Solving ordinary differential equations (ODEs)

$$F\left[\frac{d^n y}{d x^n}, \frac{d^{n-1} y}{d x^{n-1}}, \ldots, \frac{d y}{d x}, y, x\right] = 0$$

Find y(x). To have a unique solution, need some additional conditions, usually as many as the order of the equation *n*.

Two major practically important types:

1. Initial conditions: all n conditions are specified at the same $x = x_0$:

$$y(x_0) = y_0;\ \left.\frac{dy}{dx}\right|_{x=x_0} = y_1; \ldots; \left.\frac{d^{n-1} y}{dx^{n-1}}\right|_{x=x_0} = y_{n-1}$$

We are interested in the solution for $x>x_0$. Usually this arises in problems where the time evolution of the system is of interest and the initial state is given. So the independent variable x is the time.

2. Boundary conditions: some conditions are specified at $x=x_0$ and some at $x=x_1$

$$y'' + f_1(x) y' + f_2(x) y = g(x);\ y(x_0) = y_0;\ y(x_1) = y_1$$

Initial conditions (initial value problem)

$$F\left[\frac{d^n y}{d t^n}, \frac{d^{n-1} y}{d t^{n-1}}, \ldots, \frac{d y}{d t}, y, t\right]=0$$

$$y(x_0)=y_0;\ \left.\frac{dy}{dt}\right|_{t=t_0}=y_1;\ldots;\left.\frac{d^{n-1} y}{dt^{n-1}}\right|_{t=t_0}=y_{n-1}$$

Equations of order *n* can be reduced to a set of *n* first-order equations. Define *n*-1 new variables:

$$v_1=\frac{dy}{dt};\ v_2=\frac{d^2 y}{dt^2};\ldots;v_{n-1}=\frac{d^{n-1} y}{dt^{n-1}}$$

$$\frac{dy}{dt}=v_1;\ \frac{dv_1}{dt}=v_2;\ldots;\frac{dv_{n-2}}{dt}=v_{n-1};\ F\left[\frac{dv_{n-1}}{dt}, v_{n-1}, \ldots, v_{1,} y, t\right]=0$$

Vector notation: $\frac{d\vec{y}}{dt}=\vec{f}(\vec{y}, t)$ Of course, such sets of ODEs can also arise on their own. Generally, methods for a single 1st order ODE are easily generalizable to sets of 1st order ODEs.

There are some methods specifically for 2nd order ODEs.

3 $\frac{dy(t)}{dt} = f(y(t), t)$ or for brevity, y'(t)=f(y,t). Initial condition $y(t_0)=y_0$.

To solve numerically, advance in discrete steps. Start with $y(t_0)$, define the step h, so the next time is $t_1=t_0+h$. Obtain $y(t_1)$ from $y(t_0)$. Then $t_2=t_1+h$, and $y(t_2)$ is obtained from $y(t_1)$ and, in general, $y(t_0)$ ...

The simplest case is single step methods, where $y(t_{n+1})$ only depends on $y(t_n)$

Taylor expansion:

$$y(t_{n+1}) = y(t_n) + y'(t_n)h + y''(t_n)h^2/2 + \ldots = y(t_n) + f(y(t_n), t_n)h + y''(t_n)h^2/2 + \ldots$$

We don't know the 2nd derivative. Euler method: only keep the term linear in *h*.

$$y(t_{n+1}) = y(t_n) + f(y(t_n), t_n)h$$

The error is $O(h^2)$ and is proportional to the 2nd derivative of y.

Euler method

$$y'(t) = f(y(t), t) \qquad y(t_{n+1}) = y(t_n) + f(y(t_n), t_n)\,h$$

For brevity, $y_n \equiv y(t_n)$. $\qquad y_{n+1} = y_n + f(y_n, t_n)\,h$

The error is $O(h^2)$ and is proportional to y''. But this is the local error, after 1 step. What we really want to know is what the error is at a particular fixed time t.

In the worst case, errors from all steps add up. After N steps, the error is $\sim Nh^2$. $N = (t-t_0)/h \sim 1/h$. So the error at time *t* (the global error) is $O(h^1)$.

Euler method is a first-order method. In general, a numerical method for solving ODEs is of order m, if its global error is $O(h^m)$.

Euler method

$$y'(t)=f(y(t),t) \qquad y_{n+1}=y_n+f(y_n,t_n)h$$

Suppose we have an equation whose solution is oscillatory. So the 2nd derivative changes sign. Will the error still accumulate?

$y'=iy$ The exact solution is $y(t)=y_0e^{it}$. $|y(t)|=|y_0|=\text{const}$

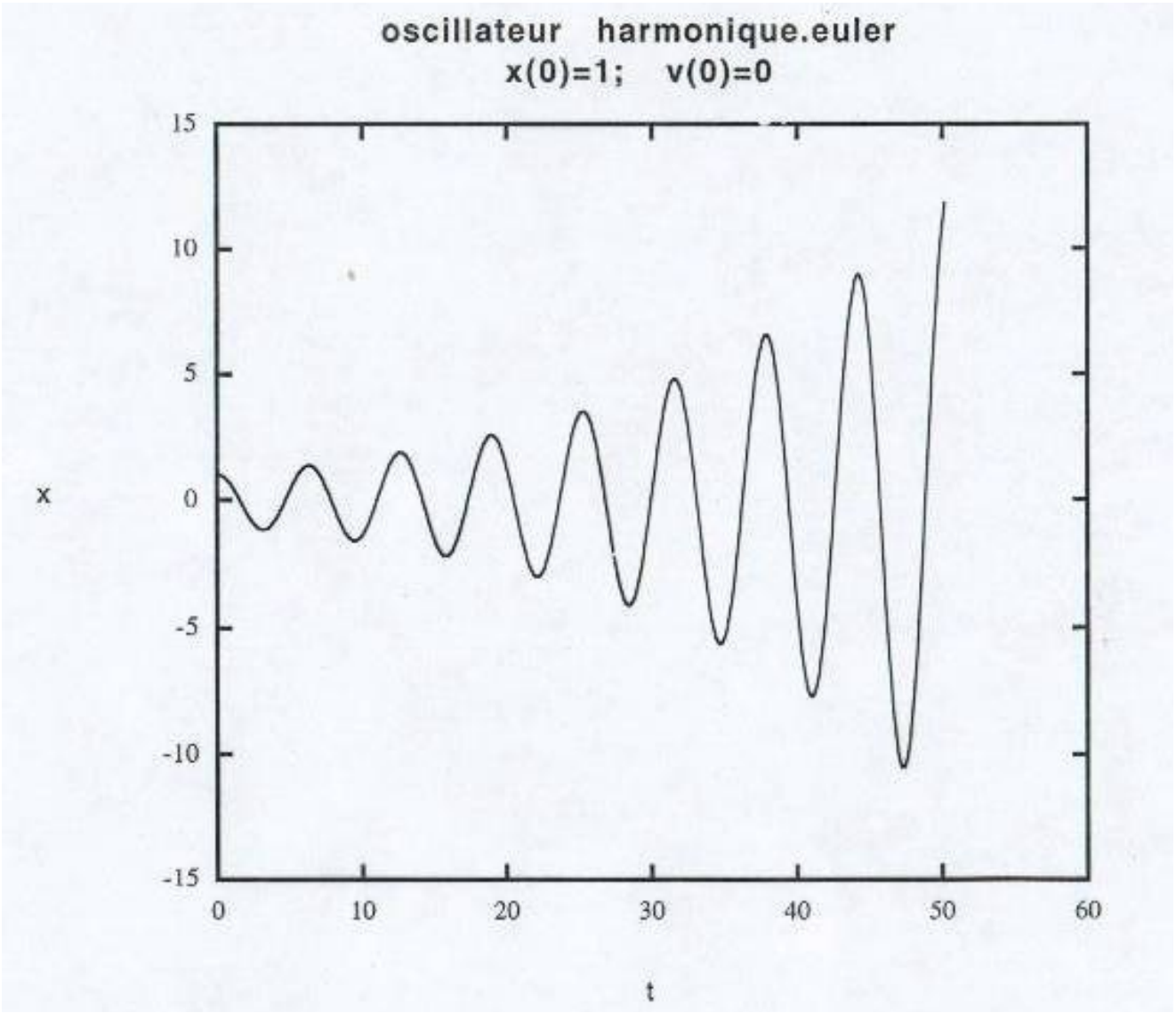
In this case f(y,t)=iy. The Euler method gives

$$y_{n+1}=y_n+ihy_n=(1+ih)y_n \qquad |y_{n+1}|=|1+ih||y_n|=(1+h^2)|y_n|$$

Every step, the absolute value increases by a factor of $(1+h^2)$. Not only does the error accumulate, but it does so in a nasty way – the absolute value grows indefinitely. For this problem, the method is unstable, no matter how small *h* is.

From Prof. L'Heureux notes:

$$\ddot{x} = -x \text{ or } \dot{x} = v,\ \dot{v} = -x$$



$$y'(t) = f(y(t), t) \qquad y_{n+1} = y_n + f(y_n, t_n)\, h$$

More generally, consider $y' = \lambda y$, $\lambda = \text{const}$

|y| decreases with time for Re λ < 0.

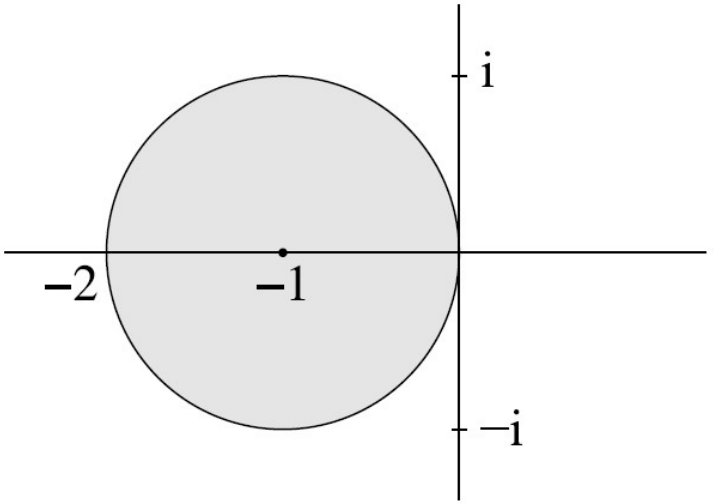
Euler: $y_{n+1} = y_n + h\lambda y_n = (1 + h\lambda) y_n$ y_n decreases with n, when |1+hλ| < 1.

$$|1 + h\lambda| = \sqrt{(1 + \text{Re}\,(h\lambda))^2 + (\text{Im}\,(h\lambda))^2} < 1$$

A circle in the (Re hλ,Im hλ) plane.

The imaginary axis is completely outside the circle.

Once there is any amount of damping, for small enough h the solution will be damped.



Gustafsson, Fundamentals of Scientific Computing

But for very large damping (large negative Re λ), becomes unstable again!

$$y'(t) = f(y(t), t) \qquad y_{n+1} = y_n + f(y_n, t_n)h$$

$$\frac{y_{n+1} - y_n}{h} = f(y_n, t_n)$$

$\frac{y_{n+1} - y_n}{h}$ is a forward difference approximation for $y'(t_n)$

Can view the Euler method as an approximation of the ODE where the derivative is replaced with its forward difference approximation

$\frac{y_{n+1} - y_n}{h}$ is also a backward difference approximation for $y'(t_{n+1})$

$$\frac{y_{n+1} - y_n}{h} = f(y_{n+1}, t_{n+1}) \qquad y_{n+1} = y_n + f(y_{n+1}, t_{n+1})h$$

Not as easy to use: rather than having an explicit expression for y_{n+1}, need to solve a (generally nonlinear) equation.

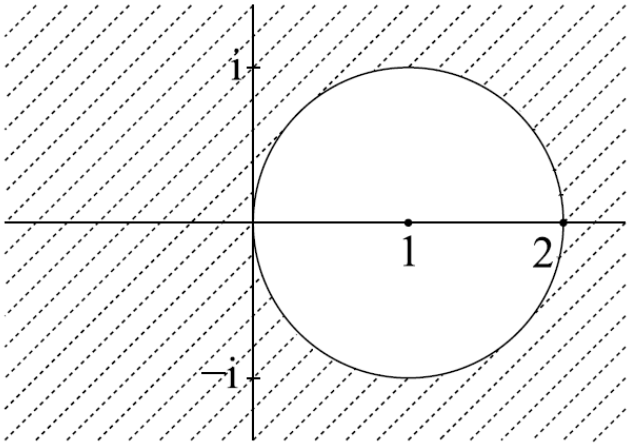
Such methods are called implicit. Despite being harder to use, they are sometimes useful. This particular method is called backward Euler or implicit Euler.

$$y_{n+1} = y_n + f(y_{n+1}, t_{n+1})h$$

Consider again y'= λy. $y_{n+1} = y_n + h\lambda y_{n+1}$ Can solve the equation:

$$y_{n+1} = \frac{y_n}{1 - h\lambda}$$ Stable, if |1-hλ| > 1.

$$|1 - h\lambda| = \sqrt{(1 - \mathrm{Re}\,(h\lambda))^2 + (\mathrm{Im}\,(h\lambda))^2} > 1$$



Gustafsson, Fundamentals of Scientific Computing

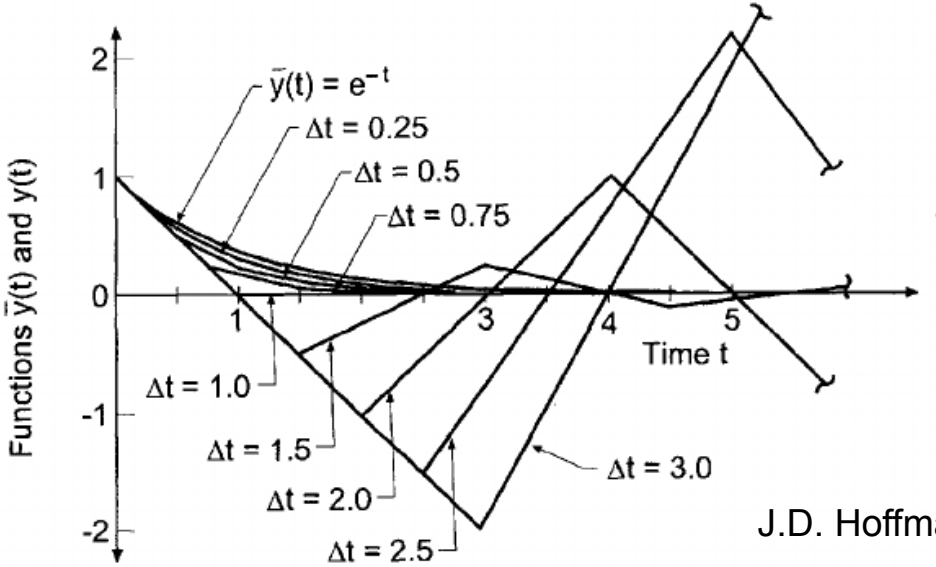
Again, a circle, but now the outside of the circle is stable.

For purely imaginary λ, oscillations always decay. For real negative λ, always stable, no matter how large the step *h* is.

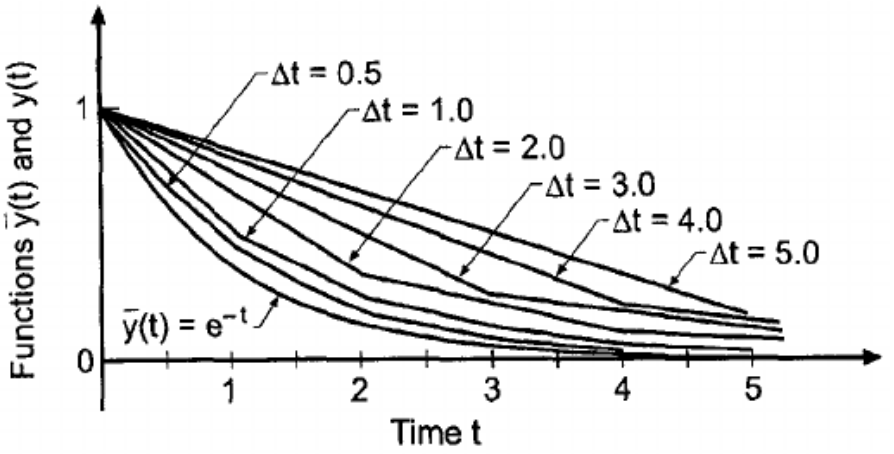
$$y(t_n) = y(t_{n+1}) - hy'(t_{n+1}) + O(h^2) = y(t_{n+1}) - hf(y_{n+1}, t_{n+1}) + O(h^2)$$

Just like for explicit Euler, the error is $O(h^2)$ locally and O(h) globally.

Explicit Euler



Implicit Euler



J.D. Hoffman, Numerical Methods for Engineers and Scientists

In this case the stability feature does not matter much, because for such large steps the error is large anyway. But there are cases when this is more important.

$$y'(t) = f(y(t), t)$$

$$y_{n+1} = y_n + f(y_n, t_n)h \quad \text{– Euler}$$

$$y_{n+1} = y_n + f(y_{n+1}, t_{n+1})h \quad \text{– backward Euler}$$

Consider the “average” between these two methods.

$$y_{n+1} = y_n + \frac{1}{2}[f(y_n, t_n) + f(y_{n+1}, t_{n+1})]h$$

Note that we can integrate both sides of the ODE from t_n to t_{n+1} and get

$$y_{n+1} - y_n = \int_{t_n}^{t_{n+1}} f(y(t), t)\,dt$$

The new method is just the trapezoidal rule for the integral, so it is known as the trapezoidal method. Since the local error of the trapezoidal rule is $O(h^3)$, the local error of the trapezoidal method for ODEs is also $O(h^3)$ and the global error is $O(h^2)$. So we now have a second-order method.

$$y_{n+1} = y_n + \frac{1}{2}[f(y_n, t_n) + f(y_{n+1}, t_{n+1})]\,h$$

Consider again y'= λy. $\quad y_{n+1} = y_n + h\lambda[y_n + y_{n+1}]/2$

$$y_{n+1}(1 - h\lambda/2) = y_n(1 + h\lambda/2) \Rightarrow y_{n+1} = \frac{1 + h\lambda/2}{1 - h\lambda/2}\, y_n$$

The solution decays for $\left|\frac{1 + h\lambda/2}{1 - h\lambda/2}\right| < 1$

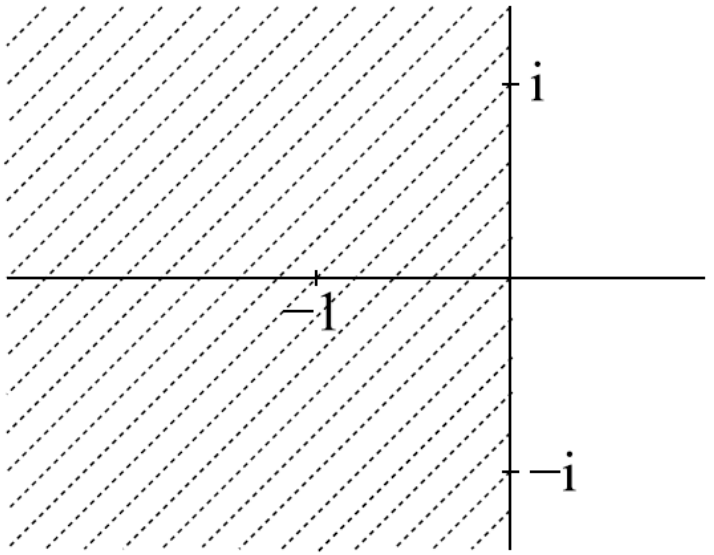
$$(1 + h\,\mathrm{Re}(\lambda)/2)^2 + (h\,\mathrm{Im}(\lambda)/2)^2 < (1 - h\,\mathrm{Re}(\lambda)/2)^2 + (h\,\mathrm{Im}(\lambda)/2)^2 \Rightarrow \mathrm{Re}(\lambda) < 0$$

Same condition as for the original ODE.

Does not mean that the decay rate is the same: note that in the limit $\lambda \to -\infty$ $y_{n+1} = -y_n$

Implicit Euler is actually better in this respect.

L-stability vs. A-stability



This consideration is actually relevant for the analysis of stability of solutions for an arbitrary f(y,t).

$$y' = f(y,t) \qquad y_s(t) + \delta y(t) \qquad y_s' + \delta y' = f(y_s + \delta y, t) \Rightarrow \delta y' \approx \left.\frac{\partial f}{\partial y}\right|_{y=y_s} \delta y$$

For a set of equations,

$\vec{y}' = A\vec{y}$ Matrix A has eigenvalues λ_i Solution decays if all Re λ_i<0

$\vec{y}_{n+1} = (I + hA)\vec{y}_n$ Eigenvalues of (I+hA) are $1+h\lambda_i$. Can analyze one by one.

This is why the consideration for y'=iy was also valid for the harmonic oscillator

14 Even when stability is not an issue, the low accuracy of the (explicit) Euler method is a problem (it's only 1st order). Are there higher-order explicit methods?

Trapezoidal method $$y_{n+1} = y_n + \frac{1}{2}[f(y_n, t_n) + f(y_{n+1}, t_{n+1})]h$$

To make it explicit, estimate y_{n+1} using Euler method and use that estimate in the trapezoidal expression:

$$\tilde{y}_{n+1} = y_n + f(y_n, t_n)h$$

$$y_{n+1} = y_n + \frac{1}{2}[f(y_n, t_n) + f(\tilde{y}_{n+1}, t_{n+1})]h$$

Improved Euler or Heun method

Since the error of $\tilde{y}_{n+1}$ is locally 2nd order, this adds an additional contribution to the error of y_{n+1} of order h^3, which does not change the order of the method compared to the trapezoidal method.

This is the simplest example of a predictor-corrector method: the value of y_{n+1} is predicted and then corrected.

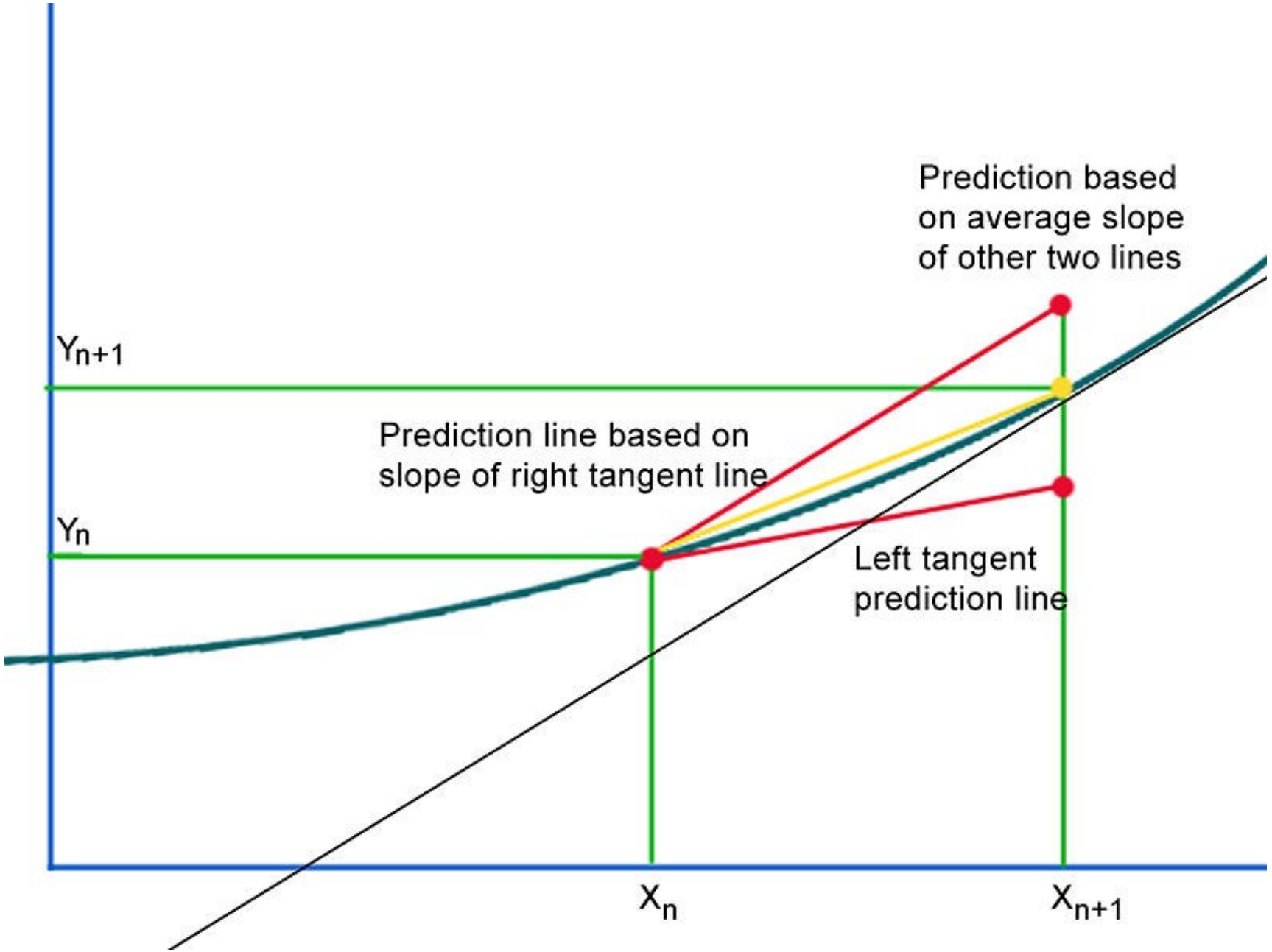


Illustration of Heun's method from Wikipedia.

$$y_{n+1} - y_n = \int_{t_n}^{t_{n+1}} f\left(y(t), t\right) dt$$

Besides trapezoid, another possible approximation of the integral of the same order is the midpoint rule:

$$\int_{t_n}^{t_{n+1}} f\left(y(t), t\right) dt \approx hf\left(y_{n+1/2}, t_{n+1/2}\right)$$

As for Heun, estimate $y_{n+1/2}$ using Euler method.

$$\tilde{y}_{n+1/2} = y_n + f\left(y_n, t_n\right) h/2$$

$$y_{n+1} = y_n + f\left(\tilde{y}_{n+1/2}, t_{n+1/2}\right) h$$

Heun and midpoint methods are examples of a class of methods called Runge-Kutta methods.

$$y_{n+1} = y_n + h\sum_{i=1}^{s} b_i k_i \qquad \sum_{i=1}^{s} b_i = 1$$

$$k_1 = f(y_n, t_n)$$

$$k_2 = f(y_n + a_{21} h k_1, t_{n+c_2}) \qquad a_{21} = c_2$$

$$k_3 = f(y_n + a_{31} h k_1 + a_{32} h k_2, t_{n+c_3}) \qquad a_{31} + a_{32} = c_3$$

.....

$$k_s = f(y_n + a_{s1} h k_1 + \ldots + a_{s,s-1} h k_{s-1}, t_{n+c_s})$$

Here k_i are some estimates of the derivative at various points c_i, and the increment of y is proportional to some weighted average of these derivatives.

Heun: s = 2, $c_2 = a_{21} = 1$; $b_1 = b_2 = 1/2$.

Midpoint: s = 2, $c_2 = a_{21} = 1/2$; $b_1 = 0$; $b_2 = 1$

Deriving Runge-Kutta methods is a bit complicated already for 2nd order

$$y_{n+1}-y_n=h\left[(1-b_2)f\left(y_n,t_n\right)+b_2 f\left(y_n+c_1 hf\left(y_n,t_n\right),t_{n+c_1}\right)\right]$$

$$hf\left(y_n,t_n\right)+\frac{h^2}{2}\frac{\partial f}{\partial y}f+\frac{h^2}{2}\frac{\partial f}{\partial t}=h\left[(1-b_2)f\left(y_n,t_n\right)+b_2\left[f\left(y_n,t_n\right)+\frac{\partial f}{\partial y}c_1 h f+\frac{\partial f}{\partial t}c_1 h\right]\right]$$

Equate the coefficients of $f,\ \frac{\partial f}{\partial y}f,\ \frac{\partial f}{\partial t}$

$$b_2=\frac{1}{2c_1}$$

So there is a whole series of 2nd order Runge-Kutta methods, none of which are 3rd order.

Heun and midpoint are two particular cases.

The most famous 4th order Runge-Kutta method:

$$y_{n+1} = y_n + \frac{1}{6} h \left(k_1 + 2k_2 + 2k_3 + k_4 \right)$$

$$k_1 = f\left(y_n, t_n\right)$$

$$k_2 = f\left(y_n + hk_1/2, t_{n+1/2}\right)$$

$$k_3 = f\left(y_n + hk_2/2, t_{n+1/2}\right)$$

$$k_4 = f\left(y_n + hk_3, t_{n+1}\right)$$

0				
1/2	1/2			
1/2	0	1/2		
1	0	0	1	
	1/6	1/3	1/3	1/6

RK4

Compact representation of Runge-Kutta schemes – Butcher tableau

c_1	a_{11}	a_{12}	$\dots$	a_{1s}
c_2	a_{21}	a_{22}	$\dots$	a_{2s}
$\vdots$	$\vdots$	$\vdots$	$\ddots$	$\vdots$
c_s	a_{s1}	a_{s2}	$\dots$	a_{ss}
	b_1	b_2	$\dots$	b_s

General scheme including implicit

0		
α	α	
	$(1 - \frac{1}{2\alpha})$	$\frac{1}{2\alpha}$

RK2

0	0	0
1	$\frac{1}{2}$	$\frac{1}{2}$
	$\frac{1}{2}$	$\frac{1}{2}$

Trapezoidal

0	
1	

Euler

1	1
	1

Backward Euler

For orders higher than 4 more stages are needed than the order.

Table 2. Number of stages to achieve a specified order

order p	1	2	3	4		5	6		7		8
Conditions	1	2	4	8		17	37		85		200
Stages	1	2	3	4	5	6	7	8	9	10	11
Parameters	1	3	6	10	15	21	28	36	45	55	66

http://www.scholarpedia.org/article/Runge-Kutta_methods

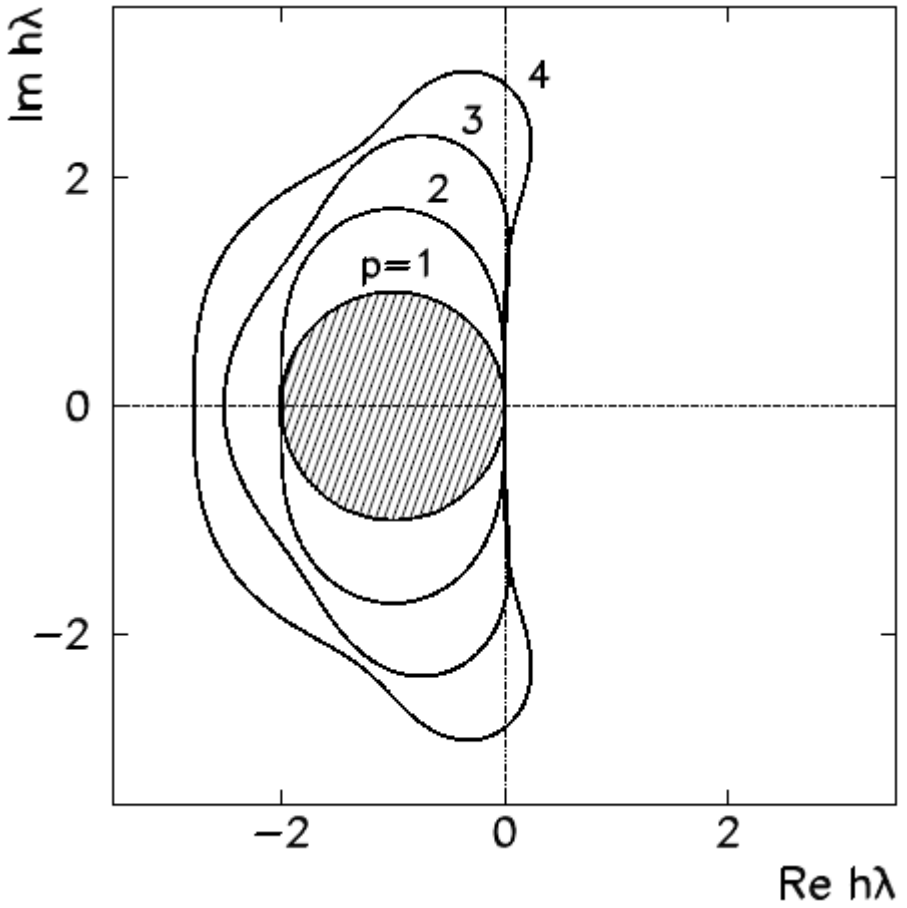
Local error estimate

Do a step twice: once as a single step and once as two half-steps. Find the difference to estimate the local error. Can do an adaptive step: if the error is too big, reduce the step; if it is too small, increase.

Another possibility: embedded schemes. Two schemes of different orders sharing the same k_i, but different b_i. Cash-Karp (5^{th}-4^{th} order):

0						
1/5	1/5					
3/10	3/40	9/40				
3/5	3/10	−9/10	6/5			
1	−11/54	5/2	−70/27	35/27		
7/8	1631/55296	175/512	575/13824	44275/110592	253/4096	
	37/378	0	250/621	125/594	0	512/1771
	2825/27648	0	18575/48384	13525/55296	277/14336	1/4

Stability diagram



Size of the stability region increases with the order.

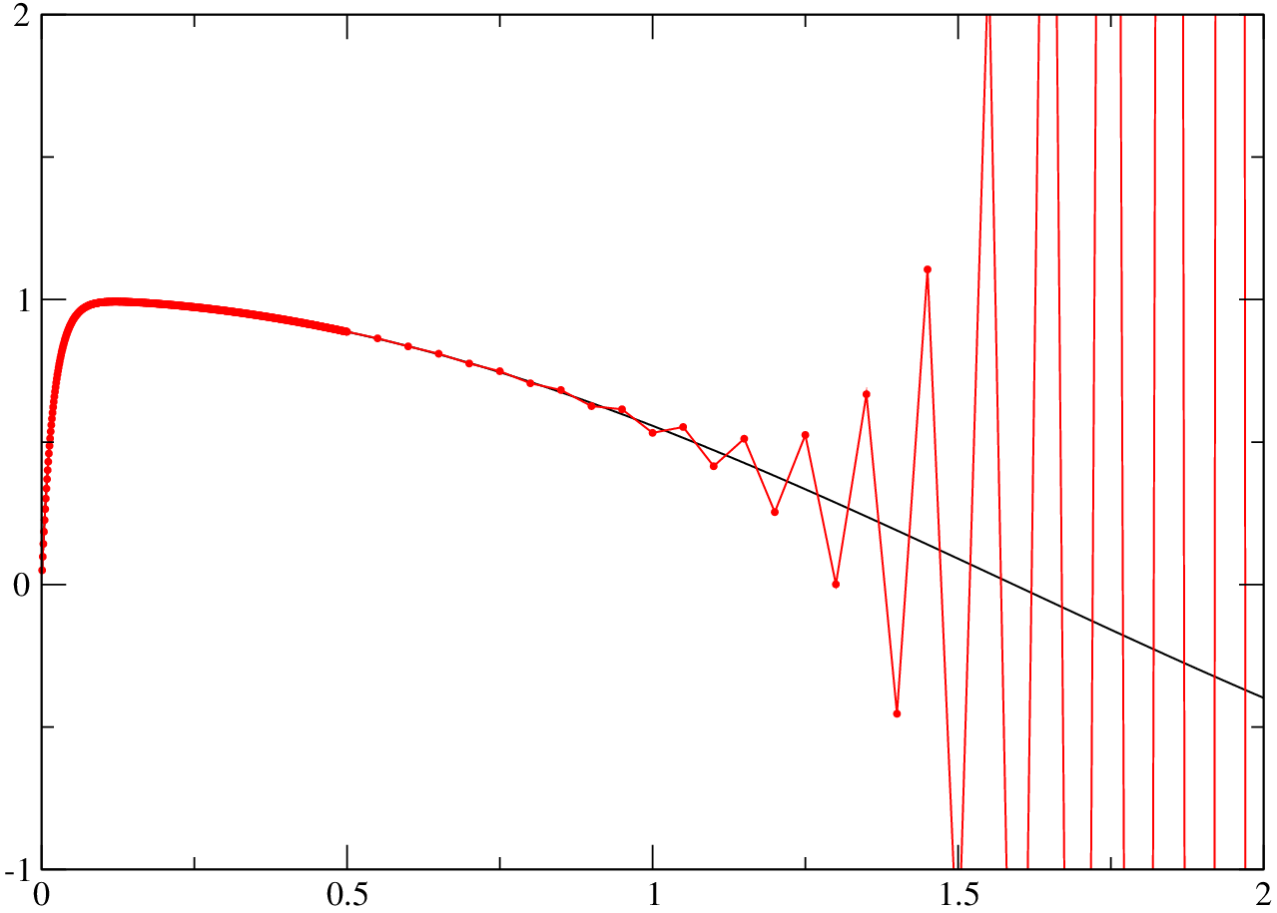
For RK4, points on the imaginary axis below ~3 are slightly inside – oscillations will decay, but very slowly.

Potentially, same problem with large negative Re λ.

Stiff equations

An ODE is stiff if the step size required for stability is much smaller than the smoothness of the exact solution would suggest.

$$y' = -50(y - \cos t),\ y(0) = 0$$



After a short transient, follows y=cos t. There are no explicit methods stable for any step size. Implicit methods can actually be useful.

Multistep methods

Runge-Kutta methods take some intermediate steps evaluating f(y,t) in between steps, but then the information is discarded before the next step is taken. Instead, can use the information from previous steps to increase the order of accuracy of the method. General form for linear multistep methods:

$$y_{n+s} + a_{s-1} y_{n+s-1} + \ldots + a_0 y_n =$$

$$h\left(b_s f\left(y_{n+s}, t_{n+s}\right) + b_{s-1} f\left(y_{n+s-1}, t_{n+s-1}\right) + \ldots + b_0 f\left(y_n, t_n\right)\right)$$

Implicit, if $b_s \neq 0$. Some of these are predictor-corrector methods.

http://en.wikipedia.org/wiki/Linear_multistep_method

Adams-Bashforth-Moulton. Adams-Bashforth as predictor, Adams-Moulton as corrector.

Multivalue methods are equivalent, but easier to do adaptive step size.

May be worth considering if function calculation is costly.

Implicit Runge-Kutta methods based on Gaussian integration.

Bulirsch-Stoer method

Uses several different step sizes and extrapolates to zero step size, similar to Romberg integration for doing integrals. A special “modified midpoint method” is used, which is a 2^{nd} order method, but with even terms in the series only, just as . Extrapolation can be done using polynomial or rational functions. Can be useful if very high accuracy is required.

As mentioned, one problem with even high-order Runge-Kutta methods is that the amplitude of oscillations changes slowly. Basically, means slow energy drift. This is obviously undesirable. Can we fix this problem? Symplectic algorithms and Verlet algorithm in particular.