Last time we introduced Runge-Kutta methods for solving first order ODEs

- Given an initial $y = y_o$ at $t = t_o$ choose a step size, h.
- Find $y = y(t_o + h)$ by approximating the derivative at a point by the slope of a secant line at two nearby points.
- Runge-Kutta adjusts the slope by *correcting* the slope at intermediate points between $t = t_o$ and $t = t_o + h$

Learning Goals:

1. Adaptive Step Size
2. Systems of first order ODEs

How good is my answer? In solving ODEs numerically, we've not determined whether the answer obtained is *'good'*.

Let's now examine the question of how good our numerical answer is. Do question 1 on the worksheet and S T O P.

Most common way to ascertain the *goodness* of the solution to a numerically solved ODE is to use *two different step sizes* and *compare the results*.

i. A possible strategy. Compute y with step size h then $\hat{y}$ with step size $h/2$
ii. If $| y - \hat{y} |$ *small* keep y, else repeat process with step size = $h/2$
iii. Proceed to next time step with *original* step size h.
iv. For each step use a *Runge-Kutta* technique.

Do question 1 on worksheet.

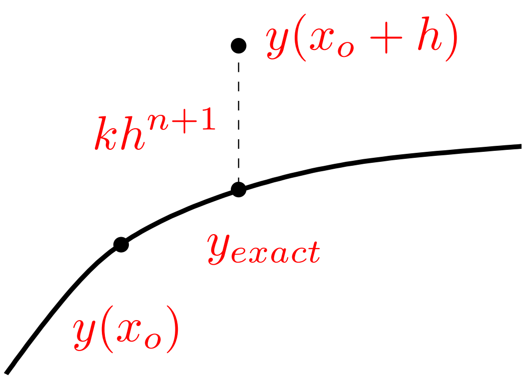
(1)

```
function myodesolver2(to,yo,h,tf,tol)
%driver program to solve odes using adaptive stepping with the adaptive
%step set to h/2.
t = to;
ho=h;
while t < tf
   y1 = myrk4(to,yo,h);
   y2 = myrk4(to,yo,h/2);
   if abs(y2 - y1) < tol
      plot(t+h,y1,'.k','MarkerSize',15)
      hold on
      t = t+h;
      yo = y1;
      h = ho;  % resetting h to original h
   else
      h = h/2;  % tolerance not met, try again by halving h
   end
end
end
```

Having a rigid adaptive step size is not optimal. We would like our code to *adjust step to conditions of the solution.*

Fehlberg modification

Use *normal* Runge-Kutta n^{th} order method to obtain a solution for ODE. The error will be of order $n + 1$ in h. That is we have $y(x_o + h) = y_{exact} + kh^{n+1}$ where x_o is the point where solving the ODE, h is the step size, and k an unknown constant.



Now do an $(n+1)^{th}$ order and we get that $\hat{y}(x_o + h) = y_{exact} + \hat{k}h^{n+2}$

Subtracting the two and using condition that h is small gives

$$y(x_o + h) - \hat{y}(x_o + h) = kh^{n+1} - \hat{k}h^{n+2} \approx kh^{n+1}$$

We can now solve for the constant k to find that $k \approx \frac{y - \hat{y}}{h^{n+1}}$

Now, we are going to require that the error between each step size be the same, let's call that $\hat{\epsilon}$

So we can say that any new step size h_{new} be such that

$$k h_{new}^{n+1} = \underbrace{\frac{y - \hat{y}}{h^{n+1}}}_{k} h_{new}^{n+1} \leq \epsilon$$

Or that

$$h_{new} \leq h \sqrt[n+1]{\frac{\hat{\epsilon}}{|y(x_o + h) - \hat{y}(x_o + h)|}}$$

Note that if error is ``too large", then

$$|y(x_o + h) - \hat{y}(x_o + h)| > \hat{\epsilon} \Rightarrow \frac{h_{new}}{h} < 1 \Rightarrow h_{new} < h$$

and the step size h is too large.

Algorithm:

1. Find y and $\hat{y}$ using the same h.
2. Calculate h_{new}
3. If $h_{new} > h$ step size is good and keep $y(x_o + h)$ else set $h = h_{new}$ and repeat

One modification to step size. The previous result used absolute error. In general, relative error is preferred. So we modify the definition for h_{new} as

$$h_{new} = h \sqrt[n+1]{\frac{\epsilon}{|(y-\hat{y})/\hat{y}|}}$$

where ϵ is now the largest ***relative*** error allowed.

Note that this is all well and good, but it does require one to use two different orders of Runge-Kutta. That means computing *two different sets* of intermediate functions for each order.

Luckily Fehlberg found intermediate functions for 4^{th} and 5^{th} order Runge-Kutta methods that were identical, saving us a lot of work. In other words, Fehlberg found that

4^{th} order: $$y = y_o + h\left(b_o f_o + b_2 f_2 + b_3 f_3 - b_4 f_4\right)$$

5^{th} order: $$\hat{y} = y_o + h\left(c_o f_o + c_2 f_2 + c_3 f_3 - c_4 f_4 + c_5 f_5\right)$$

On the next slide, I put up the various *f's*. But remember that the key here is that both the 4^{th} and 5^{th} order Runge-Kutta share the same *f's*

$$y = y_o + h\left(b_o f_o + b_2 f_2 + b_3 f_3 - b_4 f_4\right)$$
$$\hat{y} = y_o + h\left(c_o f_o + c_2 f_2 + c_3 f_3 - c_4 f_4 + c_5 f_5\right)$$

$$\begin{aligned}
f_o &= f(x_o, y_o) \\
f_1 &= f\left(x_o + \frac{h}{4},\ y_o + \frac{h}{4} f_o\right) \\
f_2 &= f\left(x_o + \frac{3h}{8},\ y_o + \frac{3h}{32} f_o + \frac{9h}{32} f_1\right) \\
f_3 &= f\left(x_o + \frac{12h}{13},\ y_o + \frac{1932h}{2197} f_o - \frac{7200h}{2197} f_1 + \frac{7296h}{2197} f_2\right) \\
f_4 &= f\left(x_o + h,\ y_o + \frac{439h}{216} f_o - 8h f_1 + \frac{3680h}{513} f_2 - \frac{845h}{4104} f_3\right) \\
f_5 &= f\left(x_o + \frac{h}{2},\ y_o - \frac{8h}{27} f_o + 2h f_1 - \frac{3544h}{2565} f_2 + \frac{1859h}{4104} f_3 - \frac{11h}{40} f_4\right)
\end{aligned}$$

$$h_{new} = h \sqrt[n+1]{\frac{\epsilon}{|(y - \hat{y})/\hat{y}|}}$$

So while the constants b_i and c_i are different, the functions f_i are identical

Do question (3) on the worksheet and S T O P

Do questions (4) and (5) on the worksheet and S T O P

- Most physics problems involve *multiple dependent* variables

$$\frac{d\boldsymbol{p}}{dt} = m\frac{d\boldsymbol{v}}{dt}$$

← Three equations which may be coupled

- Thus we need to study how *systems of ODEs* can be solve numerically.

- It turns out we can handle *systems of ODEs* pretty straight-forwardly

1. Write the dependent variables as a vector, $\vec{\boldsymbol{S}} = \begin{pmatrix} x \\ y \end{pmatrix}$
2. Do the same with RHS of the ODEs so that we have $\vec{\boldsymbol{F}} = \begin{pmatrix} f(x,y) \\ g(x,y) \end{pmatrix}$
3. The *system of ODEs* can now be written as $\frac{d\vec{\boldsymbol{S}}}{dt} = \vec{\boldsymbol{F}}(x,y).$

Do questions 6 & 7 on the worksheet

(7)

1. Rather then sending over the variable *y* to function **derivs**, one sends the vector ***S*** so that the function looks like

```
function [der] = derivs(t,S)
    x = S(1);
    y = S(2);
    der = [f(x,y); g(x,y)]
end
```