Second Order Runge-Kutta

Contents

Statement of Problem

Consider the situation in which the solution, $y(t)$, to a differential equation

$$\frac{dy(t)}{dt} = y'(t) = f(y(t), t), \qquad \text{with } y(t_0) = y_0$$

is to be approximated by computer (starting from some known initial condition, $y(t_0)=y_0$; also, note that the tick mark denotes differentiation). The following text develops an intuitive technique for doing so, and presents some examples. This technique is known as "Second Order Runge-Kutta".

Second Order Runge-Kutta Method (Intuitive)

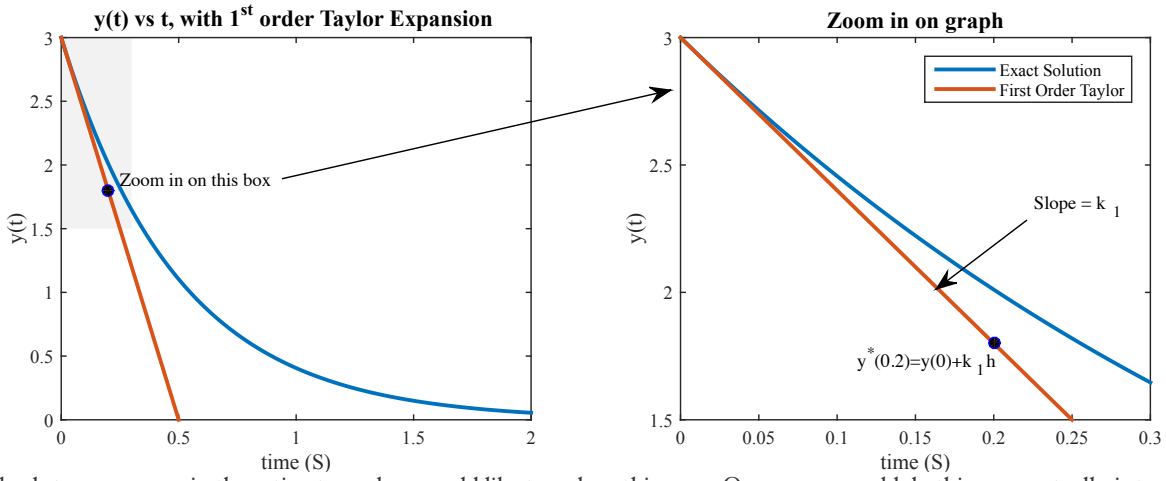
A First Order Linear Differential Equation with No Input

The first order Runge-Kutta method used the derivative at time t_0 ($t_0=0$ in the graph below) to estimate the value of the function at one time step in the future. If you are not familiar with it, you should read the section entitled: *A First Order Linear Differential Equation with No Input*. We repeat the central concept of generating a step forward in time in the following text.

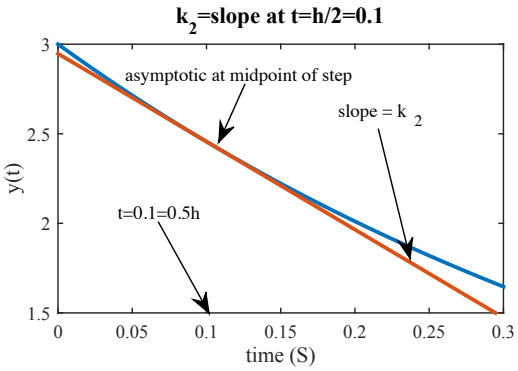
$$\frac{dy(t)}{dt} + 2y(t) = 0 \quad \text{or} \quad \frac{dy(t)}{dt} = -2y(t)$$

with the initial condition set as $y(0)=3$. The exact solution in this case is $y(t)=3e^{-2t}$, $t\geq 0$, though in general we won't know this and will need numerical integration methods to generate an approximation.

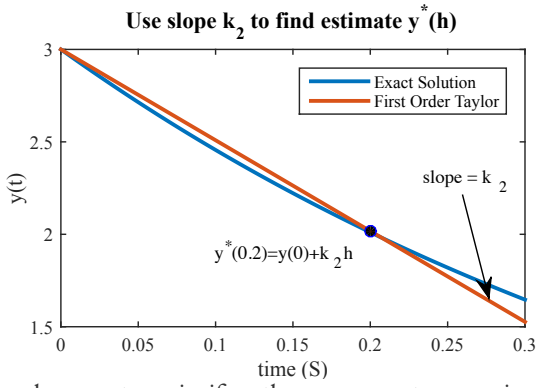
In the graph below, the slope at $t=0$ is called k_1, and the estimate is called $y^*(h)$; in this example $h=0.2$.



This obviously leads to some error in the estimate, and we would like to reduce this error. One way we could do this, conceptually, is to use the derivative at the halfway point between $t=0$ and $t=h=0.2$. The slope at this point ($t=½h=0.1$) is shown below (and is labeled k_2). Note the line (orange) is tangent to the curve (blue) at $t=½h$.



Now if we use this intermediate slope, k_2, as we step ahead in time then we get better estimate, $y^*(h)$, than we did before. On the diagram below the exact value of the solution is $y(0.1)$=2.0110 and the approximation is $y^*(0.1)$=2.0175 for an error of about 0.3% (compared with about 10% error for the first order Runge-Kutta).



This seems like a very nice solution, and obviously generates a significantly more accurate approximation than the first order technique that uses a line with slope, k_1, calculated at $t=0$. The problem is we don't know the exact value of $y(½h)$ so we can't find the exact value of $k_2$the slope at $t=½h$ (Recall that the calculation of the derivative requires knowledge of the value of the function, $y'(t) = -2y(t)$).

What we do instead is use the First Order Runge-Kutta to generate an approximate value for $y(t)$ at $t=½h=0.1$, call it $y_1(½h)$. We then use this estimate to generate k_2 (which will be an approximation to the slope at the midpoint), and then use k_2 to find y*(h). To step from the starting point at $t=0$ to an estimate at $t=h$, follow the procedure below.

$$y'(0) = -2y(0) \qquad \text{expression for derivative at } t = 0$$
$$k_1 = -2y(0) \qquad \text{derivative at } t = 0$$
$$y_1\left(\frac{h}{2}\right) = y(0) + k_1\frac{h}{2} \qquad \text{intermediate estimate of function at } t = h/2$$
$$k_2 = -2y_1\left(\frac{h}{2}\right) \qquad \text{estimate of slope at } t = h/2$$
$$y(h) = y(0) + y'(0)h + y''(0)\frac{h^2}{2} + \cdots \qquad \text{Taylor Series around } t = 0$$
$$y(t) \approx y(0) + y'(0)h \qquad \text{Truncate Taylor Series}$$
$$y^*(h) = y(0) + k_2 h \qquad \text{estimate of } y(h)$$

In general, to go from the estimate $t=t_0$ to an estimate at $t=t_0+h$

$$y'(t_0) = -2y(t_0) \qquad \text{expression for derivative at } t = t_0$$
$$k_1 = -2y^*(t_0) \qquad \text{approximate derivative at } t = t_0$$
$$y_1\left(t_0 + \frac{h}{2}\right) = y^*(t_0) + k_1\frac{h}{2} \qquad \text{intermediate estimate of function at } t = t_0 + h/2$$
$$k_2 = -2y_1\left(t_0 + \frac{h}{2}\right) \qquad \text{estimate of slope at } t = t_0 + h/2$$
$$y(t_0 + h) = y(t_0) + y'(t_0)h + y''(0)\frac{h^2}{2} + \cdots \qquad \text{Taylor Series around } t = t_0$$
$$y(t_0 + h) \approx y(t_0) + y'(t_0)h \qquad \text{Truncated Taylor Series}$$
$$y^*(t_0 + h) = y(t_0) + k_2 h \qquad \text{estimate of } y(t_0 + h)$$

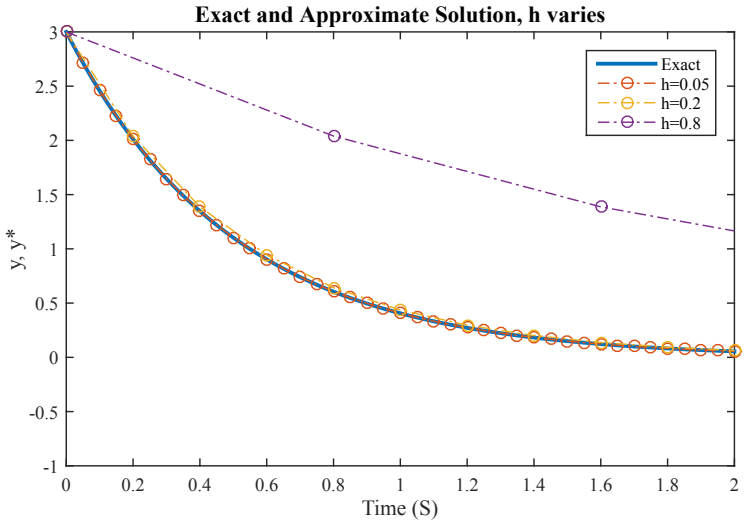
Example 1: Approximation of First Order Differential Equation with No Input Using MATLAB

We can use MATLAB to perform the calculation described above. A simple loop accomplishes this:

```
%% Example 1
% Solve y'(t)=-2y(t) with y0=3, midpoint method
y0 = 3;                   % Initial Condition
h=0.2;                    % Time step
t = 0:h:2;                % t goes from 0 to 2 seconds.
yexact = 3*exp(-2*t)      % Exact solution (in general we won't know this
ystar = zeros(size(t));  % Preallocate array (good coding practice)

ystar(1) = y0;            % Initial condition gives solution at t=0.
for i=1:(length(t)-1)
  k1 = -2*ystar(i)              % Approx for y gives approx for deriv
  y1 = ystar(i)+k1*h/2;         % Intermediate value
  k2 = -2*y1                    % Approx deriv at intermediate value.
  ystar(i+1) = ystar(i) + k2*h; % Approximate solution at next value of y
end
plot(t,yexact,t,ystar);
legend('Exact','Approximate');
```

The MATLAB commands match up easily with the steps of the algorithm. A slight variation of the code was used to show the effect of the size of $h$on the accuracy of the solution (see image below). Note that larger values of h result in poorer approximations, but that the solutions are much better than those obtained with the First Order Runge-Kutta for the same value of h.



Key Concept: Second Order Runge-Kutta Algorithm (midpoint)

For a first order ordinary differential equation defined by

$$\frac{dy(t)}{dt} = f(y(t), t)$$

to progress from a point at $t=t_0$, $y^*(t_0)$, by one time step, h, follow these steps (repetitively).

$$k_1 = f(y^*(t_0),\ t_0) \qquad \text{estimate of derivative at } t = t_0$$

$$y_1\left(t_0 + \frac{h}{2}\right) = y^*(t_0) + k_1\frac{h}{2} \qquad \text{intermediate estimate of function at } t = t_0 + \frac{h}{2}$$

$$k_2 = f\left(y_1\left(t_0 + \frac{h}{2}\right),\ t_0 + \frac{h}{2}\right) \qquad \text{estimate of slope at } t = t_0 + \frac{h}{2}$$

$$y^*\left(t_0 + h\right) = y^*(t_0) + k_2 h \qquad \text{estimate of } y\left(t_0 + h\right)$$

Notes:

- an initial value of the function must be given to start the algorithm.
- see the MATLAB programs on this page for examples.
- this is often refered to as the "midpoint" algorithm for Second Order Runge-Kutta because it uses the slope at the midpoint, k_2.

A First Order Linear Differential Equation with Input

Adding an input function to the differential equation presents no real difficulty. You just need to ensure that you evaluate the function at $t=t_0$ to find k_1, and at $t=t_0+frac12;h$ to find k_2. Consider an input of $cos(4t)$.

$$y'(t) + 2y(t) = \cos(4t) \quad \text{or} \quad y'(t) = -2y(t) + \cos(4t)$$

So

$$y'(t_0) = -2y(t_0) + \cos(4t_0) \qquad \text{expression for derivative at } t = t_0$$

$$k_1 = -2y^*(t_0) + \cos(4t_0) \qquad \text{approximate derivative at } t = t_0$$

$$y_1\left(t_0 + \frac{h}{2}\right) = y^*(t_0) + k_1\frac{h}{2} \qquad \text{intermediate estimate of function at } t = t_0 + h/2$$

$$k_2 = -2y_1\left(t_0 + \frac{h}{2}\right) + \cos\left(t_0 + \frac{h}{2}\right) \qquad \text{estimate of slope at } t = t_0 + h/2$$

$$y^*(t_0 + h) = y(t_0) + k_2 h \qquad \text{estimate of } y(t_0 + h)$$

Note: the exact solution in this case is $y(t) = 2.9e^{-2t} + 0.1\cos(4t) + 0.2\sin(4t)$.

Example 2: Approximation of First Order Differential Equation with Input Using MATLAB

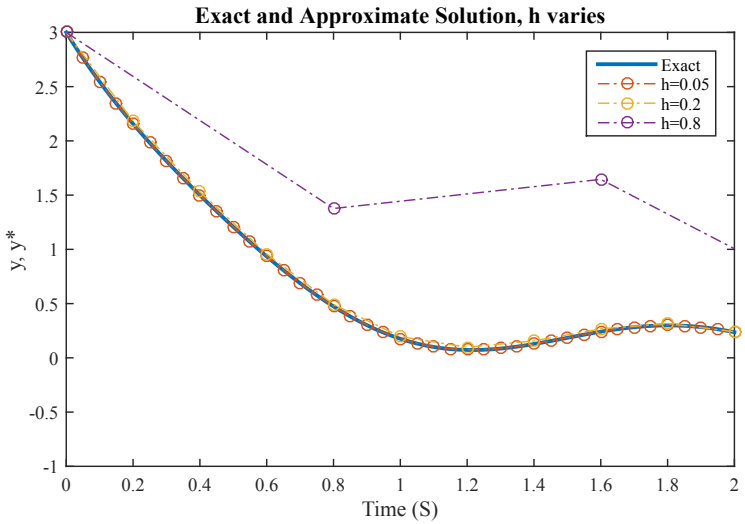
We can use MATLAB to perform the calculation described above. To perform this new approximation all that is

necessary is to change the calculation of k_1 and k_2 using the appropriate value for the time variable (the value of the exact solution is also changed, for plotting).

```
%% Example 2
% Solve y'(t)=-2y(t)-cos(4t) with y0=3
y0 = 3;                    % Initial Condition
h=0.2;                     % Time step
t = 0:h:2;                 % t goes from 0 to 2 seconds.
yexact = 0.1*cos(4*t)+0.2*sin(4*t)+2.9*exp(-2*t);  % Exact Solution
ystar = zeros(size(t));  % Preallocate array (good coding practice)

ystar(1) = y0;             % Initial condition gives solution at t=0.
for i=1:(length(t)-1)
  k1 = -2*ystar(i)+cos(4*t(i));  % Approx for y gives approx for deriv
  y1 = ystar(i)+k1*h/2;      % Intermediate value
  k2 = -2*y1+cos(4*(t(i)+h/2));        % Approx deriv at intermediate value.
  ystar(i+1) = ystar(i) + k2*h; % Approximate solution at next value of y
end
plot(t,yexact,t,ystar);
legend('Exact','Approximate');
```

A modified version of the program (that uses several values of h) generated the plot below. As before, the solution is better with smaller values of h, and the solutions are much better than those obtained with the First Order Runge-Kutta for the same value of h.



A First Order Non-Linear Differential Equation

Using a non-linear differential equation presents no real difficulty. Again, you just need to ensure that you evaluate the function at t=t_0 to find k_1, and at t=t_0+frac12;h to find k_2. Consider

$$\frac{dy(t)}{dt} - y(t)(1-2t) = 0, \qquad y(0) = 1$$

with solution (described here)

$$y(t) = e^{t-t^2}$$

The process follows those that came before

$$\begin{aligned}
y'(t_0) &= y(t_0)(1-2t_0) && \text{expression for derivative at } t = t_0 \\
k_1 &= y^*(t_0)(1-2t_0) && \text{approximate derivative at } t = t_0 \\
y_1\left(t_0 + \frac{h}{2}\right) &= y^*(t_0) + k_1\frac{h}{2} && \text{intermediate estimate of function at } t = t_0 + h/2 \\
k_2 &= -2y_1\left(t_0 + \frac{h}{2}\right)\left(1 - 2\left(t_0 + \frac{h}{2}\right)\right) && \text{estimate of slope at } t = t_0 + h/2 \\
y^*(t_0 + h) &= y(t_0) + k_2 h && \text{estimate of } y(t_0 + h)
\end{aligned}$$

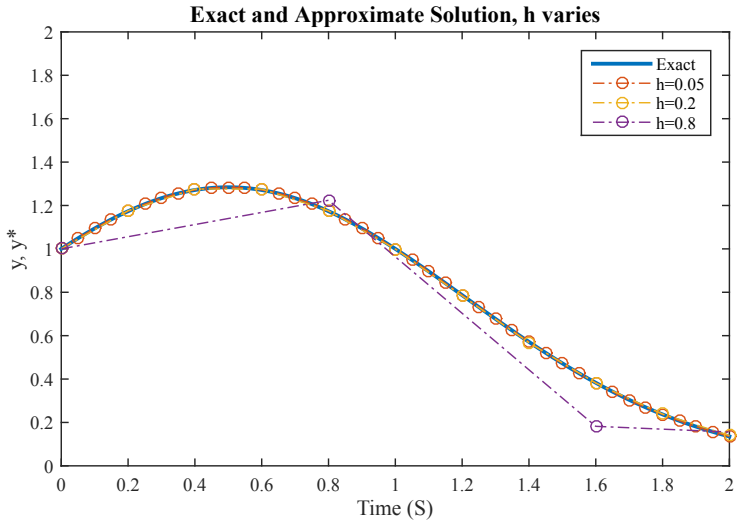
Example 3: Approximation of First Order Nonlinear Differential Equation with Input Using MATLAB

As before, to perform this new approximation all that is necessary is to change the calculation of k_1 and k_2 using the appropriate value for the time variable (the value of the exact solution is also changed, for plotting).

```
%% Example 3
% Solve y'(t)=y(t)(1-2t) with y0=1
y0 = 1;                    % Initial Condition
h=0.2;                     % Time step
t = 0:h:2;                 % t goes from 0 to 2 seconds.
% Exact solution, hard to find in this case (in general we won't have it)
yexact = exp(t-t.^2);
ystar = zeros(size(t));  % Preallocate array (good coding practice)

ystar(1) = y0;             % Initial condition gives solution at t=0.
for i=1:(length(t)-1)
  k1 = ystar(i)*(1-2*t(i));      % Approx for y gives approx for deriv
  y1 = ystar(i)+k1*h/2;          % Intermediate value
  k2 = y1*(1-2*(t(i)+h/2));      % Approx deriv at intermediate value.
  ystar(i+1) = ystar(i) + k2*h; % Approximate solution at next value of y
end
plot(t,yexact,t,ystar);
legend('Exact','Approximate');
```

A modified version of the program (that uses several values of *h)* generated the plot below. As before, the solution is better with smaller values of h, and the solutions are much better than those obtained with the First Order Runge-Kutta for the same value of h.



A Higher Order Linear Differential Equation

If we start with a higher order differential equation

$$\frac{d^3y(t)}{dt} + 4\frac{d^2y(t)}{dt} + 6\frac{dy(t)}{dt} + 4y(t) = \gamma(t) \qquad \gamma(t) = \text{unitstepfunction}$$

$$\left.\frac{d^2y(t)}{dt}\right|_{t=0^+} = 0, \qquad \left.\frac{dy(t)}{dt}\right|_{t=0^+} = -1, \qquad y(0^+) = 0$$

We can rewrite as a matrix differential equation (see previous example if this is unclear)

$$\mathbf{q}'(t) = \begin{bmatrix} q_1'(t) \\ q_2'(t) \\ q_3'(t) \end{bmatrix} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -4 & -6 & -4 \end{bmatrix} \begin{bmatrix} q_1(t) \\ q_2(t) \\ q_3(t) \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} \gamma(t)$$

$$\mathbf{q}'(t) = \mathbf{A}\mathbf{q}(t) + \mathbf{B}\gamma(t)$$

$$\mathbf{A} = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ -4 & -6 & -4 \end{bmatrix}, \qquad \mathbf{B} = \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix}$$

Proceed as before (using matrices in place of scalars where appropriate

$$\mathbf{q}'(t_0) = \mathbf{A}\mathbf{q}(t_o) + \mathbf{B}\gamma(t_0) \qquad \text{expression for derivative at } t = t_0$$

$$\mathbf{k}_1 = \mathbf{A}\mathbf{q}^*(t_o) + \mathbf{B}\gamma(t_0) \qquad \text{approximate derivative at } t = t_0$$

$$\mathbf{q}_1\left(t_0 + \frac{h}{2}\right) = \mathbf{q}^*(t_o) + \mathbf{k}_1\frac{h}{2} \qquad \text{intermediate estimate of function at } t = t_0 + h/2$$

$$\mathbf{k}_2 = \mathbf{A}\mathbf{q}_1\left(t_0 + \frac{h}{2}\right) + \mathbf{B}\gamma\left(t_0 + \frac{h}{2}\right) \qquad \text{estimate of slope at } t = t_0 + h/2$$

$$\mathbf{q}^*\left(t_0 + h\right) = \mathbf{q}^*(t_o) + \mathbf{k}_2 h \qquad \text{estimate of } \mathbf{q}(t_0 + h)$$

Note: the exact solution to this problem is $y(t) = \frac{1}{4} + e^{-t}\left(\cos(t) - \frac{5}{2}\sin(t)\right) - \frac{5}{4}e^{-2t}$, for t≥0.

Example 4: Approximation of Third Order Differential Equation Using MATLAB

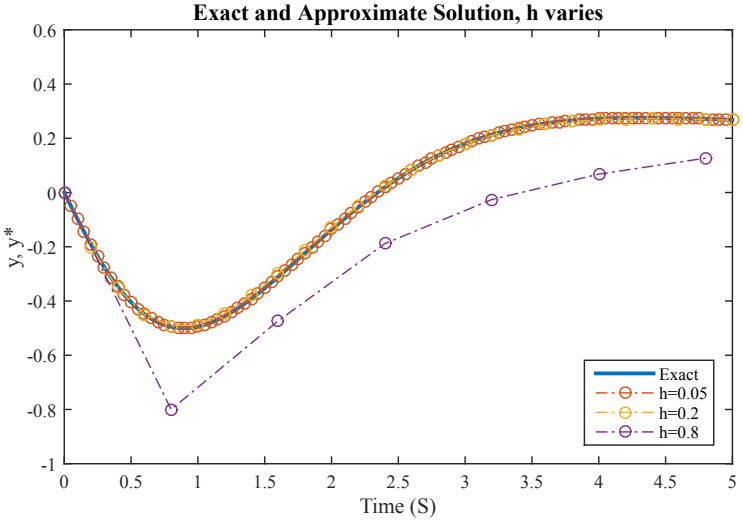
To perform this new approximation all that is necessary is to change the approriate variables from scalars to vectors or matrices, and to define the **A** and **B** matrices. Note that in this case we multiply the **B** matrix by 1 since the input is a unit step ($\gamma(t)$=1 for t≥0). If the input were a sine wave, the sine wave would multiply **B**; if there is no input, it is not necessary to include the **B** matrix at all (it is multiplied by 0).

```
%% Example 4
% Solve y'''(t)+4y''(t)+6y'(t)+4y(t)=gamma(t),  y''(0)=0, y'(0)=-1, y(0)=0
q0 = [0; -1; 0];          % Initial Condition (vector)
h=0.2;                    % Time step
t = 0:h:5;                % t goes from 0 to 2 seconds.
A = [0 1 0; 0 0 1; -4 -6 -4];  % A Matrix
B = [0; 0; 1];                 % B Matrix

yexact = 1/4 + exp(-t).*(cos(t) - 5*sin(t)/2) - 5*exp(-2*t)/4;  % Exact soln
qstar = zeros(3,length(t));  % Preallocate array (good coding practice)

qstar(:,1) = q0;             % Initial condition gives solution at t=0.
for i=1:(length(t)-1)
  k1 = A*qstar(:,i)+B*1;             % Approx for y gives approx for deriv
  q1 = qstar(:,i)+k1*h/2;            % Intermediate value
  k2 = A*q1+B*1;                     % Approx deriv at intermediate value.
  qstar(:,i+1) = qstar(:,i) + k2*h; % Approx solution at next value of q
end
plot(t,yexact,t,qstar(1,:));        % ystar = first row of qstar
legend('Exact','Approximate');
```

A modified version of the program (that uses several values of h) generated the plot below. As expected, the solution is better with smaller values of h, and the solutions are much better than those obtained with the First Order Runge-Kutta for the same value of h.



Second Order Runge-Kutta Method (The Math)

The Second Order Runge-Kutta algorithm described above was developed in a purely ad-hoc way. It seemed reasonable that using an estimate for the derivative at the midpoint of the interval between t_0 and t_0+h (i.e., at $t_0+½h$) would result in a better approximation for the function at t_0+h, than would using the derivative at t_0 (i.e., Euler's Method &emdash; the First Order Runge-Kutta). And, in fact, we showed that the

resulting approximation was better. But, is there a way to derive the Second Order Runge-Kutta from first principles? If so, we might be able to develop even better algorithms.

Aside: Some Useful Math Review

In the following derivation we will use two math facts that are reviewed here. You should be familiar with this from a course in multivariable calculus.

1) If there is a function of two variable, *g(x,y)*, it's Taylor Series about x_0, y_0 is given by:

$$g(x_0+\Delta x, y_0+\Delta y) = g(x_0,y_0) + \left.\frac{\partial g(x,y)}{\partial x}\right|_{x_o,y_0}\Delta x + \left.\frac{\partial g(x,y)}{\partial y}\right|_{x_o,y_0}\Delta y + \cdots$$
$$= g + g_x\Delta x + g_y\Delta y + \cdots$$

where Δx is the increment in the first dimenstion, and Δy is the increment in the second dimension. In the last line we use a shorthand notation that removes the explicit functional notation, and also represents the partial derivative of g with respect to x as g_x (and likewise for g_y).

2) If z is a function of two variables *z=g(x,y)*, where *x=r(t)* and *y=s(t)*, then by the chain rule for partial derivatives

$$\frac{dz}{dt} = \frac{\partial g}{\partial r}\frac{dr}{dt} + \frac{\partial g}{\partial s}\frac{ds}{dt}$$

In particular if

$$y'(t) = \frac{dy(t)}{dt} = f(y(t),t) = f$$

then

$$y''(t) = \frac{df(y(t),t)}{dt} = \frac{\partial f}{\partial t}\frac{dt}{dt} + \frac{\partial f}{\partial y}\frac{dy}{dt} = \frac{\partial f}{\partial t} + \frac{\partial f}{\partial y}\frac{dy}{dt} = f_t + f_y f$$

To start, recall that we are expressing our differential equation as

$$\frac{dy(t)}{dt} = y'(t) = f(y(t),t)$$

Now define two approximations to the derivative

$$k_1 = f(y^*(t_0), t_0)$$
$$k_2 = f(y^*(t_0) + \beta h k_1, t_0 + \alpha h)$$

In all cases α and β will represent fractional values between 0 and 1. These equation state that k_1 is the approximation to the derivative based on the estimated value of *y(t)* at *t=t0* (i.e., *y*(t0)*) and the time at t_0. The value of k_2 is based upon the estimated value, *y*(t0)*, plus some fraction of the step size, βh, times the slope k_1, and the time at $t_0+\alpha h$ (i.e., some time between t_0 and t_0+h). In the method described previously $\alpha=\beta=½$, but other choices are possible.

To update our solution with the next estimate of *y(t)* at t_0+h we use the equation

$$y^*(t_0+h) = y^*(t_0) + h(ak_1 + bk_2) \qquad \text{update equation}$$

This equation states that we get the value of *y*(t0+h)* from the value of *y*(t0)* plus the time step, h, multiplied by a slope that is a weighted sum of k_1 and k_2. In the method described previously $a=0$ and $b=1$, so we used only the second estimate for the slope. *(Note that Euler's Method (First Order Runge-Kutta) is a special case of this method with a=1, b=0, and α and β don't matter because k_2 is not used in the update equation.)*

Our goal now is to determine, from first principles, how to find the values a, b, α and β that result in low error. Starting with the update equation (above) we can substitue the previously given expressions for k_1 and k_2 which yields

$$y^*(t_0+h) = y^*(t_0) + h\left(af\left(y^*(t_0),t_0\right) + bf\left(y^*(t_0)+\beta h k_1,\ t_0+\alpha h\right)\right)$$

We can use a two-dimensional Taylor Series (where the increment in the first dimension is $\beta h k_1$, and the increment in the second dimension is αh) to expand the rightmost term.

$$
\begin{aligned}
f\left(y^*(t_0)+\beta h k_1,\, t_0+\alpha h\right) &= f + f_y \beta h k_1 + f_t \alpha h + \cdots \\
&= f + f_y f \beta h + f_t \alpha h + \cdots
\end{aligned}
$$

In the last line we used the fact the $k_1 = f$. The ellipsis denotes terms that are second order or higher. Substituting this into the previous expression for $y^*(t_0+h)$ and rearranging we get

$$
\begin{aligned}
y^*(t_0+h) &= y^*(t_0) + h\left(af + b\left(f + f_y f \beta h + f_t \alpha h + \cdots\right)\right) \\
&= y^*(t_0) + haf + hbf + h^2 b\beta f_y f + h^2 f_t b\alpha + \cdots \\
&= y^*(t_0) + h(a+b)f + h^2 f_t b\alpha + h^2 b\beta f_y f + \cdots
\end{aligned}
$$

The ellipsis was multiplied by h between the first or second line and now represents terms that are third order or higher.

To finish we compare this approximation with the expressionfor a Taylor Expansion of the exact solution (going from the first line to the second we used the chain rule for partical derivatives). In this expression the ellipsis represents terms that are third order or higher.

$$
\begin{aligned}
y(t_0+h) &= y(t_0) + h\,y'(t)\big|_{t_0} + \frac{h^2}{2}\,y''(t)\big|_{t_0} + \cdots \\
&= y(t_0) + hf + \frac{h^2}{2}\left(f_t + f_y f\right) + \cdots \\
&= y(t_0) + hf + \frac{h^2}{2} f_t + \frac{h^2}{2} f_y f + \cdots
\end{aligned}
$$

Comparing this expression with our final expression for the approximation,

$$
y^*(t_0+h) = y^*(t_0) + hf(a+b) + h^2 f_t b\alpha + h^2 b\beta f_y f + \cdots
$$

we see that they agree up to the error terms (third order and higher) represented by the ellipsis if we define the constants, a, b, α and β such that

$$
\begin{aligned}
a + b &= 1 \\
b\alpha &= \frac{1}{2} \\
b\beta &= \frac{1}{2}
\end{aligned}
$$

This system is underspecified, there are four unknowns, and only three equations, so more than one solution is possible. Clearly the choice we made $a=0$ and $b=1$ and $\alpha=\beta=\frac{1}{2}$ is one of these solutions. Because all of the terms of the approximation are equal to the terms in the exact solution, up to the error terms, the local error of this method is therefore $O(h^3)$ ($O(h^2)$ globally, hence the term "second order" Runge-Kutta).

Key Concept: Error of Second Order Runge Kutta

The global error of the Second Order Runge-Kutta algorithm is $O(h^2)$.

Another Form of the Second Order Runge-Kutta Method

Another common choice for the coefficients of the algorithm are $a=b=\frac{1}{2}$ and $\alpha=\beta=1$. Before giving an example, let's figure out, intuitively what this is doing. We start with our equations for k_1, k_2, and $y^*(t_0+h)$.

$$
\begin{aligned}
k_1 &= f(y^*(t_0), t_0) \\
k_2 &= f(y^*(t_0) + \beta h k_1, t_0 + \alpha h) \\
y^*(t_0+h) &= y^*(t_0) + h(ak_1 + bk_2)
\end{aligned}
$$

Now put in our chosen constants

$$
\begin{aligned}
k_1 &= f(y^*(t_0), t_0) \\
k_2 &= f(y^*(t_0) + h k_1, t_0 + \alpha h) \\
y^*(t_0+h) &= y^*(t_0) + h\left(\frac{k_1 + k_2}{2}\right)
\end{aligned}
$$

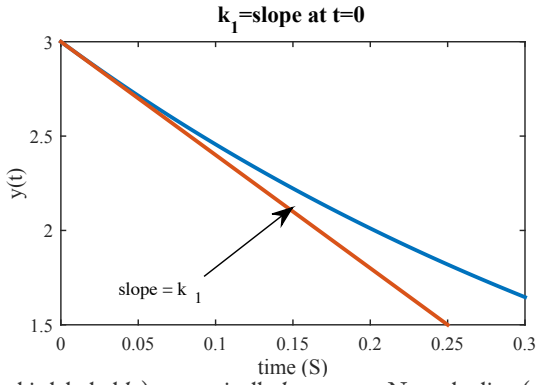
In terms of our algorithm description

$$
\begin{aligned}
k_1 &= f(y^*(t_0),\ t_0) && \text{estimate of derivative at } t = t_0 \\
y_1\left(t_0 + h\right) &= y^*(t_0) + k_1 h && \text{intermediate estimate of function at } t = t_0 + h \\
k_2 &= f\left(y_1\left(t_0 + h\right),\ t_0 + h\right) && \text{estimate of slope at } t = t_0 + h \\
y^*\left(t_0 + h\right) &= y^*(t_0) + h\left(\frac{k_1 + k_2}{2}\right) && \text{estimate of y } (t_0{+}h) \text{ using average of slopes}
\end{aligned}
$$

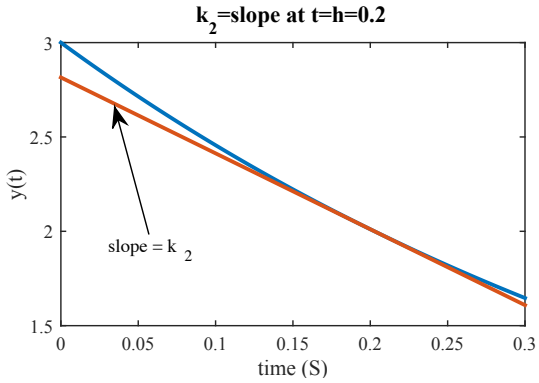
We will again use the the differential equation

$$\frac{dy(t)}{dt} + 2y(t) = 0 \quad \text{or} \quad \frac{dy(t)}{dt} = -2y(t)$$

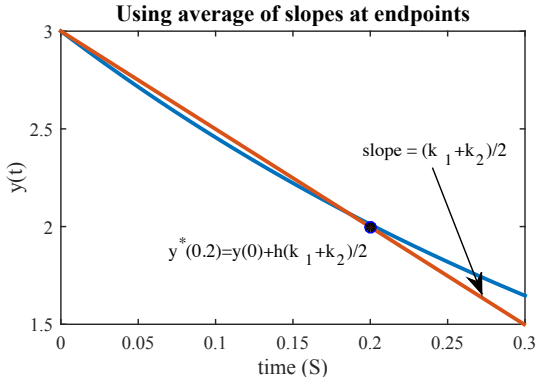
with the initial condition set as $y(0)=3$ with exact solution $y(t)=3e^{-2t}$, $t\geq 0$. Conceptually what we are trying to do is to find the derivative at the the beginning of our time step (t=0, for the first step) and at the end of our time step, $t=h=0.2$. We call the slope at the beginning k_1; numerically $k_1=-6$. The slope at this point is shown below (and is labeled k_1). Note the line (orange) is tangent to the curve (blue) at $t=0$.



The slope at the end point is shown below (and is labeled k_2); numerically $k_2=-4.0219$. Note the line (orange) is tangent to the curve (blue) at $t=h$; in this example $h=0.2$.



We now take the average of these two slopes (-5.0110) and use it to move forward by one time step.



$$y^*(h) = y^*(0) + h\left(\frac{k_1 + k_2}{2}\right) = 3 + 0.2\left(-5.0110\right) = 1.9978$$

This is very close to the exact answer, y(0.2)=2.0110, with an error of about 0.6%. This is close to that of the midpoint method (it is slightly worse, but that is becaus of the specific problem chosen, that won't generally be the case), and much better than that of the first order algorithm.

Key Concept: The Second Order Runge-Kutta Algorithm (endpoint)

For a first order ordinary differential equation defined by

$$\frac{dy(t)}{dt} = f(y(t), t)$$

to progress from a point at $t=t_0$, $y^*(t_0)$, by one time step, h, follow these steps (repetitively).

$$\begin{aligned} k_1 &= f(y^*(t_0),\ t_0) && \text{estimate of derivative at } t = t_0 \\ y_1(t_0+h) &= y^*(t_0) + k_1 h && \text{approximate enpoint value at } t = t_0 + h \\ k_2 &= f(y_1(t_0+h),\ t_0+h) && \text{estimate of slope at enpoint, } t = t_0 + h \\ y^*(t_0+h) &= y^*(t_0) + h\frac{(k_1+k_2)}{2} && \text{estimate of } y(t_0+h) \text{ using average slope} \end{aligned}$$

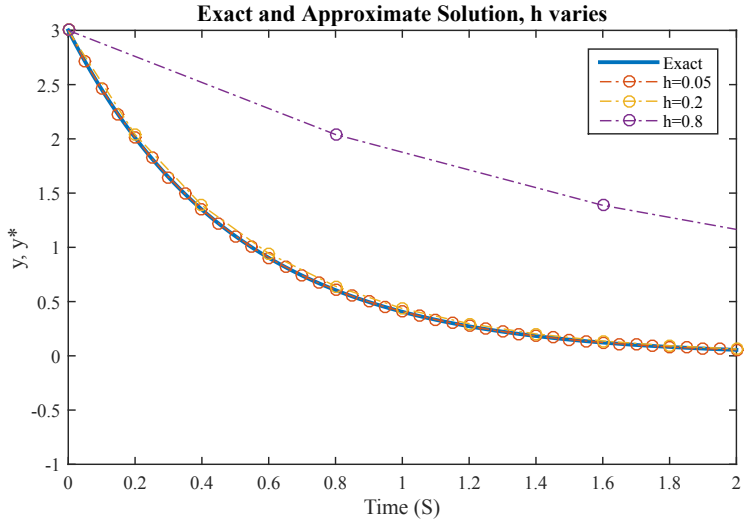
Example 5: Repeat Example 1 with endpoints method

The following MATLAB code repeats Example 1 (a linear differential equation with no input). Example 1 used the "midpoint" method, this example uses the "endpoint" method. The MATLAB commands match up easily with the steps of the algorithm (only the lines that calculate **y1** and **k2** have changed from the midpoint method).

```
%% Example 5
% Solve y'(t)=-2y(t) with y0=3, endpoint method
y0 = 3;                    % Initial Condition
h=0.2;                     % Time step
t = 0:h:2;                 % t goes from 0 to 2 seconds.
yexact = 3*exp(-2*t)       % Exact solution (in general we won't know this
ystar = zeros(size(t));  % Preallocate array (good coding practice)

ystar(1) = y0;             % Initial condition gives solution at t=0.
for i=1:(length(t)-1)
  k1 = -2*ystar(i)                % Approx for y gives approx for deriv
  y1 = ystar(i)+k1*h;             % enpoint approximation
  k2 = -2*y1                      % Approx deriv at endpoint.
  ystar(i+1) = ystar(i) + h*(k1+k2)/2; % Approx solution at next value of y
end
plot(t,yexact,t,ystar);
legend('Exact','Approximate');
```

A slight variation of the code was used to show the effect of the size of $h$on the accuracy of the solution (see image below). Note that larger values of h result in poorer approximations, but that the solutions are much better than those obtained with the First Order Runge-Kutta for the same value of h (and appears similar to results obtained with the mid-point method).



The other examples can be coded for the endpoint method in a similar way.

Moving on

With a little more work we can develop an algorithm that is accurate to even higher order. The next page describes just such a method.

References

Comments? Questions? Suggestions? Corrections?
Erik Cheever Department of Engineering Swarthmore College