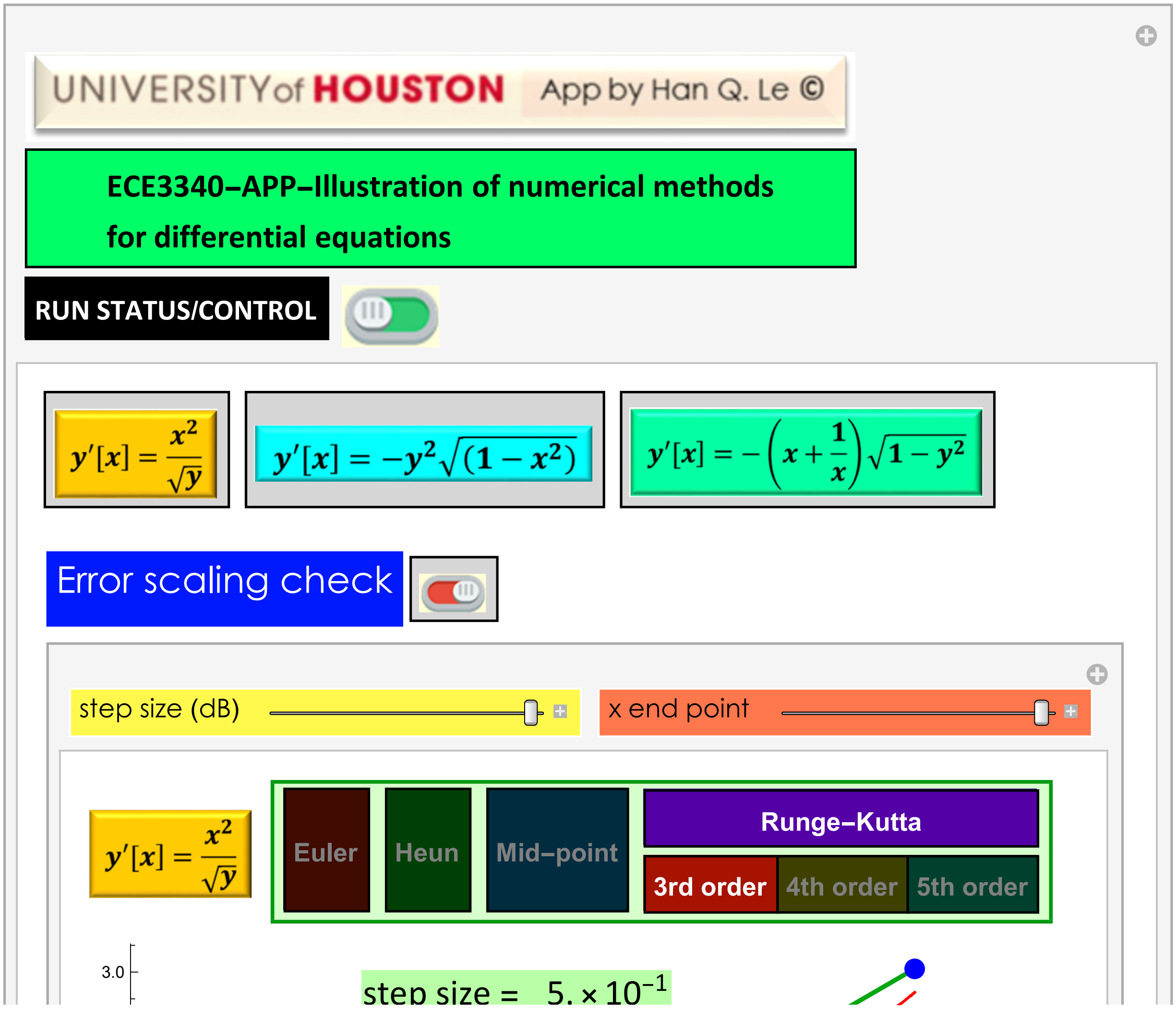
Review: Numerical Methods for Differential Equations

ECE Generic and 2331

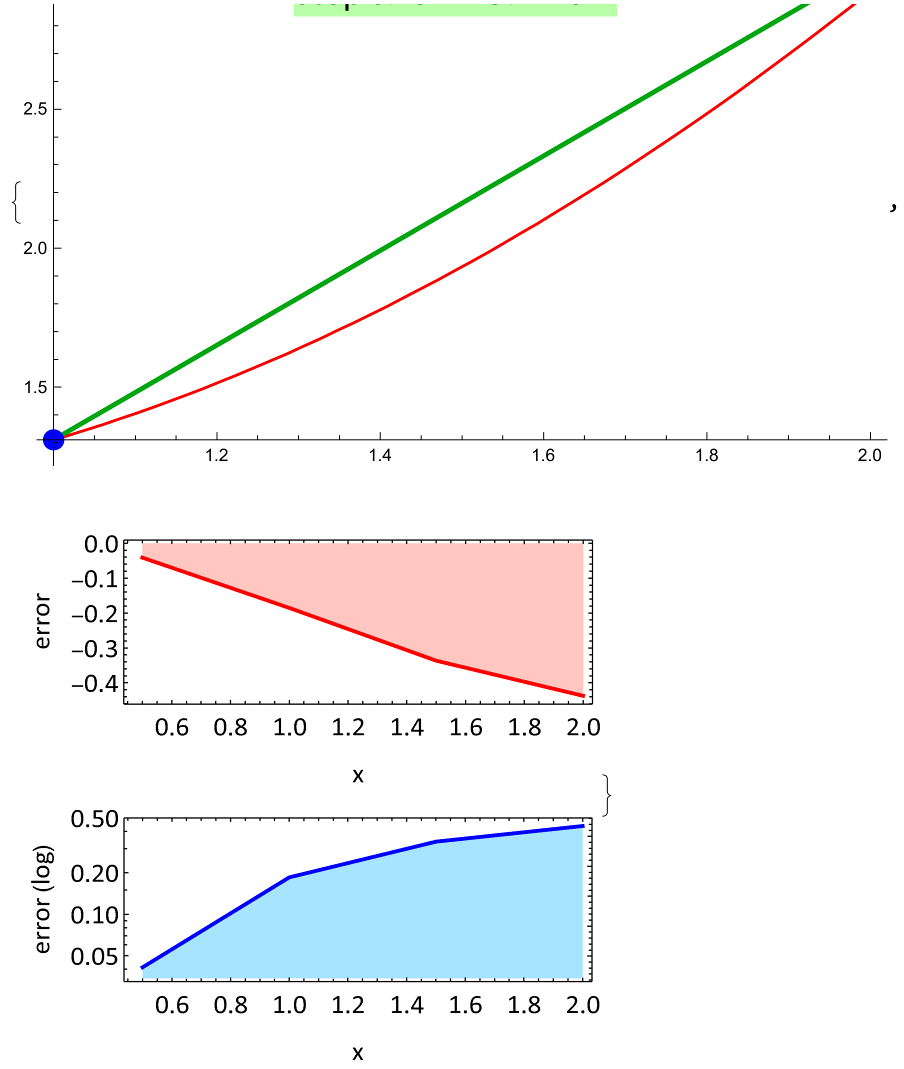
Han Q. Le(c)

Segment 1 - basic concept

The app below is used throughout this lecture. It is best to run it outside this notebook (cut and paste into a new notebook) or just download it from the website and run it in parallel with this notebook.


UNIVERSITY of HOUSTON
App by Han Q. Le ©
ECE3340–APP–Illustration of numerical methods for differential equations
RUN STATUS/CONTROL
$y'[x] = \frac{x^2}{\sqrt{y}}$
$y'[x] = -y^2\sqrt{(1-x^2)}$
$y'[x] = -\left(x+\frac{1}{x}\right)\sqrt{1-y^2}$
Error scaling check
step size (dB)
x end point
$y'[x] = \frac{x^2}{\sqrt{y}}$
Euler
Heun
Mid–point
Runge–Kutta
3rd order
4th order
5th order
3.0
step size = $5. \times 10^{-1}$

Out[]=



Text → ON AA

Numerical methods for differential equations
Select a DE. Click on buttons of various methods
Click on Error scaling bar to see the error of each method. Then click save to compare errors

1. Introductory example

1.1 Discussion

Suppose we a problem like this: $\frac{dy}{dx} = f[x; y]$.

Let’s start out with one point: x_0 and a known initial or boundary value y_0. Then:

$$\left.\frac{dy}{dx}\right|_{x_0} = f[x_0; y_0]$$

How do we obtain a numerical estimate for the next point, and eventually for all other points over the range x of interest? For the next point x_1, we can do this: (see Taylor’sseries below)

$$y_1 \sim y_0 + \left.\frac{dy}{dx}\right|_{x_0} (x_1 - x_0) = y_0 + h\ f[x_0; y_0]\text{ ; where } x_1 = x_0 + h$$

then, this can be repeated for the next point and so on.

This simplistic approach is actually known as the Euler’s method. Let’s consider a simple example:

$$\frac{dy}{dx} = f(x, y) = \frac{x^2}{\sqrt{y}}$$

Clearly, this is solvable because we can write: $\sqrt{y}\ dy = x^2\ dx$

and integrate both side:

$$\int \sqrt{y}\ dy == \int x^2\ dx$$

$$\frac{2\,y^{3/2}}{3} == \frac{x^3}{3}$$

In fact, we can just ask *Mathematica* to do it, so that it includes an arbitrary constant:

$$\text{DSolve}\left[y'[x] == \frac{x^2}{\sqrt{y[x]}}, y[x], x\right]$$

$$\left\{\left\{y(x) \to \frac{\left(3\,c_1 + x^3\right)^{2/3}}{2^{2/3}}\right\}\right\}$$

But we will use this example so that we can compare the various numerical methods to the exact result.

1.2 Numerical illustration

We will choose an initial condition so that we can test the simple approach above. W

$$y(x) \to \frac{\left(3\,c_1 + x^3\right)^{2/3}}{2^{2/3}}\ /.\ x \to 0$$

$$\text{y[1]} \to \frac{(1 + 3\,\text{C[1]})^{2/3}}{2^{2/3}}$$

We will let y(0)=1, so that $c_1 = 2/3$. We have the APP below (click on Euler).

Observe for discussion (next section):

- How does the error change vs. step size h?
- What are the causes of error?

Concept review: types of error

"To err is human"

"... and so is computer!"

Error: how does it happen?

Classification of error types

1. **Truncation error**: Technically, this name refers to errors that are caused by *truncating* or stopping a numerical procedure before it has really *converged* or stopped changing. Many numerical procedures would have to be run forever in order to truly converge if you could record intermediate values with infinite precision. However, in practice there is always some sort of stopping criterion, and the difference between the final intermediate value and the true value represents the truncation error associated with the procedure.

2. **Inherent errors**: Errors resulting from the inaccuracies of the measurement process itself, or possibly from inaccuracies in the mathematical model that the analyst uses to represent the physical world, are lumped together and called inherent errors.

3. **Round-off Error**: Since all computers have a finite word length and use the binary number system, most decimal numbers cannot be represented with complete accuracy in a computer. Note that this is true whether the numbers are stored in fixed-point format or floating-point format. Whether the infinite binary string that represents the true value of the decimal number is truncated (*i.e.*, chopped off) or rounded somehow to produce an internal representation, this type of error is simply called round-off error.

4. **Propagated error**: This is error in succeeding steps of a process that is *propagated forward* from a previous inaccurate result. This can result in a procedure that never converges or one which “blows up.”

2. Euler's method and error

2.1 Error consideration

The method we use above is actually the basic Euler’s method. We also see that it can have significant error. The larger step size h is, the larger is the error. The issue is how to determine the magnitude of this error and how it varies vs. h. In order to do this, we will review Taylor’s series theorem

2.1.1 Review of Taylor’s series

Taylor’s series theorem states that if a function is continuous and differentiable of all orders in an interval $[x, x_0]$, then:

$$f[x] = f[x_0] + f'[x_0]\,(x - x_0) + \frac{1}{2!} f''[x_0]\,(x - x_0)^2 + \frac{1}{3!} f'''[x_0]\,(x - x_0)^3 + \ldots + \frac{1}{n!} f^{(n)}[x_0]\,(x - x_0)^n + \ldots$$

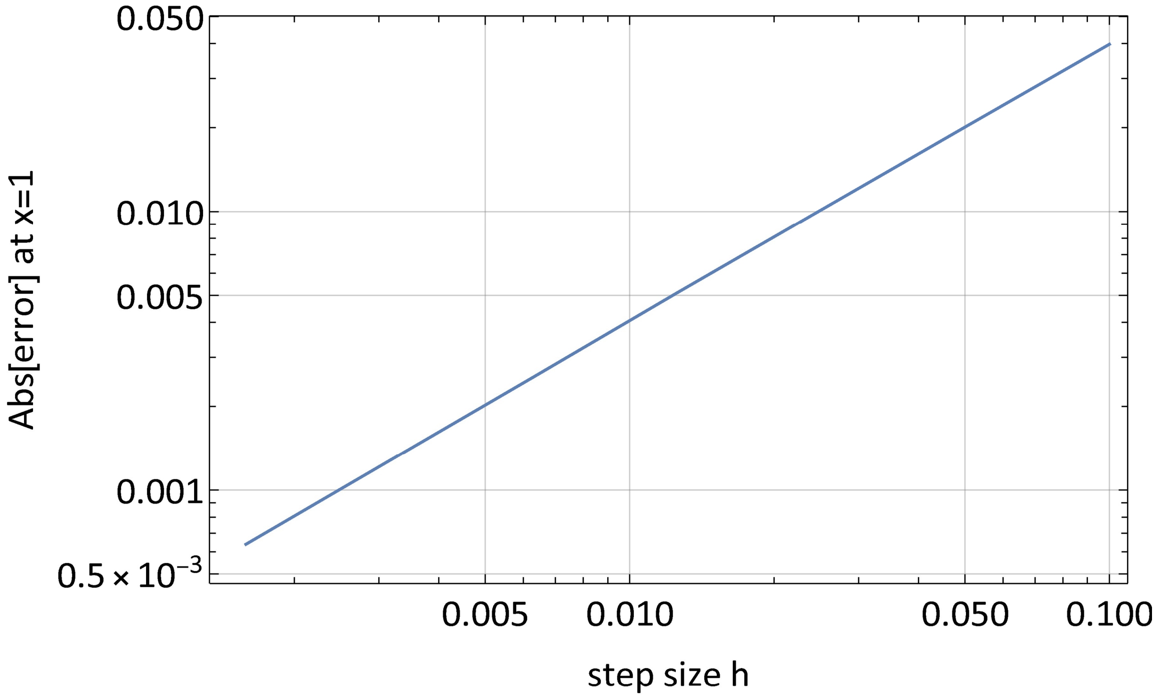
or, we can write, let $x - x_0 \equiv h$

$$f[x] = f[x_0] + f'[x_0]\,h + \frac{1}{2!} f''[x_0]\,h^2 + \ldots + \frac{1}{n!} f^{(n)}[x_0]\,h^n + O\left[h^{n+1}\right]$$

Here we introduce an important notation: $O\left[h^{n+1}\right]$, which is used to indicate that the sum of remaining terms (for n-> infinity), varies on the order of h^{n+1}.
This is important, because if we stop at the nth term when computing f[x], we can set an upper limit how the remaining error scales as a function of step size. For example, let n+1=5, then if we reduce step size a factor of 2, the error will be reduce by $\frac{1}{2^5} = 1/32$.

2.1.2 Apply to Euler’s method

If we go back to the APP above and click of error check, observe how the error at one point (x=1) varies as a function of step size. What can be seen is that it scales as $O[h]$.



We can see why. here is what is computed: $y_{i+1} = y_i + h\ f[x_i;\, y_i]$

where as $\hat{y}_{i+1} = \hat{y}_i + h\ f[x_i;\, \hat{y}_i] + O\left[h^2\right]$

The $O\left[h^2\right]$ principally includes the term of second order:

$$O\left[h^2\right] = \frac{1}{2!}\, y''[x_i]\, h^2 + O\left[h^3\right]$$

This error is at x_{i+1}, but it will propagate to the next and the next and so on...

Let $\hat{y}_i$ indicate the exact value of y, to be distinguished from the estimate value y_i. We can expand:

$$y_{i+1} - \hat{y}_{i+1} = y_i - \hat{y}_i + h\ (f[x_i;\, y_i] - f[x_i;\, \hat{y}_i]) - O\left[h^2\right]$$

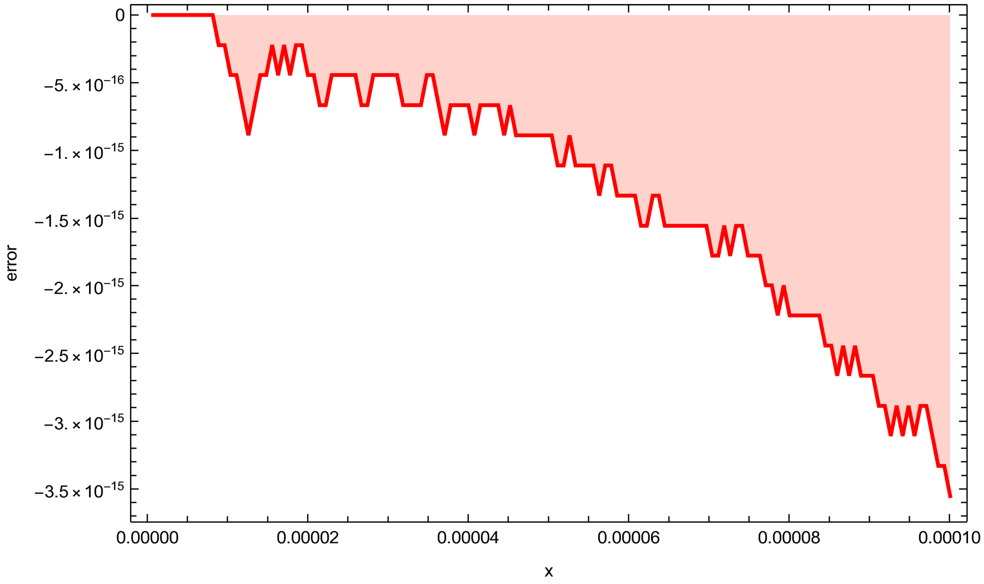
$$y_{i+1} - \hat{y}_{i+1} = \Delta y_i + h\left(\frac{\partial}{\partial y} f[x_i;\, y_i]\, \Delta y_i + O'\left[\Delta y^2\right]\right) - O\left[h^2\right]$$

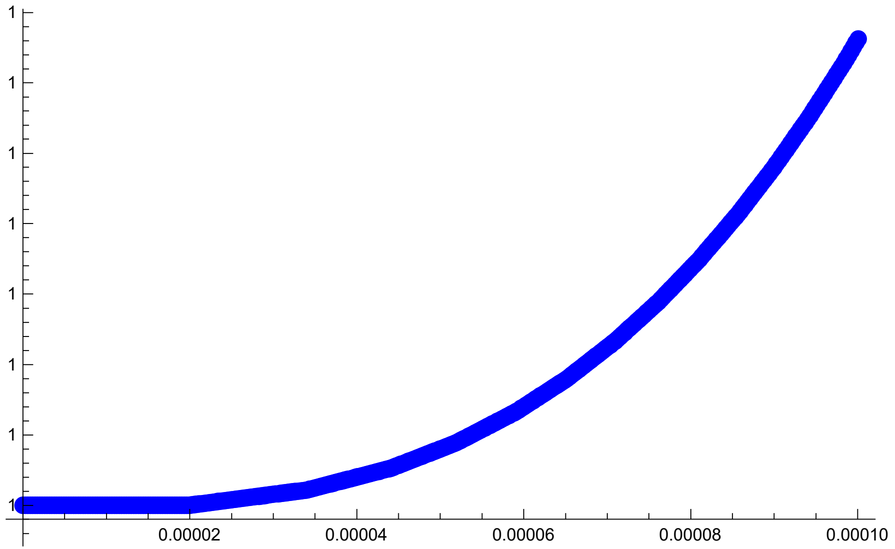
We see that the error Δy_i becomes a sum whose length increases proportional with inverse h: the smaller the step size is, the more points there are and the larger the accumulation. Hence, the error doesn’t scale as $O\left[h^2\right]$, but worse, only as $O[h]$

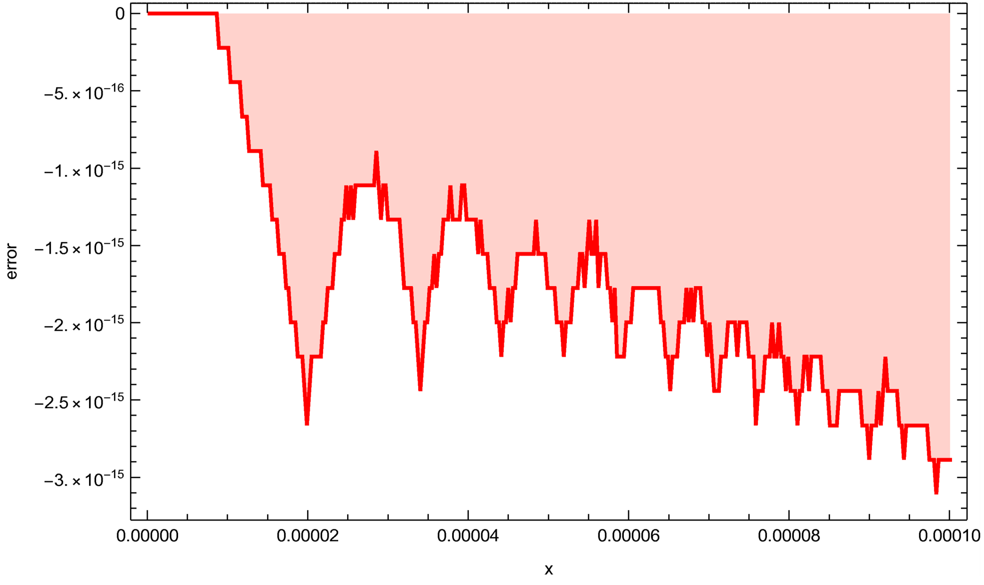
2.2 More general consideration of error

From the APP, we also see that:

- smaller step size: better approximation initially, but
- too small step size: computer round-off errors (truncation, loss of precision)


0
−5. × 10^−16
−1. × 10^−15
−1.5 × 10^−15
−2. × 10^−15
−2.5 × 10^−15
−3. × 10^−15
−3.5 × 10^−15
error
0.00000
0.00002
0.00004
0.00006
0.00008
0.00010
x


1
1
1
1
1
1
1
1
0.00002
0.00004
0.00006
0.00008
0.00010

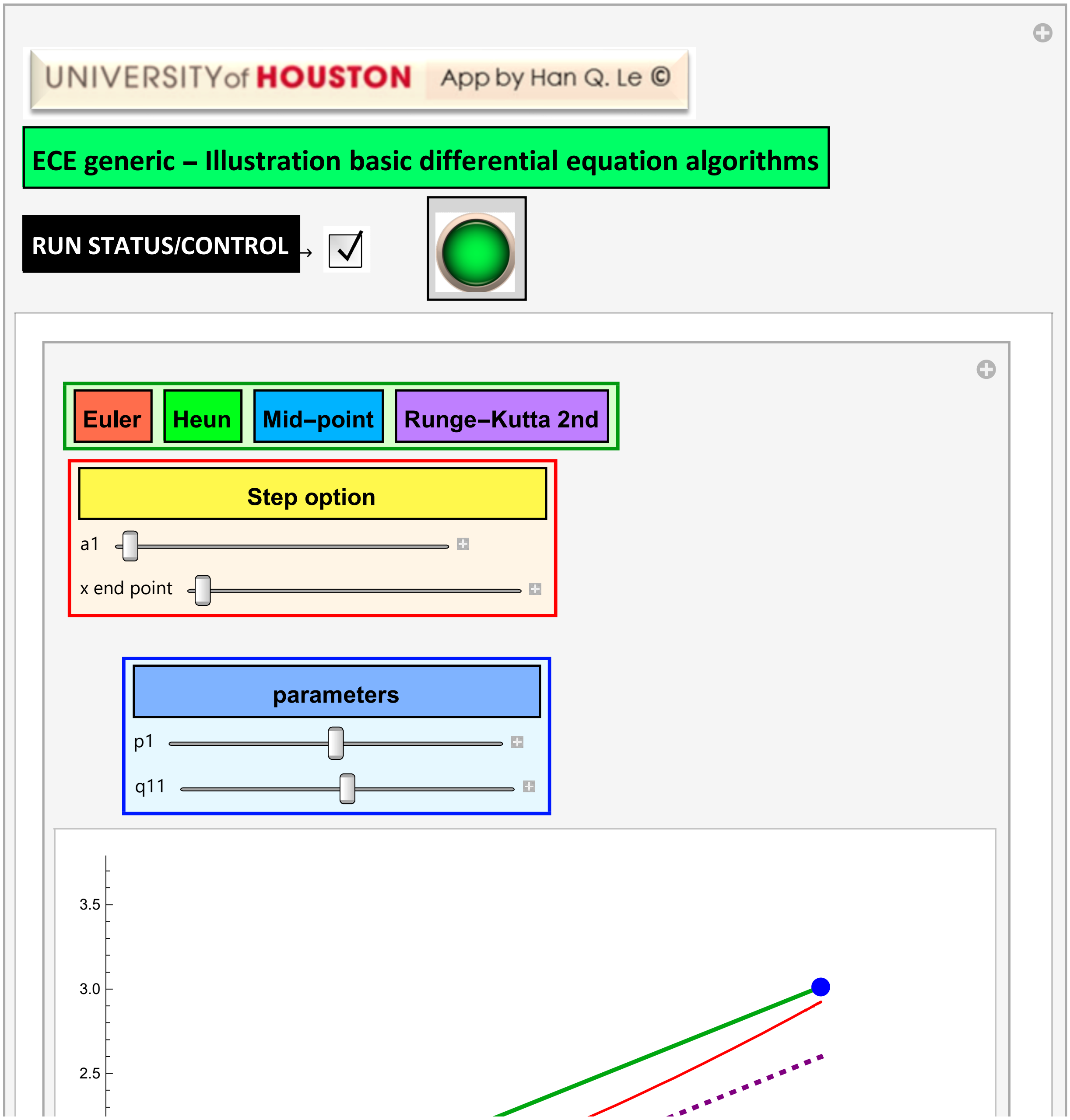

0
−5. × 10^−16
−1. × 10^−15
−1.5 × 10^−15
−2. × 10^−15
−2.5 × 10^−15
−3. × 10^−15
error
0.00000
0.00002
0.00004
0.00006
0.00008
0.00010
x

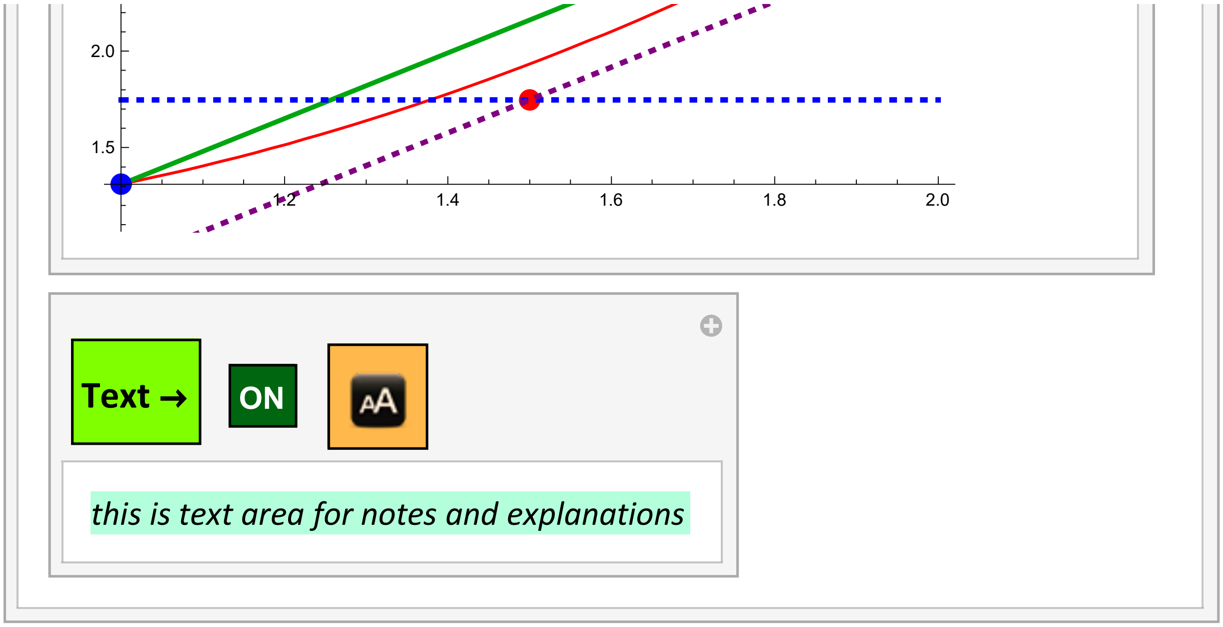
The general principle is that the algorithm should be designed such that the order of error, i. e. $O[h^{n+1}]$ should be known, which means that all other terms of higher order should be such that they are canceled out. This way, regardless of the error magnitude, we can “cap” the error with a scaling limit. This leads to improved and generalized method such as Runge-Kutta.

3. Improved Euler's method

3.1 Overview of approaches to improve

See the APP on method illustration.





this is text area for notes and explanations

3.2 Heun's method

To improve on the Euler's method, the issue is the error for calculating the derivative at just one side. This leads to an accumulation that can scale as $O[h]$ instead of second-order derivative error of $O\left[h^2\right]$.
An improvement is to use the derivative as the average of both end points. In other words:

$$y_{i+1} \;=\; y(x_i) + h\,\frac{f(x_i,y_i) + f(x_{i+1},y_{i+1})}{2}$$

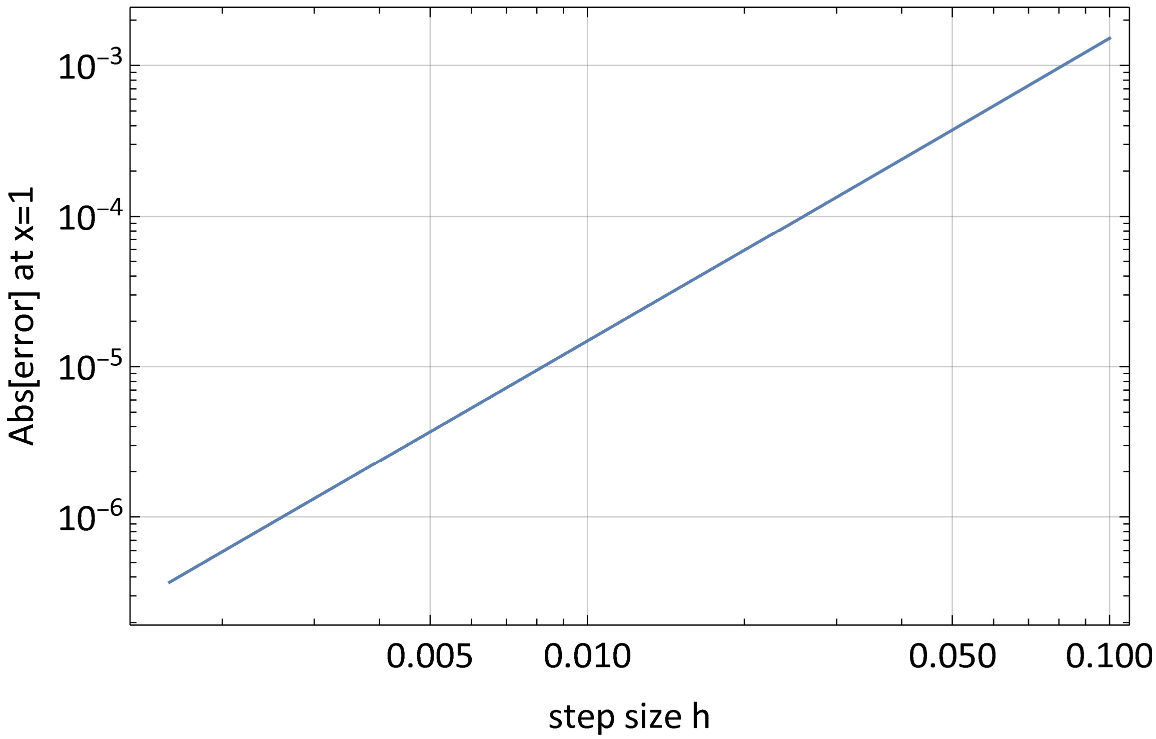
But if this is implemented exactly, it means that y_{i+1} has to be solved, which, while feasible with find root method, will cost extra computing processes. Heun’s method is that there is no need for y_{i+1} in $f(x_{i+1},\, y_{i+1})$ to be self-consistent, but just an approximation:

$$y_{i+1}{}^{(0)} \;=\; y(x_i) + h\, f(x_i,\, y_i) \text{ ; and then:}$$

$$y_{i+1} \;=\; y(x_i) + h\,\frac{f(x_i,y_i) + f\left(x_{i+1},y_{i+1}{}^{(0)}\right)}{2}$$

This algorithm costs only one extra-step (see APP illustration above), hence Heun's method is also known as two-step improved Euler’s method. This improvement can bring the residual errors to the order of $O\left[h^2\right]$ as shown below.

3.3 Heun’s method error scaling



3.4 Midpoint method (modified Euler)

Similar to Heun's method, we can also use two steps instead of just one step, but this time, instead of averaging the derivatives on both ends, we just guess a value at midpoint, and use the derivative at that point. The first step is to get "mid-point" value

$$y_{i+1/2} = y(x_i) + (h/2)\, f(x_i,\, y_i)$$

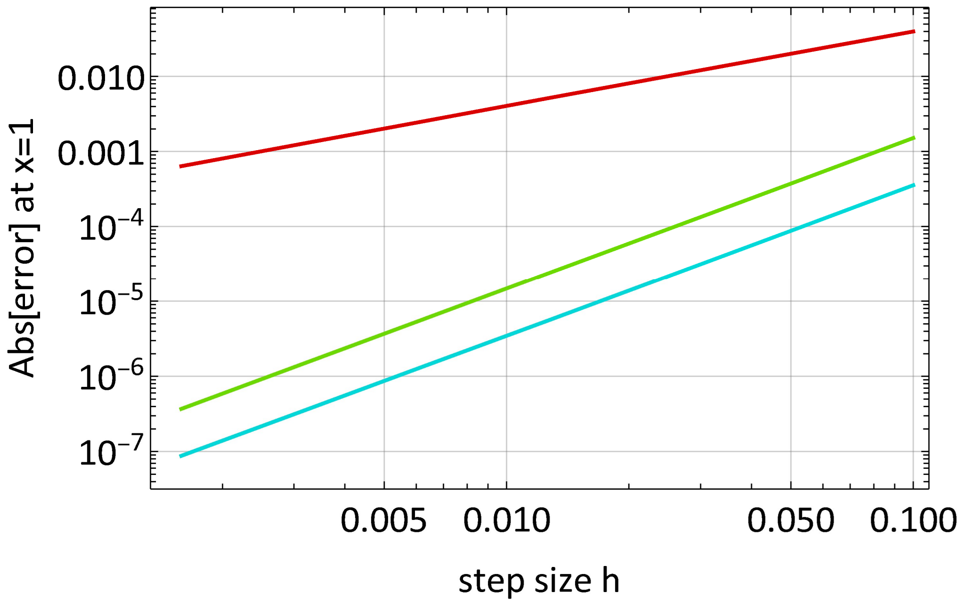
The next step: use this to get "mid-point" estimate of derivative and then the next step:

$$y_{i+1} \;\; = y(x_i) + h\, f(x_{i+1/2},\, y_{i+1/2}).$$

Use the APP above to see how it works.

3.5 Error comparison

Below is an error comparison of all three methods discussed previously.

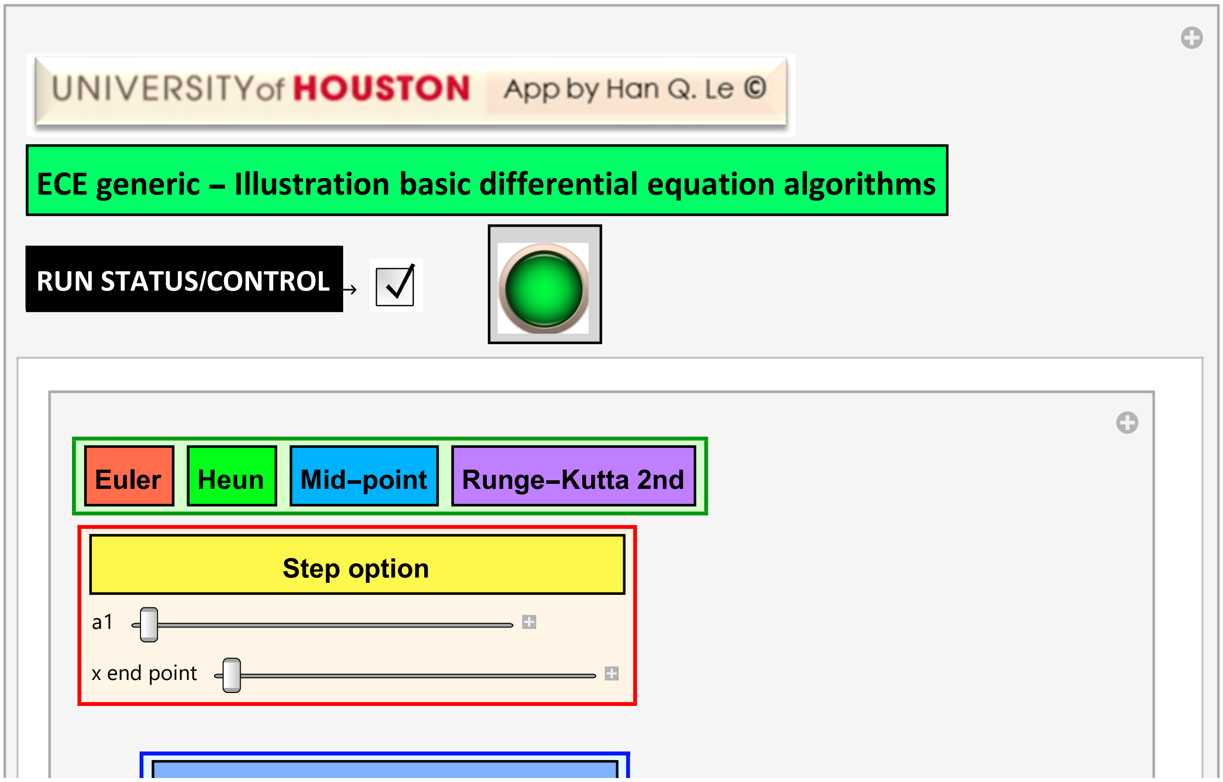


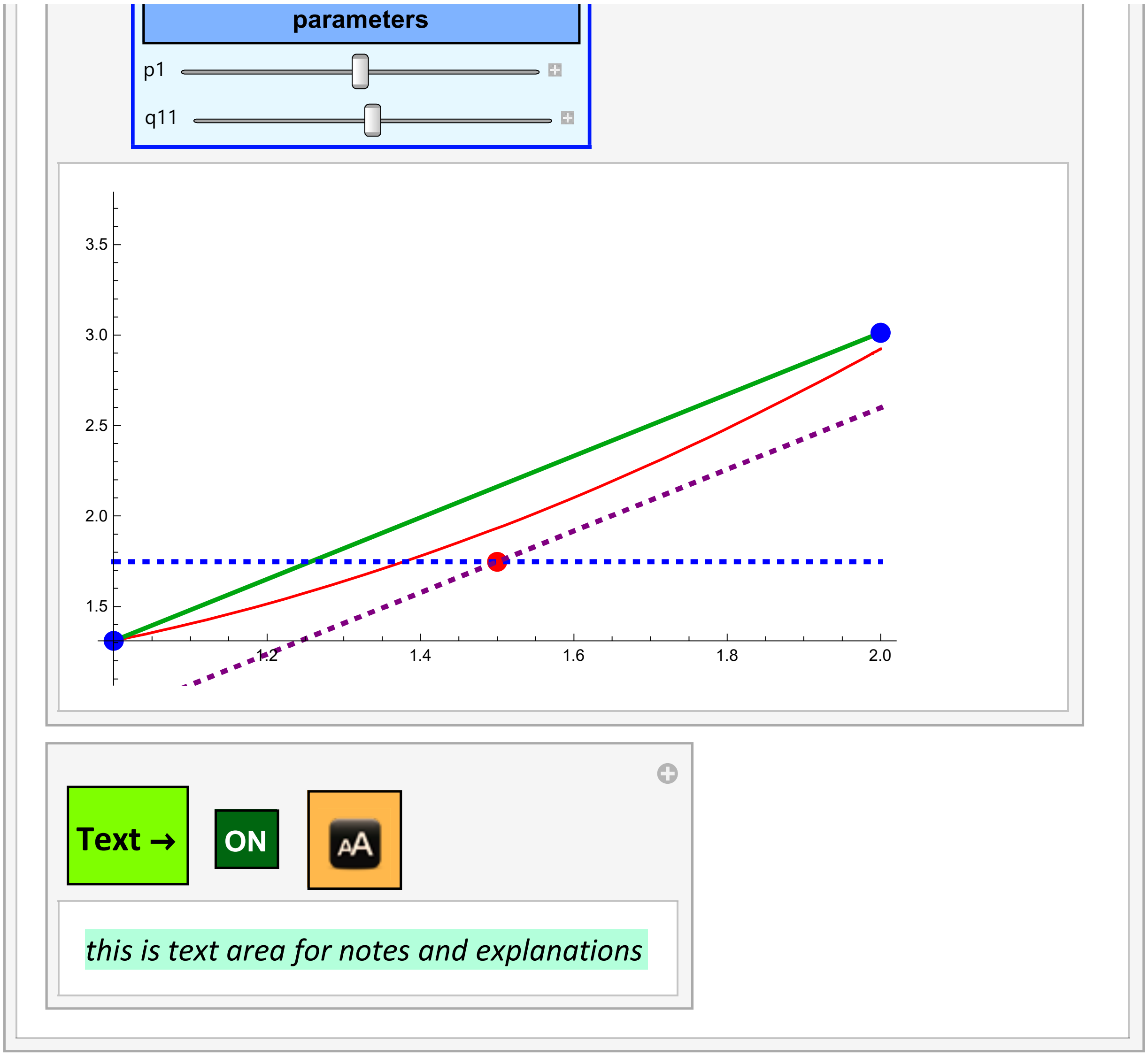
Note that both Heun and mid-point methods indeed achieve error scaling of the second order $O[h^2]$.

4. Runge-Kutta method concept (summary only)

4.1 Basic idea

Use the APP for the discussion.





this is text area for notes and explanations

Theoretically, if we use Taylor’s series theorem, we can, in principle keep on including higher and higher order derivatives in the estimate such that

$$y_{i+1} - \hat{y}_{i+1} \rightarrow O[h^n]$$

as high n as desired.

This means to estimate 1st order derivative, use it to estimate the second, and so forth, which is a recursive process. The process can be parametrized (as illustrated in the APP for second order) with the constraint and criterion such that the residual error is minimum.

This is a complex optimization problem but feasible, and the essence is only to achieve a “recipe” with optimal parameters. The implementation is then simple in concept: Every order of derivative is a function of the previous, which can be traced all the ways back to $f(x_i + p\,h,\ f[x_i + p\,h,\ Q[y_i]])$, which represents an estimate at point $x_i + p\,h$ where p is a parameter, and $Q[y_i]$ represents a corresponding value.

Thus, the general expression is: $y_{i+1} = y_i + \phi(x_i,\ y_i,\ h)\,h$

where $\phi(x_i,\ y_i,$ h) is a function whose form is to be determined for minimal error.

- For Heun's method, note that:

$$\phi(x_i,\, y_i,\, h) = \frac{f(x_i,y_i) + f\left(x_{i+1},y_{i+1}{}^{(0)}\right)}{2} = \frac{f(x_i,y_i) + f[\ x_{i+1},\, y_i + h\, f(x_i,y_i)\ \ ]}{2}$$

- For mid-point method:

$$\phi(x_i,\, y_i,\, h) = f(x_{i+1/2},\, y_{i+1/2}) \ = f(x_i \ + \ h\,/\,2\,,\, y_i \ + \ (h\,/\,2)\, f(x_i,\, y_i))$$

Hence, Runge-Kutta’s method generalizes the concept to create an algorithm (recipe) to obtain $\phi(x_i,\, y_i,$ h) for decreasing error scaling $y_{i+1} - \hat{y}_{i+1} \to O[h^n]$, which is to increase the accuracy as needed. The algorithm is recursive in a manner suitable for computer implementation.

4.2 Example: R-K for second order

4.2.1 Mid-point method

Here is an implementation of R-K method to the second order:
$y_{i+1} = y_i + \phi(x_i,\, y_i,$ h) h;
where
$\phi(x_i,\, y_i,$ h) $= a_1\, k_1 \ + a_2\, k_2$; where
$k_1 = f(x_i,\, y_i)$; and
$k_2 = f(x_i \ + \ p_1\, h,\, y_{\text{intermediate}}) = f(x_i \ + \ p_1\, h,\, y_i \ + \ q_{11}\, f(x_i,\, y_i)\, h)$;
But what are the values a_1 , a_2, p_1 , q_{11}?
Suppose we are doing mid-point method, what are those?
$y_{i+1} \ = \ \text{y}(x_i) + \text{h}\ f(x_i \ + (1\,/\,2)\, h,\, y_{\text{intermediate}})$.
$y_{\text{intermediate}} \ = y_i + (\text{h}/2)\ f(x_i,\, y_i)$.
We notice that:
a_1=0 ;
$a_2 = 1$;
$k_2 = f(x_i \ + \ p_1\, h,\, y_{\text{intermediate}}) \ = f(x_i \ + (1\,/\,2)\, h,\, y_i \ + \ (h\,/\,2)\, f(x_i,\, y_i)\,)$
so:
$p_1 = 1/2$; $q_{11} = 1/2$.
Thus, we say that mid-point method is a special case of 2nd order R-K with:
a_1=0 ;
$a_2 = 1$;
$p_1 = 1/2$; and $q_{11} = 1/2$.

4.2.2 Heun’s method

Are the above parameters the only choices?

No, in general, they are chosen to minimize the residual errors: this is the fundamental concept in the R-K's method.
For example:
$a_1 = a_2 = 1/2$; then: $p_1 = 1$ and q_{11} =1:
Here:
$y_{i+1} = y_i + (a_1\ f(x_i,\, y_i) \ \ + a_2\ \ f(x_i \ + \ p_1\, h,\, y_i \ + \ q_{11}\, f(x_i,\, y_i)\, h) \ \ )$ h;

Let's compare this expression with Heun's method:
$y_{i+1} \ = \ \text{y}(x_i) + \text{h}\frac{f(x_i,y_i) + f\left(x_{i+1},y_{i+1}{}^{(0)}\right)}{2}$

$$= y_i + \mathrm{h}\frac{f(x_i,y_i) + f(x_i + h\,,\; y_i + h\,f(x_i,y_i)\,)}{2}$$

we see that it is exactly the same. Heu’s method is one choice for 2nd order R-K.

4.2.3 Another example: Ralston's method

If we choose $a_2 = 2/3$, then we have so-called Ralston's method.
Here $a_1 = 1/3$, $p_1 = 3/4$ and $q_{11} = 3/4$.

5. Recipes for R-K for higher order

5.1 Third order

A third order R-K expression is:

$$y_{i+1} = y_i + \frac{1}{6}\,h\,(k_1 + 4\,k_2 + k_3);$$

where:

$$k_1 = f(x_i,\, y_i)\;;$$

$$k_2 = f\left(x_i + \frac{1}{2}\,h,\, y_i + \frac{1}{2}\,k_1\,h\right);$$

$$k_3 = f(x_i + h,\, y_i + (2\,k_2 - k_1)\,h\;)\;;$$

Or: $y_{i+1} = y_i + \frac{h}{6}\begin{pmatrix} 1 & 4 & 1 \end{pmatrix}.\begin{pmatrix} k_1 \\ k_2 \\ k_3 \end{pmatrix}$

5.2 Fourth order

A fourth order expression:

$$y_{i+1} = y_i + \frac{1}{6}\,h\,(k_1 + 2\,k_2 + 2\,k_3 + k_4);$$

where:

$$k_1 = f(x_i,\, y_i)\;;$$

$$k_2 = f\left(x_i + \frac{1}{2}\,h,\, y_i + \frac{1}{2}\,k_1\,h\right);$$

$$k_3 = f\left(x_i + \frac{1}{2}\,h,\, y_i + \frac{1}{2}\,k_2\,h\;\right);$$

$$k_4 = f(x_i + h,\, y_i + k_3\,h\;)\;;$$

Or:

$$y_{i+1} = y_i + \frac{h}{6}\begin{pmatrix} 1 & 2 & 2 & 1 \end{pmatrix}.\begin{pmatrix} k_1 \\ k_2 \\ k_3 \\ k_4 \end{pmatrix}$$

5.3 Fifth order R-K (or Butcher's method)

$$y_{i+1} = y_i + \frac{h}{90}\begin{pmatrix} 7 & 32 & 12 & 32 & 7 \end{pmatrix}.\begin{pmatrix} k_1 \\ k_3 \\ k_4 \\ k_5 \\ k_6 \end{pmatrix}$$

(note that k_2 is not included as it is only an intermediate value for recursion)
where:

$k_1 = f(x_i, y_i)$;

$k_2 = f\left(x_i + \frac{1}{4} h, y_i + \frac{1}{4} k_1 h\right)$;

$k_3 = f\left(x_i + \frac{1}{4} h, y_i + \frac{1}{8} k_1 h + \frac{1}{8} k_2 h\right)$;

$k_4 = f\left(x_i + \frac{1}{2} h, y_i - \frac{1}{2} k_2 h + k_3 h\right)$;

$k_5 = f\left(x_i + \frac{3}{4} h, y_i + \frac{3}{16} k_1 h + \frac{9}{16} k_4 h\right)$;

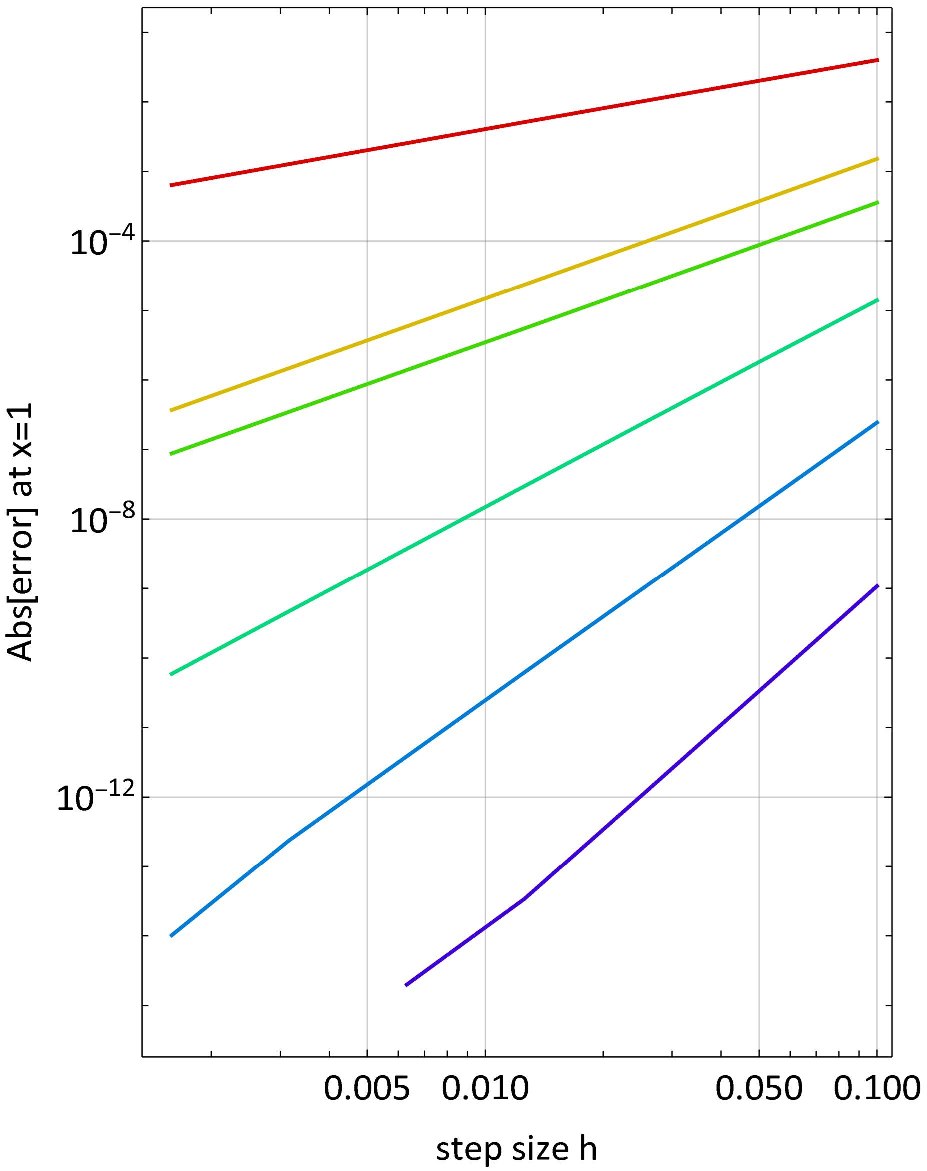
$k_6 = f\left(x_i + h, y_i - \frac{3}{7} k_1 h + \frac{2}{7} k_2 h + \frac{12}{7} k_3 h - \frac{12}{7} k_4 h + \frac{8}{7} k_5 h\right)$;

5.4 Additional note:

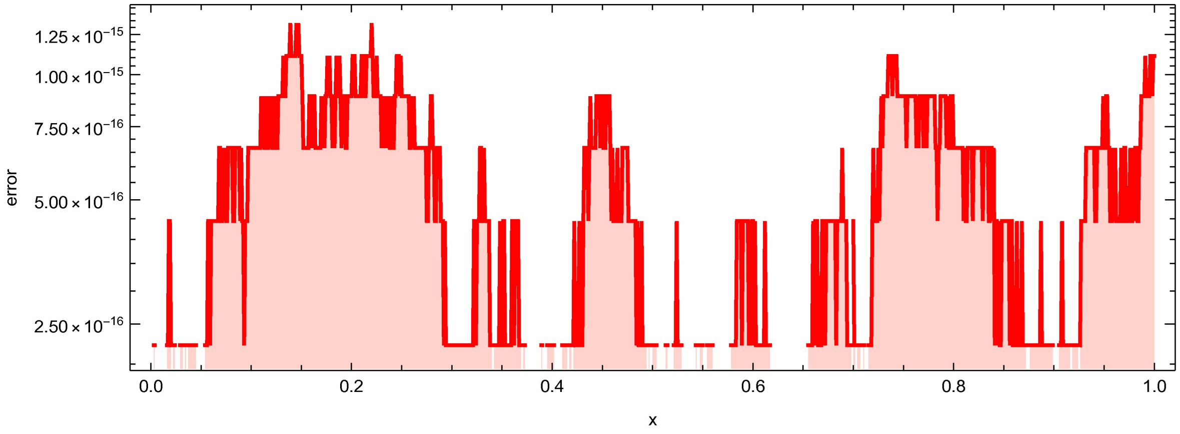
Runge-Kutta methods are useful for numerically solving certain types of ordinary differential equations. Deriving high-order Runge-Kutta methods is no easy task, however. There are several reasons for this. The first difficulty is in finding the so-called order conditions. These are nonlinear equations in the coefficients for the method that must be satisfied to make the error in the method of order $O(h^n)$ for some integer n where h is the step size. The second difficulty is in solving these equations. Besides being nonlinear, there is generally no unique solution, and many heuristics and simplifying assumptions are usually made. Finally, there is the problem of combinatorial explosion. For a twelfth-order method there are 7813 order conditions!
For *Mathematica*, specify “method” option in function NDSolve allows one to choose specific algorithm, and newer version of *Mathematica* also does automatic selection of R-K order.

6. Error comparison

Use the APP to study error. Verify that the scaling is correct for 3rd, 4th, and 5th order R-K.



Note about error near machine epsilon. *Mathematica* gives option for much higher precision if needed.


1.25×10^{-15}
1.00×10^{-15}
7.50×10^{-16}
5.00×10^{-16}
2.50×10^{-16}
error
x