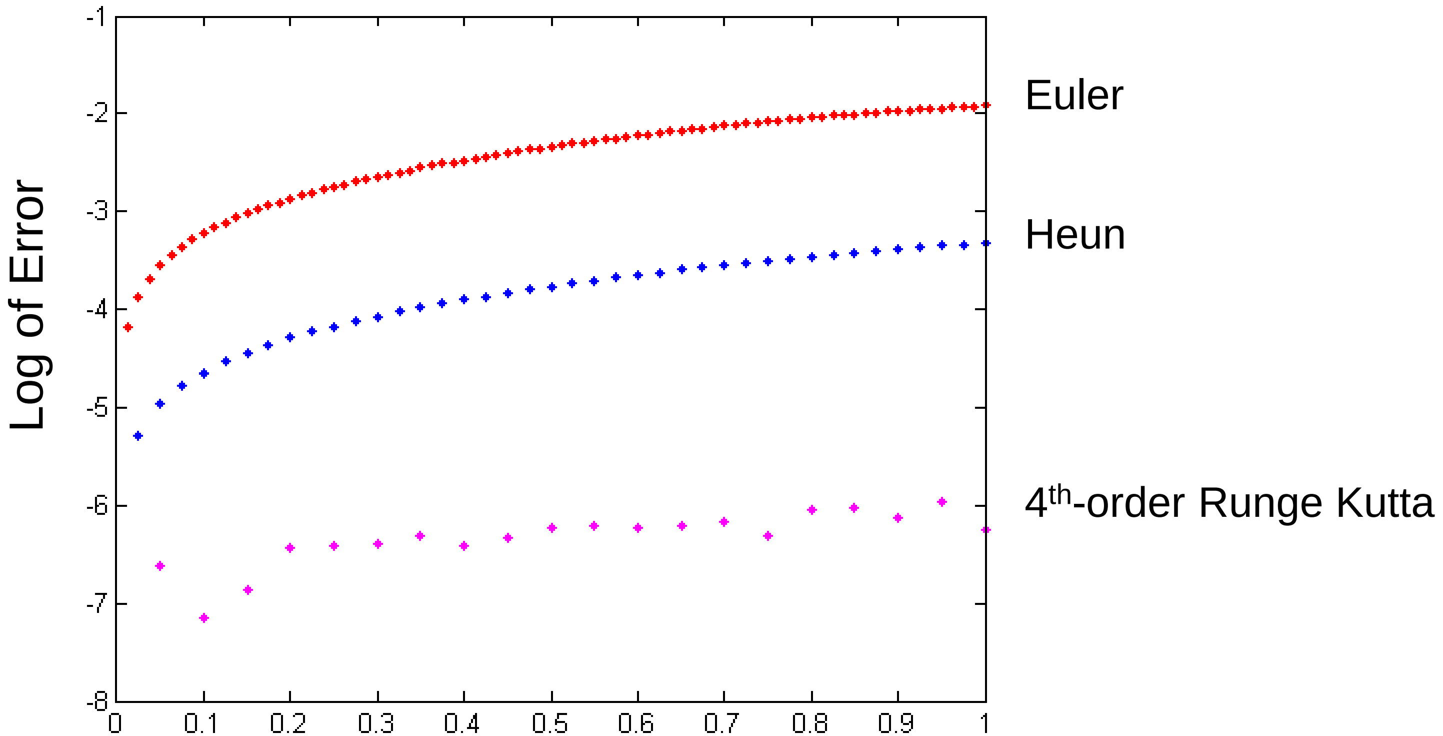
```
n = 20;
[t2e, y2e] = euler( @f2b, [0, 1], 1, 4*n+1 );
[t2h, y2h] =  heun( @f2b, [0, 1], 1, 2*n+1 );
[t2r, y2r] =   rk4( @f2b, [0, 1], 1, n+1 );
options = odeset( 'RelTol', 1e-13, 'AbsTol', 1e-13 );
[t2o, y2o] = ode45( @f2b, [0, 1], 1, options );

plot( t2e, log10(abs(y2e - interp1( t2o, y2o, t2e ))), '.r' )
hold on
plot( t2h, log10(abs(y2h - interp1( t2o, y2o, t2h ))), '.b' )
plot( t2r, log10(abs(y2r - interp1( t2o, y2o, t2r ))), '.m' )
```

We compare our approximations with the built-in ode45 function with very high relative and absolute tolerances

Comparison

Again, comparing the errors:

- The error for Euler's method is around 10^{-2}
- Heun's method has an error around 10^{-4}
- The 4th-order Runge-Kutta method has an error around 10^{-6}



The Dormand-Prince Method

With this general implementation of Runge-Kutta methods, we may now go on to the current algorithm used in Matlab today

- The routine `ode45` uses the Dormand-Prince method

The Dormand-Prince Method

Consider the ODE described by:

$$y^{(1)}(t) = y(t)(2-t)t + t - 1$$
$$y(0) = 1$$

This does not have a closed-form solution

– The best Maple can do is give an answer in terms of an integral:

```
> dsolve( {D(y)(t) = y(t)*(2 - t)*t + t - 1, y(0) = 1} );
```

$$y(t) = \left(\int_0^t e^{\frac{1}{3}\tau^2(\tau-3)} (\tau-1)\, d\tau + 1 \right) e^{\frac{1}{3}t^2(3-t)}$$

No antiderivative

The Dormand-Prince Method

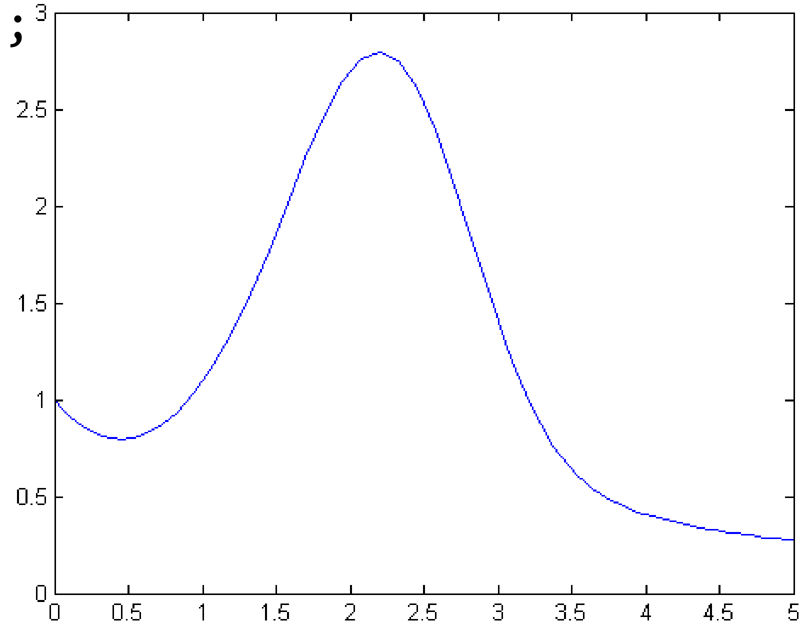
In Matlab, we would implement

$$y^{(1)}(t) = y(t)(2-t)t + t - 1$$
$$y(0) = 1$$

```
function [dy] = f3a( t, y )
    dy = y*(2 - t)*t + t - 1;
end
```

And then call:

```
>> [t3a, y3a] = ode45( @f3a, [0, 5], 1 );
>> plot( t3a, y3a );
```



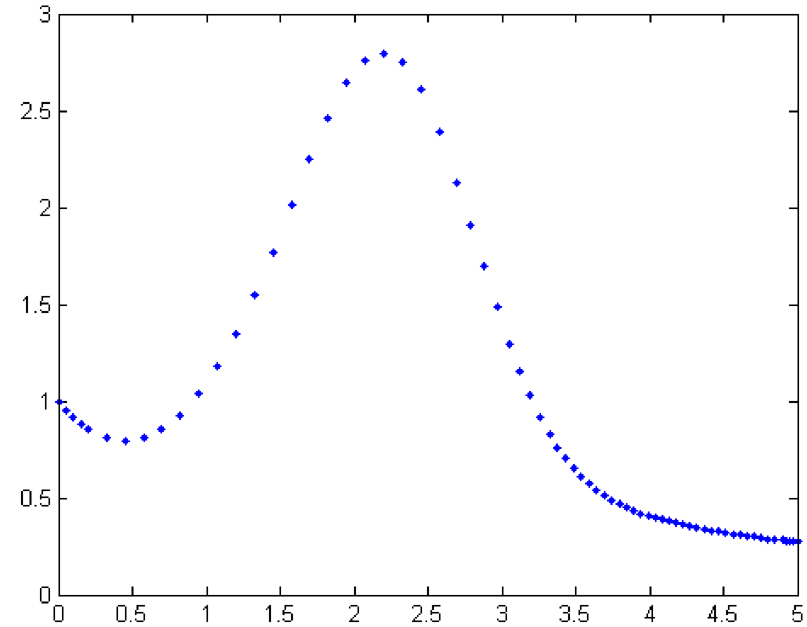
The Dormand-Prince Method

One interesting observation:

```
>> plot( t3a, y3a, '.' );
```

The points appear to be more tightly packed at the right

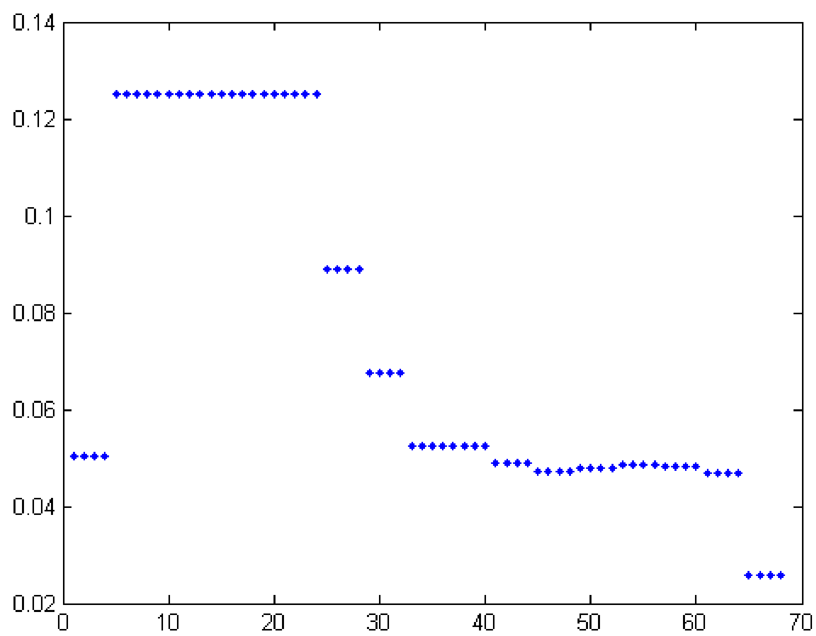
– Dormand-Prince is *adaptive*—it attempts to optimize the interval size



The Dormand-Prince Method

There are 69 points:

```
>> length( t3a );
   ans = 69

>> plot( diff( t3a ), '.' );
```



Adaptive Techniques

How does the algorithm know when to change the size of the interval?

Suppose we have two algorithms, one known to be better than the other:

- For example, Euler's method and Heun's method
- Given a point (t_k, y_k), use both methods to approximate the next point:

$$K_1 = f(t_k, y_k)$$

$$K_2 = f(t_k + h, y_k + hK_1)$$

$$y_{\text{tmp}} = y_k + hK_1$$

Euler's method

$$z_{\text{tmp}} = y_k + h\frac{K_1 + K_2}{2}$$

Heun's method

Adaptive Techniques

As a very simple example, consider:

$$y^{(1)}(t) = -y(t)$$
$$y(0) = 1$$

Suppose we want to ultimately approximate $y(0.1)$ so we start with $h = 0.1$ and we want to ensure that the error is not larger than $\varepsilon_{\text{abs}} = 0.001$

$$K_1 = f(0,1) = -1$$
$$K_2 = f(0.1, 1 + 0.1K_1) = -0.9$$
$$y_{\text{tmp}} = 1 + 0.1(-1) = 0.9$$

Euler's method

$$z_{\text{tmp}} = 1 + 0.1\frac{(-1)+(-0.9)}{2} = 0.905$$

Heun's method

Adaptive Techniques

Thus, we have two approximations:

– One is okay, the other is better

$$y_{\text{tmp}} = 0.9$$

$$z_{\text{tmp}} = 0.905$$

The actual value is $e^{-0.1} = 0.9048374180\cdots$

– The **actual** error of y_{tmp} is

$$\left| y_{\text{tmp}} - e^{-0.1} \right| = \left| 0.9 - 0.9048374180 \right| = 0.0048374180$$

– Using z_{k+1}, our **approximation** of the error is

$$\left| y_{\text{tmp}} - z_{\text{tmp}} \right| = \left| 0.9 - 0.905 \right| = 0.005$$

Adaptive Techniques

This suggests that we can use $\left|y_{\text{tmp}} - z_{\text{tmp}}\right| = 0.005$ as an approximation of the error of y_{tmp}

Problem: this error is larger than the error we were willing to tolerate

- In this case, the error should be less than $\varepsilon_{\text{abs}} = 0.001$

Solution: choose a smaller value of h

- Question: how much smaller?

Adaptive Techniques

First, we know that the error of Euler's method is $\mathrm{O}(h^2)$, that is

$$\left|y_{\text{tmp}} - z_{\text{tmp}}\right| = Ch^2 \quad \text{for some value of } C$$

If we scale h by some factor s, the error will be $C(sh)^2$:

$$C\left(sh\right)^2 < \varepsilon_{\text{abs}}$$

Adaptive Techniques

However, we want final error to be less than ε_{abs}

- The contribution of the maximum error at the kth step should be proportional to the width of the interval relative to the whole interval

Our modified goal: we want

$$C(sh)^2 < \varepsilon_{\text{abs}} \frac{sh}{t_f - t_0}$$

- Just to be sure, find a value of s such that

$$C(sh)^2 = \frac{1}{2}\varepsilon_{\text{abs}} \frac{sh}{t_f - t_0} = \frac{\varepsilon_{\text{abs}} sh}{2(t_f - t_0)}$$

Adaptive Techniques

We now have two equations:

$$\left|y_{\text{tmp}} - z_{\text{tmp}}\right| = Ch^2 \qquad C\left(sh\right)^2 = \frac{\varepsilon_{\text{abs}} sh}{2\left(t_f - t_0\right)}$$

Expand the second:

$$Cs^2h^2 = \frac{\varepsilon_{\text{abs}} sh}{2\left(t_f - t_0\right)}$$

$$s\left(Ch^2\right) = \frac{\varepsilon_{\text{abs}} h}{2\left(t_f - t_0\right)}$$

We can now substitute the first equation for Ch^2:

$$s\left|y_{\text{tmp}} - z_{\text{tmp}}\right| = \frac{\varepsilon_{\text{abs}} h}{2\left(t_f - t_0\right)}$$

Adaptive Techniques

Given the equation

$$s\left|y_{\text{tmp}} - z_{\text{tmp}}\right| = \frac{\varepsilon_{\text{abs}} h}{2\left(t_f - t_0\right)}$$

we can solve for s to get

$$s = \frac{\varepsilon_{\text{abs}} h}{2\left(t_f - t_0\right)\left|y_{\text{tmp}} - z_{\text{tmp}}\right|}$$

Adaptive Techniques

In this particular example:

$$h = 0.1$$

$$\varepsilon_{\text{abs}} = 0.001$$

$$\left|y_{\text{tmp}} - z_{\text{tmp}}\right| = 0.005$$

$$[t_0, t_f] = [0, 0.1]$$

and thus we find that

$$s = \frac{\varepsilon_{\text{abs}} h}{2\left(t_f - t_0\right)\left|y_{\text{tmp}} - z_{\text{tmp}}\right|} = \frac{0.001 \cdot 0.1}{2 \cdot 0.1 \cdot 0.005} = 0.1$$

To get the accuracy we want, we need a smaller value of h

Adaptive Techniques

Now, using $h = 0.01$, we get

$$K_1 = f(0,1) = -1$$

$$K_2 = f(0.01, 1 + 0.01 \cdot K_1) = -0.99$$

$$y_{\text{tmp}} = 1 + 0.01(-1) = 0.99$$ Euler's method

$$z_{\text{tmp}} = 1 + 0.01\frac{(-1)+(-0.99)}{2} = 0.99005$$ Heun's method

The actual value is $e^{-0.031} = 0.9900498337\cdots$

- The absolute error using Euler's method is $0.0000498337\cdots$ which is of the same order of $\frac{\varepsilon_{\text{abs}} h}{2(t_f - t_0)} = 0.00005$
- Use y_{tmp} as the approximation y_{k+1}

Adaptive Techniques

If we repeat this process, we get the output

```
>> t_out =
 0 0.0100 0.0200 0.0301 0.0403 0.0506 0.0610 0.0715 0.0822 0.0929 0.1

>> y_out =
 1 0.9900 0.9801 0.9702 0.9603 0.9504 0.9405 0.9306 0.9207 0.9108 0.9044

>> format long
>> y_out(end)
    0.904837418035960
>> exp( -0.1 )
    0.904837418035960
>> abs( y_out(end) - exp( -0.1 ) )
    4.599948547183708e-004
```

$$\approx \frac{\varepsilon_{\text{abs}}}{2} = 0.0005$$

Adaptive Techniques

In general, however, it isn't always a good idea to update

```
h = s*h;
```

as s could be either very big or very small

It is safer to be a little conservative—do not expect h to change too much in the short run:

- If $s \geq 2$, double the value of h
- If $1 \leq s < 2$, leave h unchanged, and
- If $s < 1$, halve h and try again

The Dormand-Prince Method

Dormand-Prince calculates seven different slopes:

$$K_1, K_2, K_3, K_4, K_5, K_6, \text{ and } K_7$$

These slopes are then used in two different linear combinations to find two approximations of the next point:

- One is $\mathrm{O}(h^4)$ while the other is $\mathrm{O}(h^5)$
- The coefficients of the 5th-order approximate were chosen to minimize its error
- We now use these two approximations to find s:

$$s = \sqrt[4]{\frac{h\varepsilon_{\text{abs}}}{2\left(t_f - t_0\right)\left|y_{\text{tmp}} - z_{\text{tmp}}\right|}}$$

The Dormand-Prince Method

The *modified* Butcher tableau of the Dormand-Prince method is:

$$
\begin{array}{c|ccccccc}
0 & & & & & & & \\
\frac{1}{5} & 1 & & & & & & \\
\frac{3}{10} & \frac{1}{4} & \frac{3}{4} & & & & & \\
\frac{4}{5} & \frac{11}{9} & -\frac{14}{3} & \frac{40}{9} & & & & \\
\frac{8}{9} & \frac{4843}{1458} & -\frac{3170}{243} & \frac{8056}{729} & -\frac{53}{162} & & & \\
1 & \frac{9017}{3168} & -\frac{355}{33} & \frac{46732}{5247} & \frac{49}{176} & -\frac{5103}{18656} & & \\
1 & \frac{35}{384} & 0 & \frac{500}{113} & \frac{125}{192} & -\frac{2187}{6784} & \frac{11}{84} & \\
\hline
 & \frac{5179}{57600} & 0 & \frac{7571}{16695} & \frac{393}{640} & -\frac{92097}{339200} & \frac{187}{2100} & \frac{1}{40} \\
 & \frac{35}{384} & 0 & \frac{500}{1113} & \frac{125}{192} & -\frac{2187}{6784} & \frac{11}{84} & 0
\end{array}
\begin{array}{l}
\\ \\ \\ \\ \\ \\ \\
\text{O}(h^4)\text{ approximation} \\
\text{O}(h^5)\text{ approximation}
\end{array}
$$

The Dormand-Prince Method

Each row sums to 1

$$
\begin{array}{c|ccccccc}
0 & & & & & & & \\
\frac{1}{5} & 1 & & & & & & \\
\frac{3}{10} & \frac{1}{4} & \frac{3}{4} & & & & & \\
\frac{4}{5} & \frac{11}{9} & -\frac{14}{3} & \frac{40}{9} & & & & \\
\frac{8}{9} & \frac{4843}{1458} & -\frac{3170}{243} & \frac{8056}{729} & -\frac{53}{162} & & & \\
1 & \frac{9017}{3168} & -\frac{355}{33} & \frac{46732}{5247} & \frac{49}{176} & -\frac{5103}{18656} & & \\
1 & \frac{35}{384} & 0 & \frac{500}{113} & \frac{125}{192} & -\frac{2187}{6784} & \frac{11}{84} & \\
\hline
 & \frac{5179}{57600} & 0 & \frac{7571}{16695} & \frac{393}{640} & -\frac{92097}{339200} & \frac{187}{2100} & \frac{1}{40} \\
 & \frac{35}{384} & 0 & \frac{500}{1113} & \frac{125}{192} & -\frac{2187}{6784} & \frac{11}{84} & 0
\end{array}
$$

The Dormand-Prince Method

Each row sums to 1

– In the literature, for example, the fourth row would be multiplied by 4/5:

$$
\begin{array}{c|ccccccc}
0 & & & & & & & \\
\frac{1}{5} & 1 & & & & & & \\
\frac{3}{10} & \frac{1}{4} & \frac{3}{4} & & & & & \\
\frac{4}{5} & \frac{11}{9} & -\frac{14}{3} & \frac{40}{9} & & & & \\
\frac{8}{9} & \frac{4843}{1458} & -\frac{3170}{243} & \frac{8056}{729} & -\frac{53}{162} & & & \\
1 & \frac{9017}{3168} & -\frac{355}{33} & \frac{46732}{5247} & \frac{49}{176} & -\frac{5103}{18656} & & \\
1 & \frac{35}{384} & 0 & \frac{500}{113} & \frac{125}{192} & -\frac{2187}{6784} & \frac{11}{84} & \\
\hline
 & \frac{5179}{57600} & 0 & \frac{7571}{16695} & \frac{393}{640} & -\frac{92097}{339200} & \frac{187}{2100} & \frac{1}{40} \\
 & \frac{35}{384} & 0 & \frac{500}{1113} & \frac{125}{192} & -\frac{2187}{6784} & \frac{11}{84} & 0
\end{array}
$$

The Dormand-Prince Method

You can, if you want, use:

```
A = [ 0          0          0         0          0         0   0;
      1          0          0         0          0         0   0;
     1/4        3/4         0         0          0         0   0;
    11/9       -14/3       40/9       0          0         0   0;
  4843/1458 -3170/243  8056/729  -53/162       0         0   0;
  9017/3168  -355/33  46732/5247  49/176 -5103/18656   0   0;
    35/384      0       500/1113 125/192 -2187/6784  11/84 0]';

by = [5179/57600 0 7571/16695 393/640 -92097/339200 187/2100 1/40]';
bz = [  35/384   0  500/1113  125/192  -2187/6784    11/84    0]';

c = [0 1/5 3/10 4/5 8/9 1 1]';
```

The Dormand-Prince Method

All we need now are different matrices:

```
A = [...]';
c = [...]';
by = [...]';
bz = [...]';

// ...

n_K = 7;
K = zeros( 1, n_K );

for m = 1:n_K
    K(m) = f( t_out(k) + h*c(m), ...
              y_out(k) + h*c(m)*K*A(:,m) );
end

y_tmp = y_out(k) + h*K*by;
z_tmp = y_out(k) + h*K*bz;

% Determine s and modify h as appropriate
```

The Dormand-Prince Method

What value of h?

- Previously, we specified the interval and the number of points

For Dormand-Prince, we will specify an initial value of h

- `ode45` actually determines a good initial value of h

We will not know *apriori* how many steps we will require

- The value of h could increase or decrease depending on the problem
- We will have to have a different counter tracking where we are in the array

The Dormand-Prince Method

We will therefore grow the vectors `t_out` and `y_out`:

```
>> t_out = 1
t_out =
     1

>> t_out(2) = 1.1
t_out =
    1.0000    1.1000

>> t_out(3) = 1.2
t_out =
    1.0000    1.1000    1.2000

>> size( t_out )
ans =
     1     3
```

The Dormand-Prince Method

Thus, the steps we will take:

```
% Initialize t_out and y_out
% Initialize our location to k = 1
%
% while t_out(k) < tf
%     Use Dormand Prince to find two approximations
%     y_tmp and z_tmp to approximate y(t) at
%     t = t_out(k) + h for the current value of h
%
%     Calculate the scaling factor 's'
%
%     if s >= 2,
%         We use z_tmp to approximate y_out(k + 1)
%         t_out(k + 1) is the previous t-value plus h
%         Increment k and double the value of h for the
%         next iteration.
```

The Dormand-Prince Method

```
%     else if s >= 1,
%         We use z_tmp to approximate y_out(k + 1)
%         t_out(k + 1) is the previous t-value plus h
%         In this case, h is neither too large or too
%         small, so only increment k
%     else s < 1
%         Divide h by two and try again with the smaller
%         value of h (just go through the loop again
%         without updating t_out, y_out, or k)
%     end
%
%     We must make one final check before we end the loop:
%        if t_out(k) + h > tf, we must reduce the
%        size of h so that t_out(k) + h == tf
% end
```

Runge-Kutta Methods

As an example,

```
>> format long
>> [t2a_out, y2a_out] = dp45( @f2a, [0, 1], 0, 0.1, 0.001 )
t2a_out =
  0   0.100000000000000   0.300000000000000   0.700000000000000   1.000000000000000
y2a_out =
  0   0.082849238339751   0.179654557289050   0.244899192641371   0.249996176157670

>> [t2a_out, y2a_out] = dp45( @f2a, [0, 1], 0, 0.1, 0.0001 )
t2a_out =
  0   0.100000000000000   0.300000000000000   0.500000000000000   0.900000000000000   1.000000000000000
y2a_out =
  0   0.082849238339751   0.179654557289050   0.225805610339612   0.249811473416968   0.249999020845017
```

$$y^{(1)}(t) = f_{2a}(t, y(t)) = (y(t) - 1)^2 (t - 1)^2$$

$$y(0) = 0$$

$$y_{2a}(t) = \frac{t^3 - 3t^2 + 3t}{t^3 - 3t^2 + 3t + 3}$$

Runge-Kutta Methods

In the 2nd example, the values of $\mathbf{K}$, y, z, and s at the four steps are

$$t_1 = 0.0$$
$$h = 0.1$$

Approximating $t_2 = 0.1$

$$y_{\text{tmp}} = 0.082849167706690$$
$$z_{\text{tmp}} = 0.082849238339751$$
$$s = 2.900617713421327$$

$$\mathbf{K}^T = \begin{pmatrix} 1.000000000000000 \\ 0.922368160000000 \\ 0.888484042496882 \\ 0.733102686848602 \\ 0.707007263868476 \\ 0.678484594287190 \\ 0.681344070887320 \end{pmatrix}$$

Note: $y_{2a}(0.1) = 0.082849281565271$

Note: double the value of h for the next interval...

Runge-Kutta Methods

In the 2nd example, the values of $\mathbf{K}$, y, z, and s at the four steps are

$$t_2 = 0.1$$
$$h = 0.2$$

$$\mathbf{K}^T = \begin{pmatrix} 0.681344070887320 \\ 0.585701568035401 \\ 0.547129735146171 \\ 0.379211266617984 \\ 0.350996491857387 \\ 0.323819676610967 \\ 0.329753656234546 \end{pmatrix}$$

Approximating $t_3 = 0.3$

$$y_{\text{tmp}} = 0.179652821005170$$
$$z_{\text{tmp}} = 0.179654557289050$$
$$s = 1.549154799018235$$

Note: $y_{2a}(0.3) = 0.179655455291222$

Runge-Kutta Methods

In the 2nd example, the values of $\mathbf{K}$, y, z, and s at the four steps are

$$t_3 = 0.3$$
$$h = 0.2$$

Approximating $t_4 = 0.5$

$$y_{\text{tmp}} = 0.225805183165541$$
$$z_{\text{tmp}} = 0.225805610339612$$
$$s = 2.199625668274607$$

$$\mathbf{K}^T = \begin{pmatrix} 0.329753656234546 \\ 0.283793257387157 \\ 0.263869405085534 \\ 0.177463478001606 \\ 0.164100516485867 \\ 0.149106549812094 \\ 0.149844238245405 \end{pmatrix}$$

Note: $y_{2a}(0.5) = 0.225806451612903$

Note: double the value of h for the next interval...

Runge-Kutta Methods

In the 2nd example, the values of $\mathbf{K}$, y, z, and s at the four steps are

$$t_4 = 0.5$$
$$h = 0.4$$

$$\mathbf{K}^T = \begin{pmatrix} 0.149844238245405 \\ 0.102481217539320 \\ 0.083509894743704 \\ 0.018218081633068 \\ 0.011628653029033 \\ 0.005569766971191 \\ 0.005627828254168 \end{pmatrix}$$

Approximating $t_5 = 0.9$

$$y_{\text{tmp}} = 0.249809684810377$$
$$z_{\text{tmp}} = 0.249811473416968$$
$$s = 1.828642429049916$$

Note: $y_{2a}(0.9) = 0.249812453113278$

Note: but $0.9 + 0.4 > 1$, so use $h = 1 - 0.9 = 0.1$

Runge-Kutta Methods

In the 2nd example, the values of $\mathbf{K}$, y, z, and s at the four steps are

$$t_5 = 0.9$$
$$h = 0.1$$

Approximating $t_6 = 1.0$:

$$y_{\text{tmp}} = 0.249999020696080$$
$$z_{\text{tmp}} = 0.249999020845017$$
$$s = 13.536049392119093$$

$$\mathbf{K}^T = \begin{pmatrix} 0.005627828254168 \\ 0.003600729349110 \\ 0.002756729986632 \\ 0.000225001411005 \\ 0.000069444596853 \\ 0 \\ 0 \end{pmatrix}$$

Note: $y_{2a}(1) = 0.25$

Remember, we artificially reduced h

The Dormand-Prince Method

You will use the Dormand-Prince function in Labs 5 and 6 and in NE 217

- Dormand-Prince is the algorithm used in the Matlab ODE solver

```
>> help ode45
 ODE45  Solve non-stiff differential equations, medium order method.
    [TOUT,YOUT] = ODE45(ODEFUN,TSPAN,Y0) with TSPAN = [T0 TFINAL]
    integrates the system of differential equations y' = f(t,y) from
    time T0 to TFINAL with initial conditions Y0.

    ODEFUN is a function handle. For a scalar T and a vector Y,
    ODEFUN(T,Y) must return a column vector corresponding to f(t,y).

    Each row in the solution array YOUT corresponds to a time
    returned in the column vector TOUT.
```

Summary

We have looked at solving initial-value problems with better techniques:

- Weighted averages and integration techniques
- 4th-order Runge Kutta
- Adaptive methods
- The Dormand-Prince method

References

[1] Glyn James, *Modern Engineering Mathematics*, 4th Ed., Prentice Hall, 2007.

[2] Glyn James, *Advanced Modern Engineering Mathematics*, 4th Ed., Prentice Hall, 2011.

[3] J.R. Dormand and P. J. Prince, "A family of embedded Runge-Kutta formulae," *J. Comp. Appl. Math.*, Vol. 6, 1980, pp. 19-26.