ODEs

Introduction to Numerical Analysis

Riccardo Mannella

April 23, 2008

ODE: generality

The problem is the numerical integration of an Ordinary (set of) Differential Equation(s):

$$y' = f(x, y)$$

starting from $y(x_0) = y_0$ and for $x_0 \leq x \leq x_f$.

ODE: generality

The problem is the numerical integration of an Ordinary (set of) Differential Equation(s):

$$y' = f(x, y)$$

starting from $y(x_0) = y_0$ and for $x_0 \leq x \leq x_f$.
A set of equations can normally be cast in the form of a system of first order ODE's: $y'' = f(x, y, y')$ becomes

$$y' = v$$

$$v' = f(x, y, v)$$

ODE: generality

Given something like

$$y'' = f(x, y, y') \qquad x_0 \leq x \leq x_f$$

there are actually two different problems:

- Initial boundary value problems: $y(x_0)$ and $y'(x_0)$ are given, and y and y' for x up to x_f should be found

ODE: generality

Given something like

$$y'' = f(x, y, y') \qquad x_0 \leq x \leq x_f$$

there are actually two different problems:

- Initial boundary value problems: $y(x_0)$ and $y'(x_0)$ are given, and y and y' for x up to x_f should be found
- Two-point boundary value problems: the boundaries are given at more than one x: for instance, we know $y(x_0)$ and $y(x_f)$, and again we want to compute y and y' for all x.

ODE: generality

Typically, the solution is obtained taking a mesh for x, $x_0 < x_1 < ... < x_i < ... < x_f$, and moving y from x_i to x_{i+1}, until the point x_f is reached. Of course, there is a need for an appropriate mesh and/or "integration time step" h.
We need three different pieces of code:

- The code to get $y(x_{i+1})$ given $y(x_i)$, ie the *integration algorithm*

ODE: generality

Typically, the solution is obtained taking a mesh for x, $x_0 < x_1 < ... < x_i < ... < x_f$, and moving y from x_i to x_{i+1}, until the point x_f is reached. Of course, there is a need for an appropriate mesh and/or "integration time step" h.
We need three different pieces of code:

- The code to get $y(x_{i+1})$ given $y(x_i)$, ie the *integration algorithm*
- The code to get the optimal h, ie some *stepper*. Typically, this will check the error associated with the integration algorithm to decide the optimal step

ODE: generality

Typically, the solution is obtained taking a mesh for x, $x_0 < x_1 < ... < x_i < ... < x_f$, and moving y from x_i to x_{i+1}, until the point x_f is reached. Of course, there is a need for an appropriate mesh and/or "integration time step" h.
We need three different pieces of code:

- The code to get $y(x_{i+1})$ given $y(x_i)$, ie the *integration algorithm*
- The code to get the optimal h, ie some *stepper*. Typically, this will check the error associated with the integration algorithm to decide the optimal step
- The code to start and end the integration at x_0, x_f.

Generic multistep

Given the ODE

$$y' = f(y, x)$$

and a time mesh $\{x_0, x_1, ..., x_n\}$, with $x_i = ih$, a generic multistep has the form ($y'_i \equiv f(y_i, x_i)$)

$$y_{n+1} = \sum_{j=0}^{p} a_j y_{n-j} + h \sum_{j=-1}^{p} b_j y'_{n-j}$$

How can we use this? Let us derive a simple second order explicit scheme, of Adams type ($\equiv a_j = 0 \ \ \forall j > 1$)

$$y_{n+1} = a_0 y_n + h \left[b_0 y'_n + b_1 y'_{n-1} \right]$$

Generic multistep

$$y_{n+1} = \sum_{j=0}^{p} a_j y_{n-j} + h \sum_{j=-1}^{p} b_j y'_{n-j}$$

$$y_{n+1} = a_0 y_n + h \left[b_0 y'_n + b_1 y'_{n-1} \right]$$

Take $y_{n+j} = (t + jh)^3$ and substitute,

$$(t+h)^3 = a_0 t^3 + h \left[3b_0 t^2 + 3b_1 (t-h)^2 \right]$$

and expanding and matching the h coefficients

$$1 = a_0 \quad 3t^2 h = 3b_0 t^2 h + 3b_1 t^2 h \quad 3th^2 = -6b_1 th^2$$

Generic multistep

$$y_{n+1} = \sum_{j=0}^{p} a_j y_{n-j} + h \sum_{j=-1}^{p} b_j y'_{n-j}$$

$$y_{n+1} = a_0 y_n + h \left[b_0 y'_n + b_1 y'_{n-1} \right]$$

Take $y_{n+j} = (t + jh)^3$ and substitute,

$$(t+h)^3 = a_0 t^3 + h \left[3b_0 t^2 + 3b_1 (t-h)^2 \right]$$

and expanding and matching the h coefficients

$$1 = a_0 \quad 3t^2 h = 3b_0 t^2 h + 3b_1 t^2 h \quad 3th^2 = -6b_1 th^2$$

$$a_0 = 1 \quad b_1 = -1/2 \quad b_0 = 3/2 \quad \text{Error} \propto 6h^3/2$$

Generic multistep: precision

The first problem is the precision of the integration. It is not a good idea to decrease the integration time step h too much, because the roundoff takes over.

Generic multistep: precision

The first problem is the precision of the integration. It is not a good idea to decrease the integration time step h too much, because the roundoff takes over.

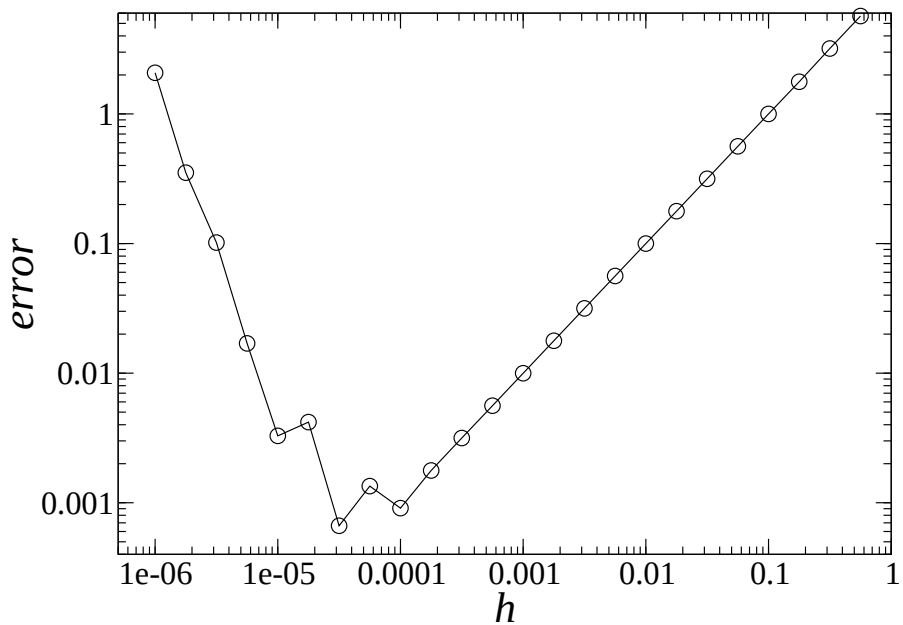
$$y' = 2x$$

$$y(0) = 0$$

$$y_{n+1} = y_n + 2hx_n$$

Exact result:

$$y(x) = x^2$$



ODE: a couple of definitions

We need to define *stability* and *convergence* for a given numerical algorithm. Suppose the mesh is $x_0, x_1, ..., x_p, ..., x_n, x_f$, and we have a solution y_i at each x_i, with h time step and $y' = f(x, y)$.

Stability if changing slightly the initial conditions we have some new z_i such that $|z_i - y_i| < \epsilon$ for $i < p$, then $|z_i - y_i| < c\epsilon$ for $i \geq p$.

Convergence if y_i is such that $\text{Max}_{i<p}|Y_i - y_i| \longrightarrow 0$ if $h \longrightarrow 0$, then for $h \longrightarrow 0$ we have $\text{Max}_{i>p}|Y_i - y_i| \longrightarrow 0$, and y_i is said to converge to Y_i.

ODE: a couple of definitions

We need to define *stability* and *convergence* for a given numerical algorithm. Suppose the mesh is $x_0, x_1, ..., x_p, ..., x_n, x_f$, and we have a solution y_i at each x_i, with h time step and $y' = f(x, y)$.

Stability if changing slightly the initial conditions we have some new z_i such that $|z_i - y_i| < \epsilon$ for $i < p$, then $|z_i - y_i| < c\epsilon$ for $i \geq p$.

Convergence if y_i is such that $\text{Max}_{i<p}|Y_i - y_i| \longrightarrow 0$ if $h \longrightarrow 0$, then for $h \longrightarrow 0$ we have $\text{Max}_{i>p}|Y_i - y_i| \longrightarrow 0$, and y_i is said to converge to Y_i.

A consistent method ($1 = \sum_{j=0}^{p} a_j$) is convergent iff it is stable

Generic multistep: stability/1

The problem of stability is a difficult one.

$$y_{n+1} = \sum_{j=0}^{p} a_j y_{n-j} + h \sum_{j=-1}^{p} b_j y'_{n-j}$$

In general $1 = \sum_{j=0}^{p} a_j$ & $0 = -\sum_{j=0}^{p} j a_j + \sum_{j=-1}^{p} b_j$
Stability and convergence are related to the root of the equation
(in a moment we see where this comes from!)

$$\rho(r) = r^{p+1} - \sum_{j=0}^{p} a_j r^{p-j}$$

Note that $\rho(1) = 0$, ie 1 is always a root.

Generic multistep: stability/2

Stability is assured by the *root condition*: start from

$$\rho(r) = r^{p+1} - \sum_{j=0}^{p} a_j r^{p-j}$$

It has $p+1$ roots. The *root condition* means

- $|r_j| \leq 1 \quad j = 0, 1, ..., p$
- $|r_j| = 1$ if r_j is a simple root

and, if satisfied, then the method is stable.
Recall our multistep, we have $p = 1$ and $a_0 = 1$, and $0 = r^2 - r$, roots are 1 and 0, so the method is stable.

Generic multistep: stability/3

$$y_{n+1} = \sum_{j=0}^{p} a_j y_{n-j} + h \sum_{j=-1}^{p} b_j y'_{n-j}$$

More stringent conditions are obtained considering a simple differential equation:

$$y' = \lambda y(x)$$

which has solution

$$y_n = y(x_n) = y(0) \exp \lambda x_n = y(0) \left[\exp \lambda h\right]^n$$

Assess stability using $y_n = \rho^n$ in the recursive relation

Generic multistep: stability/3

$$y_{n+1} = \sum_{j=0}^{p} a_j y_{n-j} + h \sum_{j=-1}^{p} b_j y'_{n-j}$$

More stringent conditions are obtained considering a simple differential equation:

$$y' = \lambda y(x)$$

Study the root behaviour now

$$\rho^{n+1} = \sum_{j=0}^{p} a_j \rho^{n-j} + \lambda h \sum_{j=-1}^{p} b_j \rho^{n-j}$$

Generic multistep: stability/4

$$\rho^{p+1} = \sum_{j=0}^{p} a_j \rho^{p-j} + \lambda h \sum_{j=-1}^{p} b_j \rho^{p-j}$$

There will be $p+1$ roots $\rho_i(\lambda h)$. If they are such that

$$|\rho_j(\lambda h)| \leq \rho_0(\lambda h) \quad j = 1, ..., p$$

we talk of *relatively stable* method. A method which is stable but not relatively stable is called *weakly stable*.

Generic multistep: stability/4

$$\rho^{p+1} = \sum_{j=0}^{p} a_j \rho^{p-j} + \lambda h \sum_{j=-1}^{p} b_j \rho^{p-j}$$

There will be $p+1$ roots $\rho_i(\lambda h)$. If they are such that

$$|\rho_j(\lambda h)| \leq \rho_0(\lambda h) \quad j = 1, ..., p$$

we talk of *relatively stable* method. A method which is stable but not relatively stable is called *weakly stable*. For our multistep $a_0 = 1, b_0 = 3/2, b_1 = -1/2, p = 1$:

$$\rho^2 - (1 + 3\lambda h/2)\rho + \lambda h/2 = 0$$

Generic multistep: stability/4

$$\rho^{p+1} = \sum_{j=0}^{p} a_j \rho^{p-j} + \lambda h \sum_{j=-1}^{p} b_j \rho^{p-j}$$

There will be $p+1$ roots $\rho_i(\lambda h)$. If they are such that

$$|\rho_j(\lambda h)| \leq \rho_0(\lambda h) \quad j = 1, ..., p$$

we talk of *relatively stable* method. A method which is stable but not relatively stable is called *weakly stable*. For our multistep $a_0 = 1, b_0 = 3/2, b_1 = -1/2, p = 1$:

$$\rho = \frac{1 + 3\lambda h/2 \pm \sqrt{(1 + 3\lambda h/2)^2 - 2\lambda h}}{2}$$

Generic multistep: stability/4

$$\rho^{p+1} = \sum_{j=0}^{p} a_j \rho^{p-j} + \lambda h \sum_{j=-1}^{p} b_j \rho^{p-j}$$

There will be $p+1$ roots $\rho_i(\lambda h)$. If they are such that

$$|\rho_j(\lambda h)| \leq \rho_0(\lambda h) \quad j = 1, ..., p$$

we talk of *relatively stable* method. A method which is stable but not relatively stable is called *weakly stable*. For our multistep $a_0 = 1, b_0 = 3/2, b_1 = -1/2, p = 1$:

$$\rho_{0,1} \approx 1 + \lambda h, \frac{\lambda h}{2} \quad \text{relatively stable}$$

Generic multistep: stability/5

$$\rho^{p+1} = \sum_{j=0}^{p} a_j \rho^{p-j} + \lambda h \sum_{j=-1}^{p} b_j \rho^{p-j}$$

Finally, if the $p+1$ roots $\rho_i(\lambda h)$ are such that

$$|\rho_j(\lambda h)| < 1 \quad j = 0, ..., p$$

we talk of *absolutely stable* method.

Generic multistep: stability/5

$$\rho^{p+1} = \sum_{j=0}^{p} a_j \rho^{p-j} + \lambda h \sum_{j=-1}^{p} b_j \rho^{p-j}$$

Finally, if the $p+1$ roots $\rho_i(\lambda h)$ are such that

$$|\rho_j(\lambda h)| < 1 \quad j = 0, ..., p$$

we talk of *absolutely stable* method. For our multistep $a_0 = 1, b_0 = 3/2, b_1 = -1/2, p = 1$:

$$\rho^2 - (1 + 3\lambda h/2)\rho + \lambda h/2 = 0$$

Generic multistep: stability/5

$$\rho^{p+1} = \sum_{j=0}^{p} a_j \rho^{p-j} + \lambda h \sum_{j=-1}^{p} b_j \rho^{p-j}$$

Finally, if the $p+1$ roots $\rho_i(\lambda h)$ are such that

$$|\rho_j(\lambda h)| < 1 \quad j = 0, ..., p$$

we talk of *absolutely stable* method. For our multistep $a_0 = 1, b_0 = 3/2, b_1 = -1/2, p = 1$:

$$\rho_{0,1} = \frac{1 + 3\lambda h/2 \pm \sqrt{(1 + 3\lambda h/2)^2 - 2\lambda h}}{2}$$

Generic multistep: stability/5

$$\rho^{p+1} = \sum_{j=0}^{p} a_j \rho^{p-j} + \lambda h \sum_{j=-1}^{p} b_j \rho^{p-j}$$

Finally, if the $p+1$ roots $\rho_i(\lambda h)$ are such that

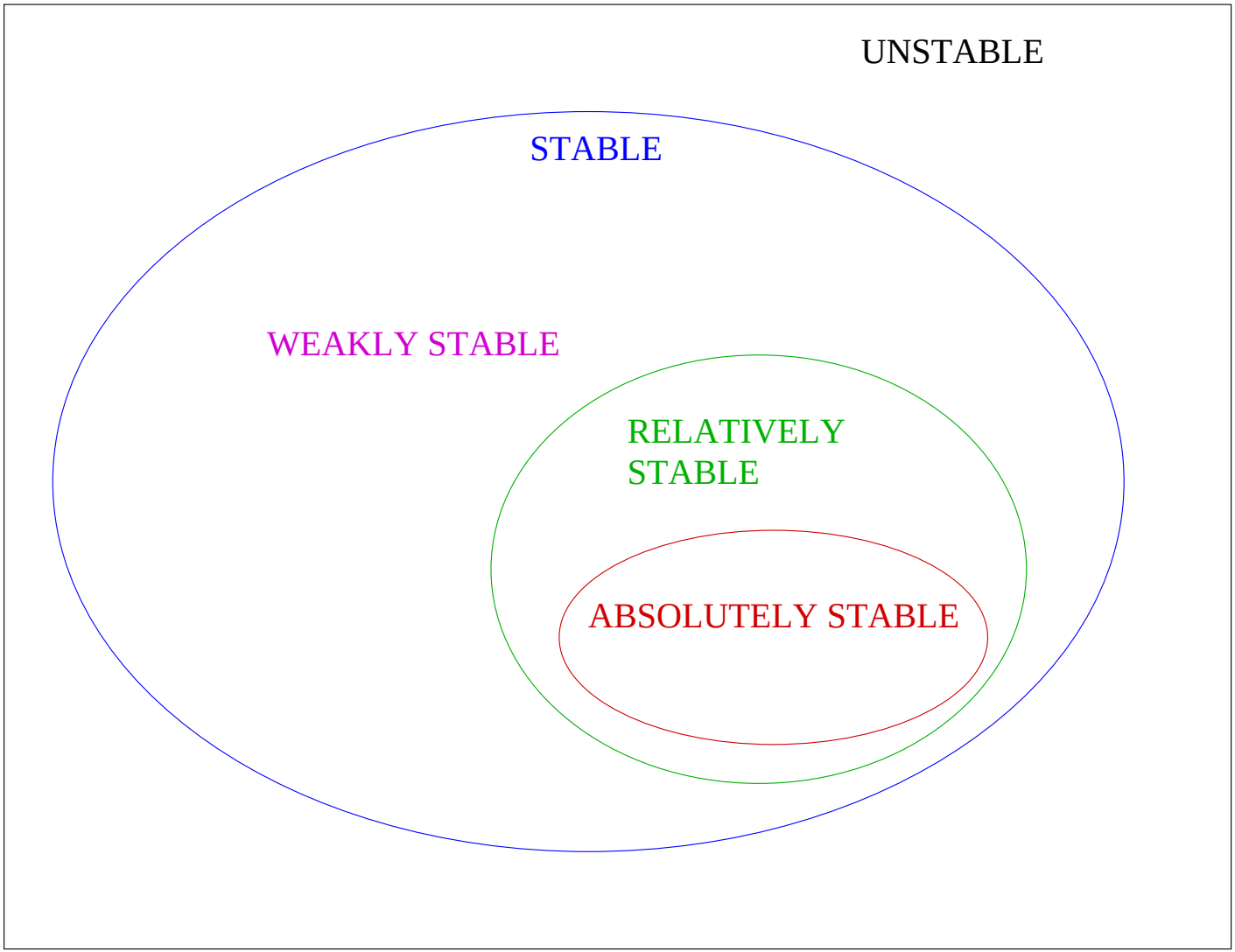
$$|\rho_j(\lambda h)| < 1 \quad j = 0, ..., p$$

we talk of *absolutely stable* method. For our multistep $a_0 = 1, b_0 = 3/2, b_1 = -1/2, p = 1$:

$$\rho_{0,1} = \frac{1 + 3\lambda h/2 \pm \sqrt{(1 + 3\lambda h/2)^2 - 2\lambda h}}{2}$$

A-stable for $-1 < \lambda h < 0$

Stability in cartoons



Generic multistep: implicit methods

$$y_{n+1} = \sum_{j=0}^{p} a_j y_{n-j} + h \sum_{j=-1}^{p} b_j y'_{n-j}$$

These are methods where $b_{-1} \neq 0$. Normally, one guesses the new y_{n+1} via an explicit method (methods which have $b_{-1} = 0$), then a number of iterations are done using the implicit method.

Generic multistep: implicit methods

$$y_{n+1} = \sum_{j=0}^{p} a_j y_{n-j} + h \sum_{j=-1}^{p} b_j y'_{n-j}$$

These are methods where $b_{-1} \neq 0$. Normally, one guesses the new y_{n+1} via an explicit method (methods which have $b_{-1} = 0$), then a number of iterations are done using the implicit method.
Trapezoidal method:

$$y_{n+1} = y_n + \frac{h}{2}(y'_n + y'_{n+1})$$

It is stable, and A-stable for $\lambda < 0$.

Generic multistep: examples

We turn now to some examples, and compare a few different methods to integrate:

- Euler: $y_{n+1} = y_n + hy'_n + O(h^2)$, rs
- Midpoint: $y_{n+1} = y_{n-1} + 2hy'_n + O(h^3)$, ws
- AB(2): $y_{n+1} = y_n + h(3y'_n - y'_{n-1})/2 + O(h^3)$, as
- Trapeizodal: $y_{n+1} = y_n + \frac{h}{2}(y'_n + y'_{n+1}) + O(h^3)$, as

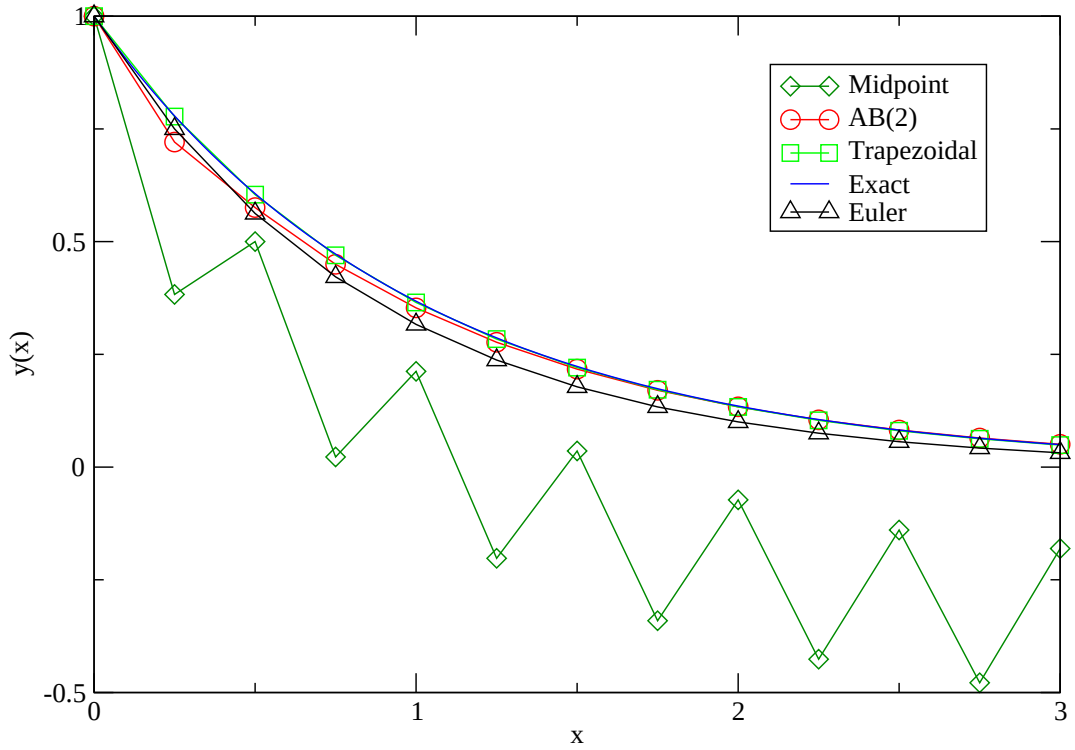
We expect the more stable the method the better the performances

Generic multistep: examples

$$y' = -y$$

$$y(0) = 1$$

$$h = 0.25$$

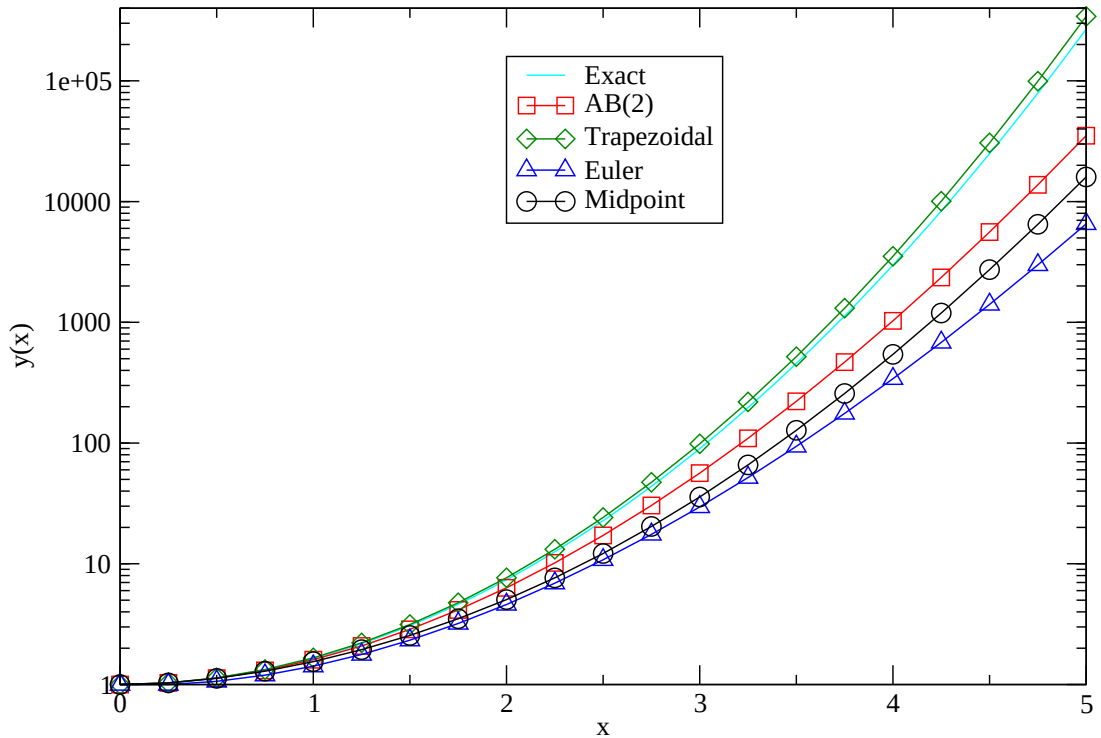


Generic multistep: examples

$$y' = xy$$

$$y(0) = 1$$

$$h = 0.25$$



Multistep, real ones!

Adams-Bashfort: $y_{n+1} = y_n + h \sum_{j=0}^{p} b_j y'_{p-j}$

p	b_0	b_1	b_2	b_3	b_4
0	1				
1	3/2	-1/2			
2	23/12	-16/12	5/12		
3	55/24	-59/24	37/24	-9/24	
4	1901/720	-2774/720	2626/720	-1274/720	251/720

Multistep, real ones!

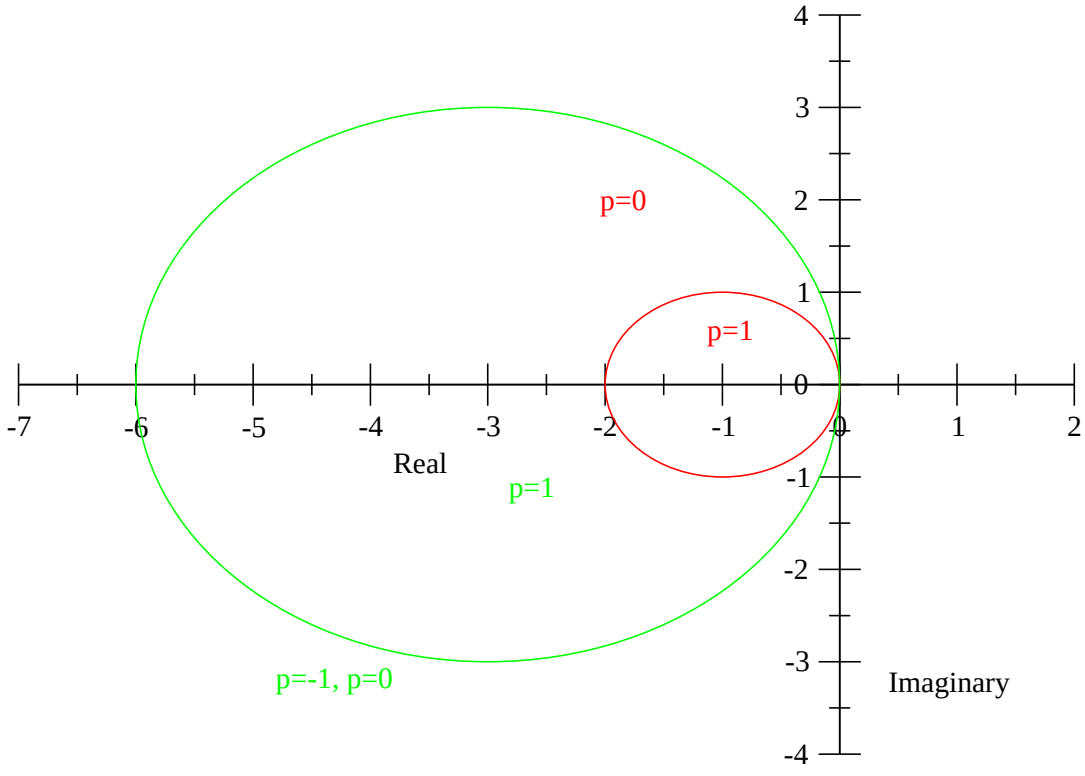
Adams-Moulton: $y_{n+1} = y_n + h \sum_{j=-1}^{p} b_j y'_{p-j}$

p	b_{-1}	b_0	b_1	b_2	b_3
-1	1				
0	1/2	1/2			
1	5/12	8/12	-1/12		
2	9/24	19/24	-5/24	1/24	
3	251/720	646/720	-264/720	106/720	-19/720

Multistep, real ones!

Point is, the stability decreases!

In red, A
schemes; i
green, Al
schemes.
The plot
a plot
$h\lambda$ with
complex.
Low orde
methods a
A-stable f
$\Re(\lambda) < 0$



One step methods: Taylor

Main difference with multisteps is that everything is done within one step. Start from $\dot{y} = f(t, y)$, and $y(t_0) = y_0$, a one step algorithm could be a straight Taylor expansion:

$$y(t+h) = y(t) + f(t,y)h + \frac{h^2}{2}(f'(t,y)f(t,y) + \dot{f}(t,y)) + \ldots$$

One step methods: Taylor

Main difference with multisteps is that everything is done within one step. Start from $\dot{y} = f(t, y)$, and $y(t_0) = y_0$, a one step algorithm could be a straight Taylor expansion:

$$y(t+h) = y(t) + f(t,y)h + \frac{h^2}{2}(f'(t,y)f(t,y) + \dot{f}(t,y)) + \ldots$$

The problem is that this method requires many evaluations of $f(x, y)$. Here 3 evaluations are needed: with 3 evaluations, other methods are better, for instance the trapeizodal method. To add one more term brings in *many* evaluations of the force. It is interesting for polynomial forces.

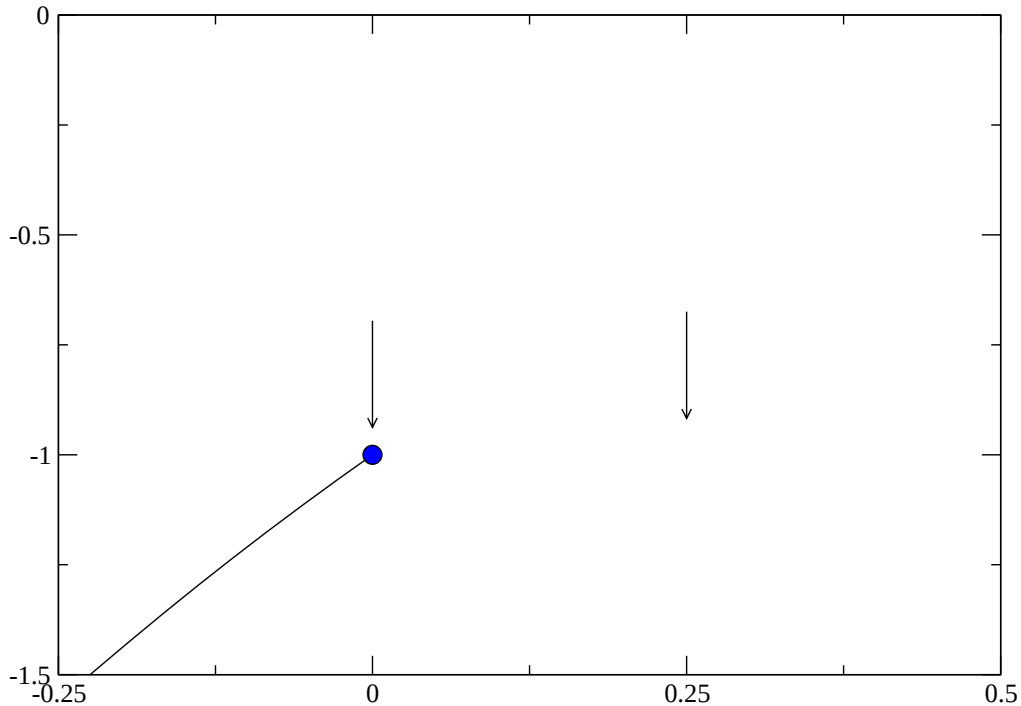
Runge Kutta in cartoons

It is better to show the method in cartoons! Take for example $y' = 2\sqrt{-y}$ with $y(0) = -1$, and use RK

Runge Kutta in cartoons

It is better to show the method in cartoons! Take for example $y' = 2\sqrt{-y}$ with $y(0) = -1$, and use RK

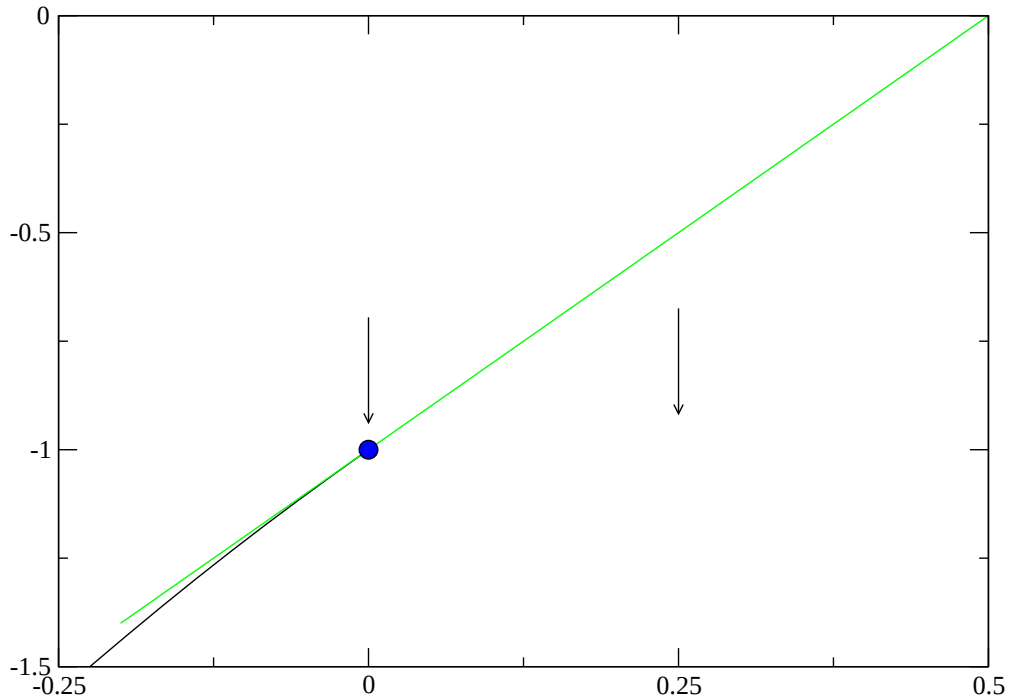
The integration got to $x_0 = 0$, and we need to step to $x_f = 0.25$ (black line, solution so far)



Runge Kutta in cartoons

It is better to show the method in cartoons! Take for example $y' = 2\sqrt{-y}$ with $y(0) = -1$, and use RK

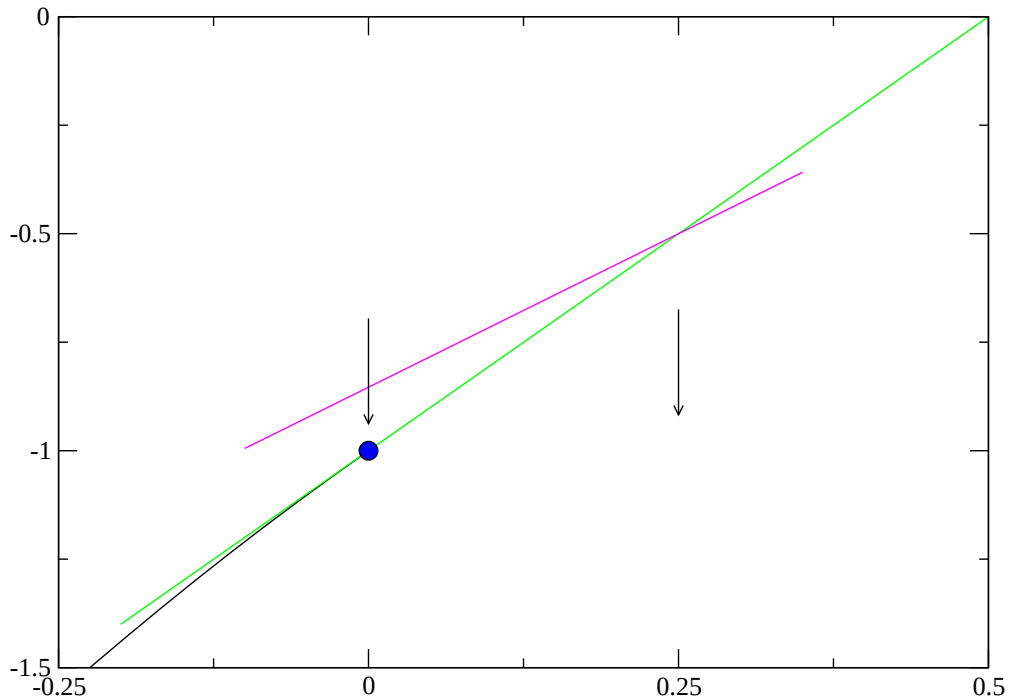
Compute the derivative at x_0 and extrapolate with a line (green line, derivative)



Runge Kutta in cartoons

It is better to show the method in cartoons! Take for example $y' = 2\sqrt{-y}$ with $y(0) = -1$, and use RK

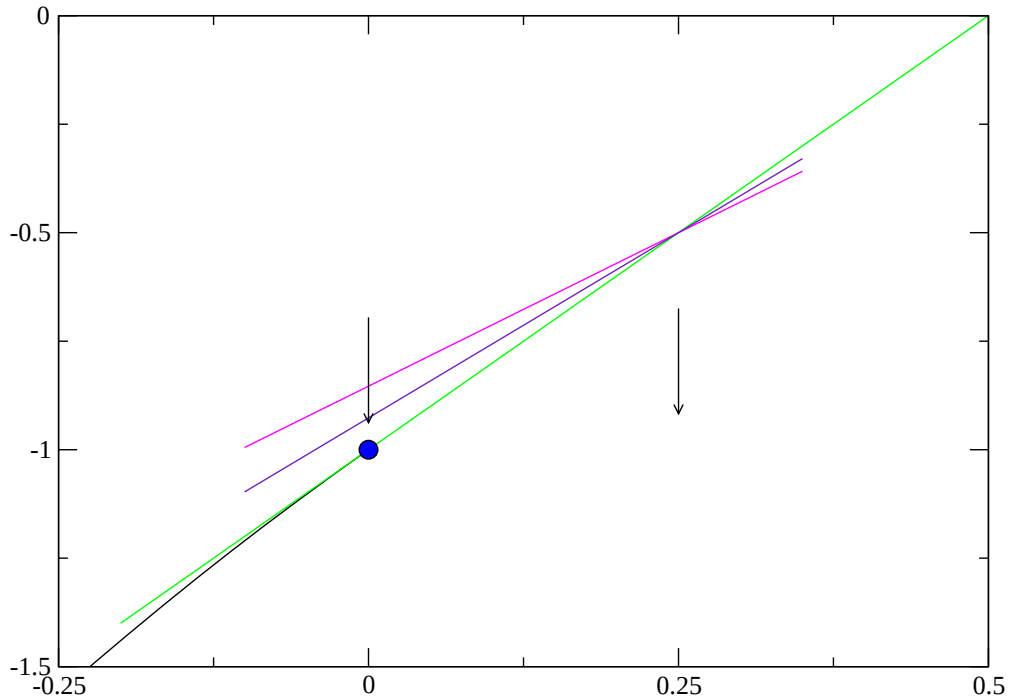
Compute the derivative at x_f (magenta line, derivative at x_f)



Runge Kutta in cartoons

It is better to show the method in cartoons! Take for example $y' = 2\sqrt{-y}$ with $y(0) = -1$, and use RK

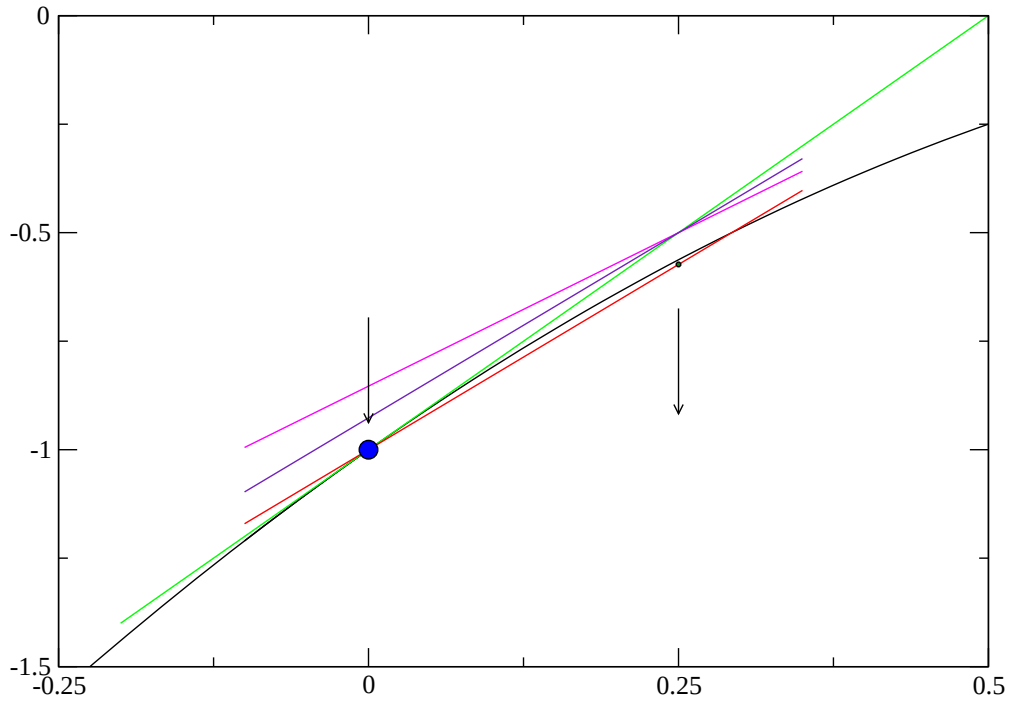
Take average of derivatives (purple line)



Runge Kutta in cartoons

It is better to show the method in cartoons! Take for example $y' = 2\sqrt{-y}$ with $y(0) = -1$, and use RK

Extrapolate from x_0 to x_f with the average derivative (red line extrapolated moving the purple line down to the blu dot, final point green dot; black line, exact)



Runge Kutta in cartoons

It is better to show the method in cartoons! Take for example $y' = 2\sqrt{-y}$ with $y(0) = -1$, and use RK

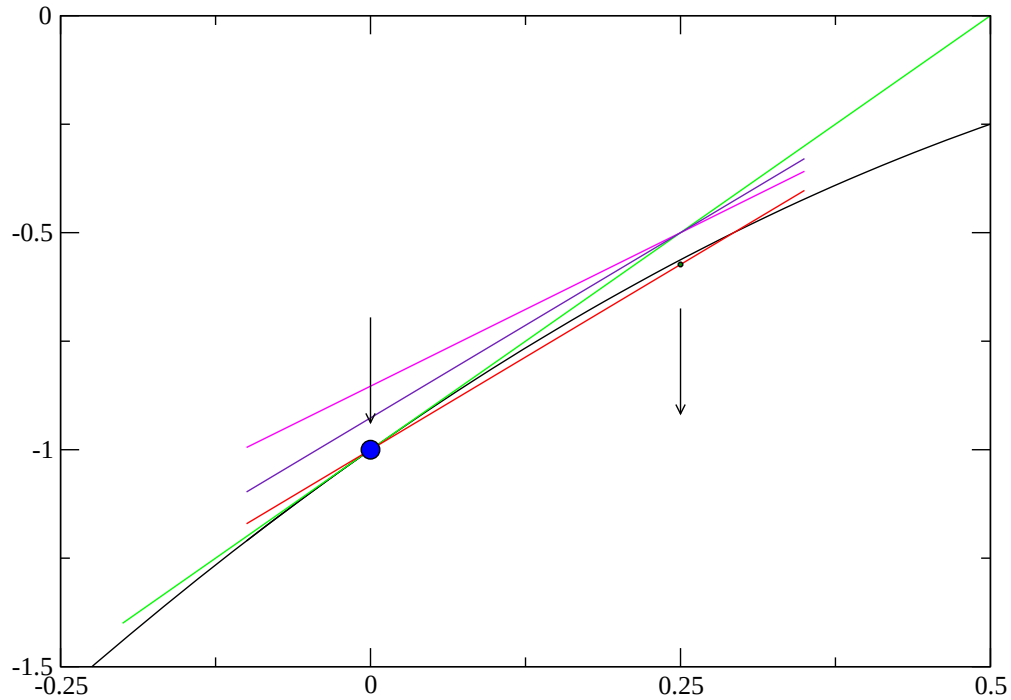
Analytically,

$$k_1 = hf(x_0, y_0)$$

$$k_2 =$$

$$hf(x_0+h, y_0+k$$

$$y_1 = y_0 + \frac{k_1 + k_2}{2}$$



Runge Kutta

There are many RK scheme: perhaps the most used is ($y' = f(x, y)$):

$$k_1 = hf(x_n, y_n)$$

$$k_2 = hf(x_n + h/2, y_n + k_1/2)$$

$$k_3 = hf(x_n + h/2, y_n + k_2/2)$$

$$k_4 = hf(x_n + h, y_n + k_3)$$

$$y_{n+1} = y_n + \frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6} + O(h^5)$$

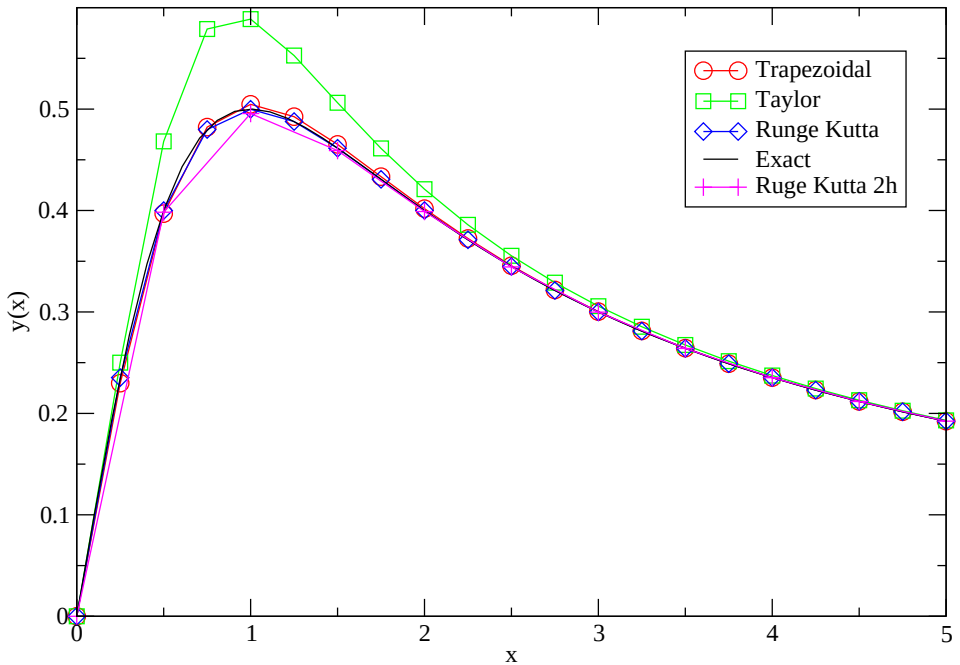
Comparison

Time for some comparison

$$y' = \frac{1}{1+x^2} - 2y^2$$

$$y(0) = 0$$

Note that we should compare the RK with $2h$ with the trapeizodal algorithm because the former requires twice as many force evaluations



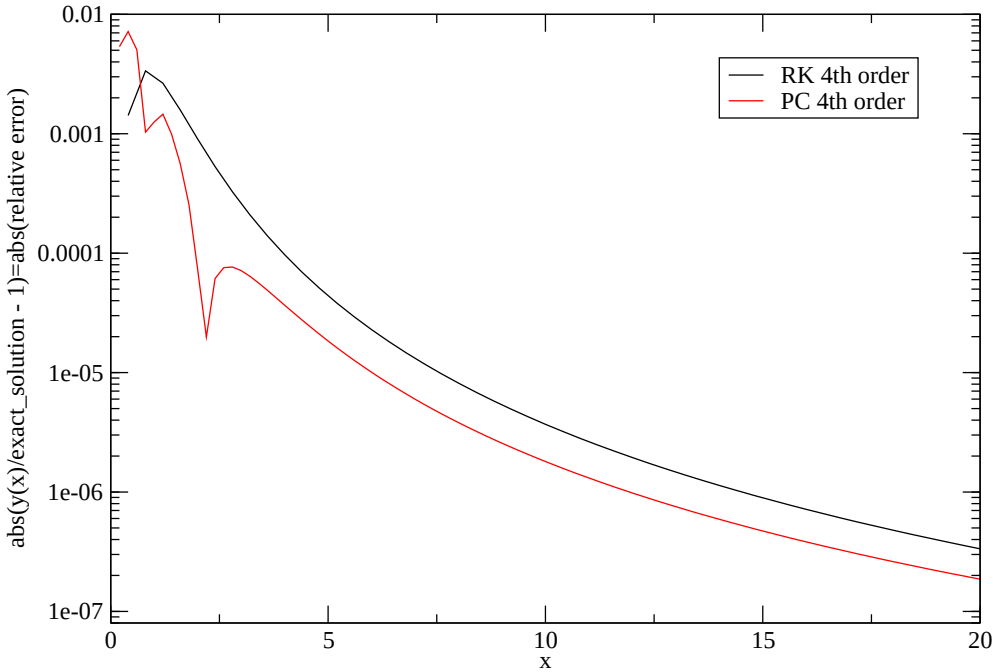
Comparison

Time for some comparison

$$y' = \frac{1}{1+x^2} - 2y^2$$

$$y(0) = 0$$

Comparison between RK and Predictor Corrector AB(p=3)+AM(p=2) h is such that the CPU time is the same in the 2 cases



Step control: Richardson

Use a method extimate the precision. Y is the exact value, y_h the value from the simulations with a step h, using for example the trapezoidal algorithm

$$Y(x_n) - y_h(x_n) = D(x_n)h^2 + O(h^3)$$

$$Y(x_n) - y_{2h}(x_n) = 4D(x_n)h^2 + O(h^3)$$

We get

$$Y(x_n) = [4y_h(x_n) - y_{2h}(x_n)]/3 + O(h^3)$$

and for an extimate of the error

$$Y(x_n) - y_h(x_n) = [y_h(x_n) - y_{2h}(x_n)]/3$$

Step control: Richardson

(from Atkinson, $y' = -y^2 \quad y(0) = 1, h = 0.25$)

x	$y_h(x)$	$y_{2h}(x)$	Error	$Y(x) - y_h(x)$
1.0	.49602	.48314	.00429	.00398
2.0	.33099	.32361	.00246	.00234
3.0	.24852	.24389	.00154	.00148
4.0	.19899	.19484	.00105	.00101
5.0	.16594	.16366	.00076	.00073

Adjust the integration time step until the required (small) error is achieved. Note: only snag, all y_{2h} should be known.

Step control: Richardson

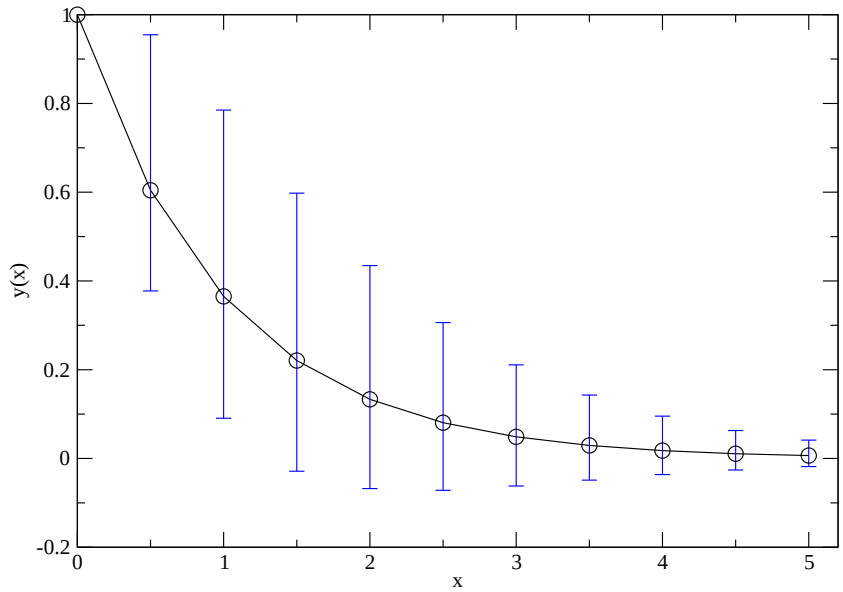
$$y' = -y$$

$$y(0) = 1$$

$$h = 0.25$$

Solution is

$$y(x) = e^{-x}$$



Error bars show $\times 100$ real error: above, Richardson error; below, exact error. Black curve, exact solution.

Modified midpoint

It moves y from x to $x + H$ using steps H/n. $n + 1$ evaluations of the force are required. The algorithm is ($y' = f(x, y)$)

$$z_0 = y(x)$$

$$z_1 = z_0 + hf(x, z_0)$$

$$z_{m+1} = z_{m-1} + 2hf(x + mh, z_m) \quad m = 1, 2, ..., n - 1$$

$$y(x + H) \approx y_n = \frac{1}{2}\left[z_n + z_{n-1} + hf(x + H, z_n)\right]$$

(n here is the number of steps in H)

Modified midpoint

The interesting thing about the method is that

$$y_n - y(t+H) = \sum_{i=1}^{\infty} \alpha_i h^{2i}$$

the error contains only even powers of h, as h is changed. The extimate

$$y(x+H) \approx \frac{4y_n - y_{n/2}}{3}$$

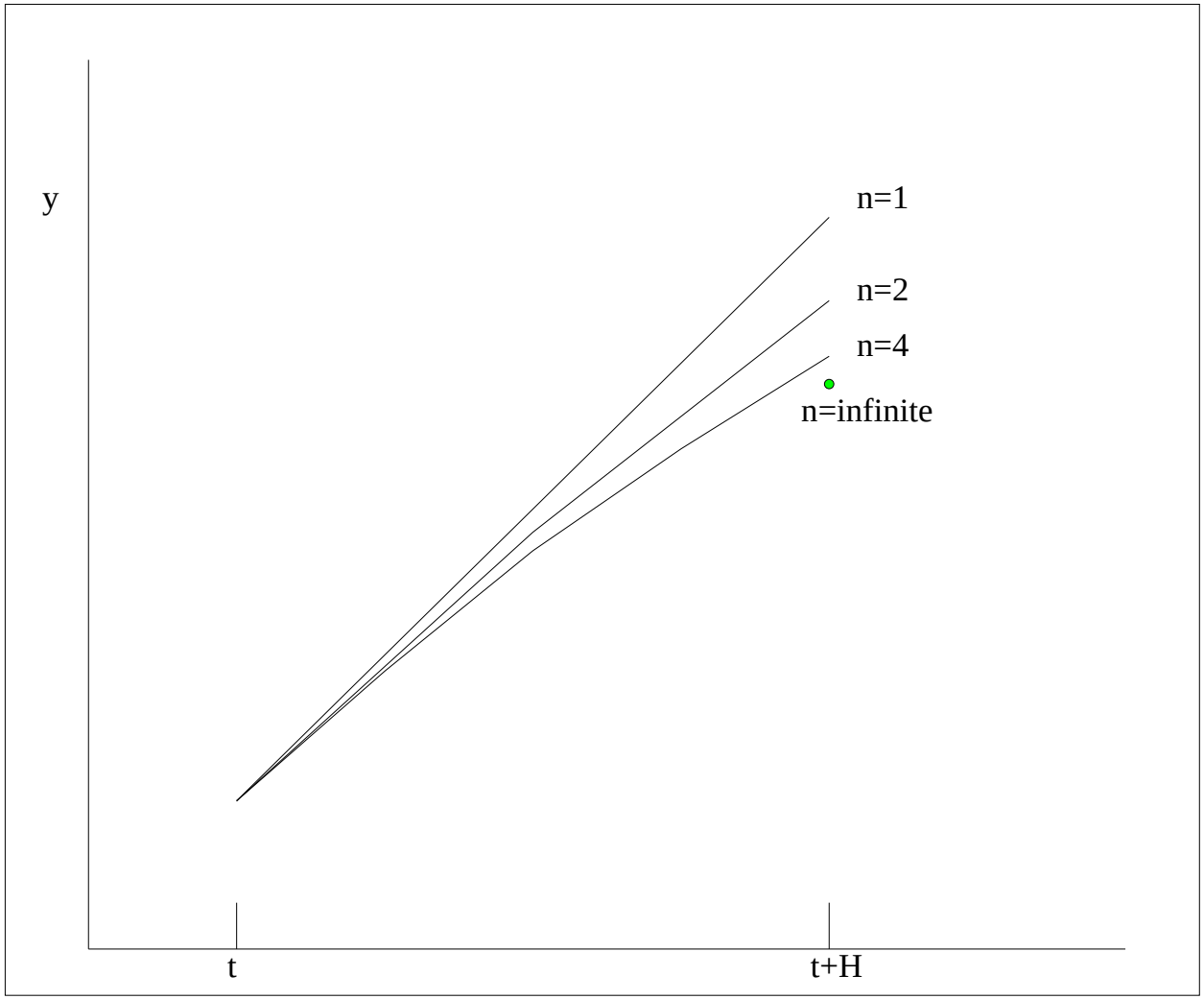
is fourth order accurate! This requires 1.5 evaluations of the force opposed to the 4 evaluations for a RK.

Richardson extrapolation

The idea is that the new point in the integration is some (complicated) analytic function of the integration time step, so it make sense to compute the point for $h \longrightarrow 0$. Take a large stepsize H, and compute the solution between t and $t + H$ with one step of an elementary algorithm; then take two steps within $[t, t + H]$, four steps, etc.. We are taking integration time steps which go like H/n, where n is the number of elementary steps. The new point is obtained extrapolating $n \longrightarrow \infty$.

Richardson extrapolation

The new point is indicated by the colored dot



Richardson extrapolation

A very nice implementation is Burlish-Stoer.

- Use a modified midpoint as the elementary algorithm to integrate
- Use a rational function to extrapolate: namely, something in the form

$$R_{i(i+1)\ldots(i+m)} = \frac{p_0 + p_1 x + \ldots + p_\mu x^\mu}{1 + q_1 x + \ldots + q_\nu x^\nu}$$

with $m + 1 = 1 + \mu + \nu$ unknowns and $x \equiv H/n$. There are efficient ways to find the unknown coefficients (NumRec)

Richardson extrapolation

Details are nitty gritty, and the best thing is to take the routine for instance from Numerical Recipes. In particular, the choice of the integration time step H is not trivial. The notable point is that the method is very fast and accurate. It fails if there are discontinuities close to the integration path. Also, it is positively not very efficient if the force is given as a look up table. It might be slow if the force is complicated (better use a Predictor Correct in that case)

Hamiltonian RK

There are cases when RK are very valuable. In Hamiltonian formulation, one starts from something like

$$H = \frac{p^2}{2} + V(q)$$

and the idea is to integrate the equations of motion starting from some $p(0)$ and $q(0)$. We could use any algorithm, but the structure of RK (relating things at two times) suggests that we could build a contact transformation via a RK.

Hamiltonian RK

$$H = \frac{p^2}{2} + V(q)$$

A contact transformation is a mapping from q, p to Q, P: the latter can be thought of as the evolution at some later time of q, p. The important thing is that a contact transformatio preserves the symplectic (hamiltonian) structure of the H flow. To know more about contact transformation and symplectic structures, see Landau and/or Goldstein (Mechanics books).

Hamiltonian RK

$$H = \frac{p^2}{2} + V(q)$$

A generic symplectic RK of order n has the form ($F(q) = -V'(q)$)

$$p_j = p_{j-1} + h b_j F(q_{j-1})$$

$$q_j = q_{j-1} + h a_j p_j$$

with p_0, q_0 initial point and p_n, q_n evoluted point after a time h. We need some sets of *magic* a_i and b_i, which can be obtained writing explicitely the scheme, and imposing it preserves H.

Hamiltonian RK

In practice, for instance

$$p_j = p_{j-1} + hb_j F(q_{j-1}) \quad q_j = q_{j-1} + ha_j p_j$$

take $b_1 = 0, b_2 = 1$, $a_1 = a_2 = 1/2$, we have

$$\tilde{q} = q_0 + hp_0/2$$

$$p_2 = p_0 + hF(\tilde{q})$$

$$q_2 = \tilde{q} + hp_2/2$$

substitute in H, and it turns out that H is conserved up to $O(h^3)$.

Hamiltonian RK

$$p_j = p_{j-1} + h b_j F(q_{j-1}) \quad q_j = q_{j-1} + h a_j p_j$$

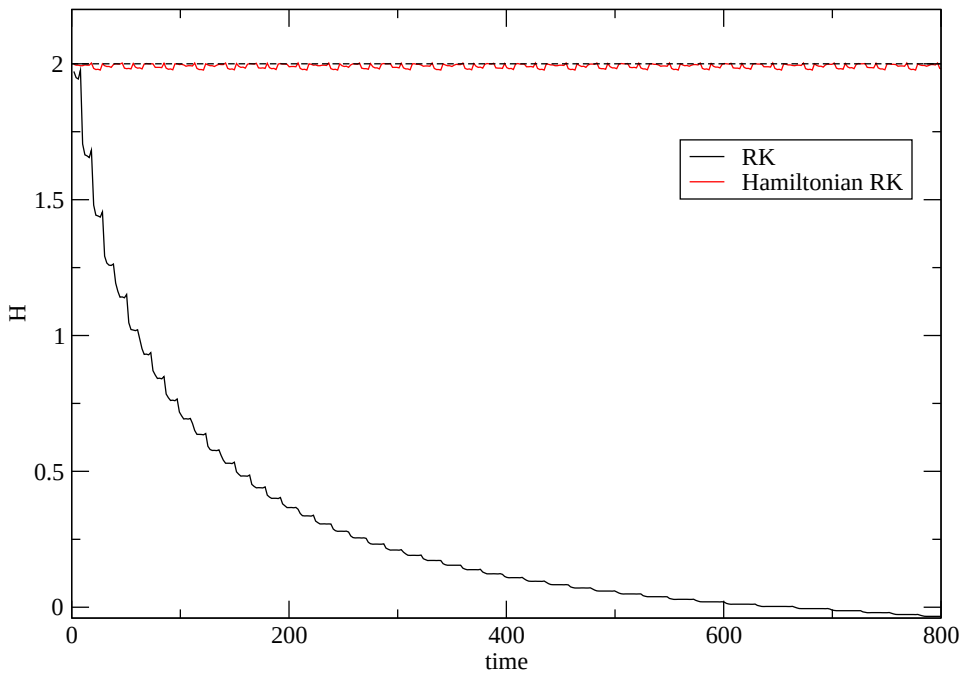
i	a_i	b_i
1	0.515 352 837 431 123	0.134 496 199 277 431
2	-0.085 782 019 412 974	-0.224 819 803 079 421
3	0.441 583 023 616 467	0.756 320 000 515 668
4	0.128 846 158 365 384	0.334 003 603 286 321

This is a possible set of *magic* coefficients for a quadratic Hamiltonian.

Hamiltonian RK

Let us see the method in practice: $H = p^2/2 + x^4/4 - x^2/2$

Comparison between a RK and a HRK, fourth order. $h = 0.5$, $H = 2$



Stiff equations: examples

Let us see some examples:

$$u' = 998u + 1998v \qquad u(0) = 1$$

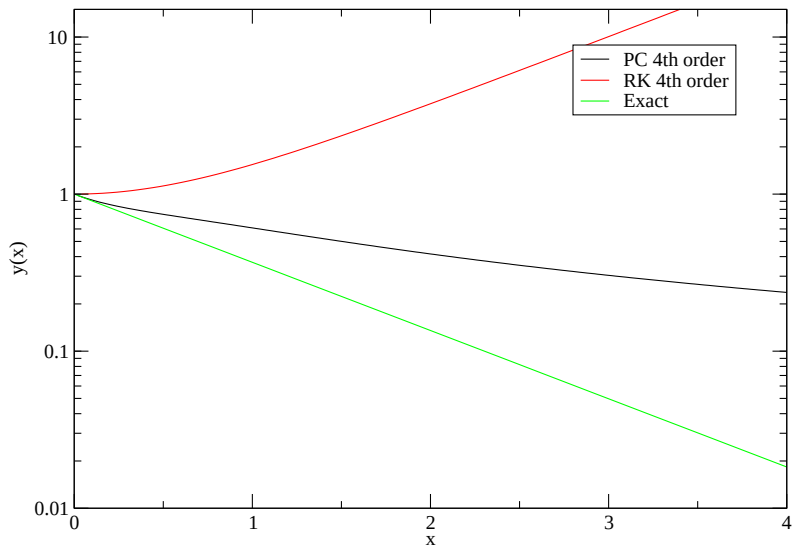
$$v' = -999u - 1999v \qquad v(0) = 0$$

The solution is $u = 2e^{-x} - e^{-1000x}$ and $v = -e^{-x} + e^{-1000x}$. It is clear that the integration time step should accomodate for the two (widely) different time scales, and hence "standard" approaches are doomed to fail. Either we end up in roundoff for one of the term, or shoot widely for the other!

Stiff equations: examples

Let us see some examples: $y' = y - 2e^{-x} \quad y(0) = 1$

The solution for the given initial condition is $y(x) = e^{-x}$ but the general solution is $y(x) = Ae^{x} + e^{-x}$



Stiff equations: integration

The cure is to use some low order implicit/semi-implicit method. Suppose we have $\mathbf{y}' = \mathbf{f}(x, \mathbf{y})$, it is solved

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h\mathbf{f}(x_{n+1}, \mathbf{y}_{n+1})$$

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h\left[\mathbf{f}(x_{n+1}, \mathbf{y}_n) + \frac{\partial \mathbf{f}}{\partial \mathbf{y}_n}(\mathbf{y}_{n+1} - \mathbf{y}_n)\right]$$

$$\mathbf{y}_{n+1} = \left[1 - h\frac{\partial \mathbf{f}}{\partial \mathbf{y}_n}\right]^{-1}(\mathbf{y}_n + h\mathbf{f}(x_{n+1}, \mathbf{y}_n) - h\frac{\partial \mathbf{f}}{\partial \mathbf{y}_n}\mathbf{y}_n)$$

Stiff equations: integration

Formally, from the equation ($\mathbf{A}$ matrix positively defined, $\lambda_i > 0 \ \ \forall i$), given $\mathbf{y}' = -\mathbf{A}\mathbf{y}$

$$\mathbf{y}_{n+1} = (\mathbf{1} - h\mathbf{A})\mathbf{y}_n = \mathbf{B}\mathbf{y}_n$$

and $\mathbf{B}^n \to 0$ for $n \to \infty$ only if all eigenvalues $(1 - \lambda h)$ of $\mathbf{B}$ are smaller than 1: typically this is not the case for stiff equations. However, using implicit methods

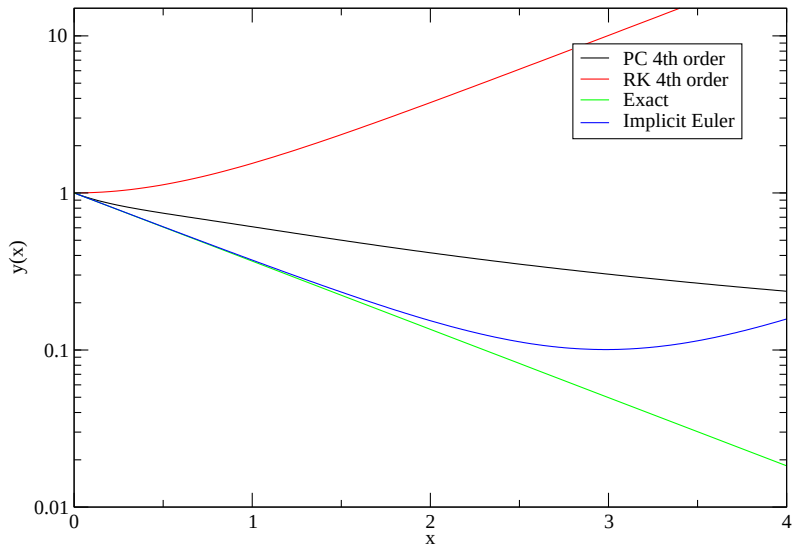
$$\mathbf{y}_{n+1} = \mathbf{y}_n - h\mathbf{A}\mathbf{y}_{n+1} \quad \to \quad (\mathbf{1} + h\mathbf{A})\mathbf{y}_{n+1} = \mathbf{y}_n$$

If the eigenvalues of $\mathbf{A}$ are $\lambda > 0$, the eigenvalues of $(\mathbf{1} + h\mathbf{A})^{-1}$ are $(1 + h\lambda)^{-1} < 1$ for all h, and then well behaved.

Stiff equations: integration

Back to $y' = y - 2e^{-x} \quad y(0) = 1$

The solution for the given initial condition is $y(x) = e^{-x}$ but the general solution is $y(x) = Ae^{x} + e^{-x}$



An implicit Euler method follows much more closely the solution.

Boundary Value Problems

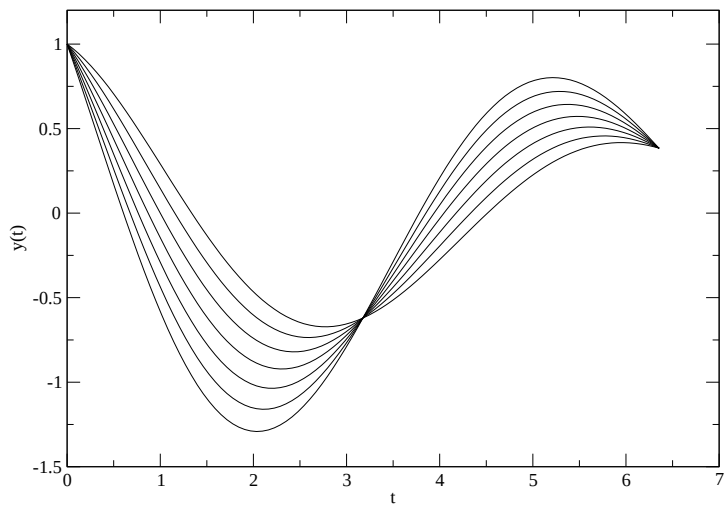
There is a problem: the solution is not unique.

Boundary Value Problems

There is a problem: the solution is not unique. Example:

$$\ddot{y} = -\gamma\dot{y} - \omega_0^2 y \quad y(0) = 1$$

Boundaries set are at $t = 0$ and $t = k\pi/\omega$ with $\omega^2 = \omega_0^2 - (\gamma/2)^2$, solution not unique.

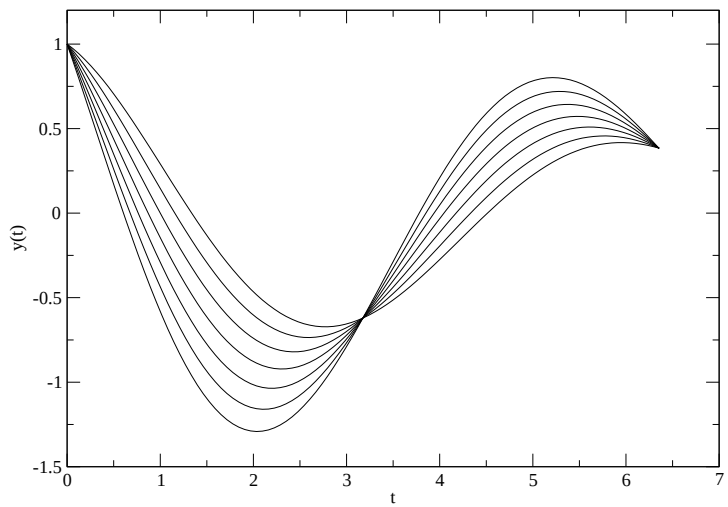


Boundary Value Problems

There is a problem: the solution is not unique. Example:

$$\ddot{y} = -\gamma\dot{y} - \omega_0^2 y \quad y(0) = 1$$

Boundaries set are at $t = 0$ and $t = k\pi/\omega$ with $\omega^2 = \omega_0^2 - (\gamma/2)^2$, solution not unique. It is not known a priori if this is the case in real situations.



BVP: how to proceed?

Assume the problem is well defined. How can we proceed? In general, it is a difficult task. There are many methods, but perhaps the most used ones are:

Shooting start from x_0 and the known values for y and its derivatives, vary the unknown values in x_0 until the BV at x_f are reached.

Relaxation (also known as finite differences method) put a tentative solution between x_0 and x_f (initial guess), and relax it until the ODE is satisfied.

There are routines available: on netlib.no, TWPBVP is an example of the latter method.

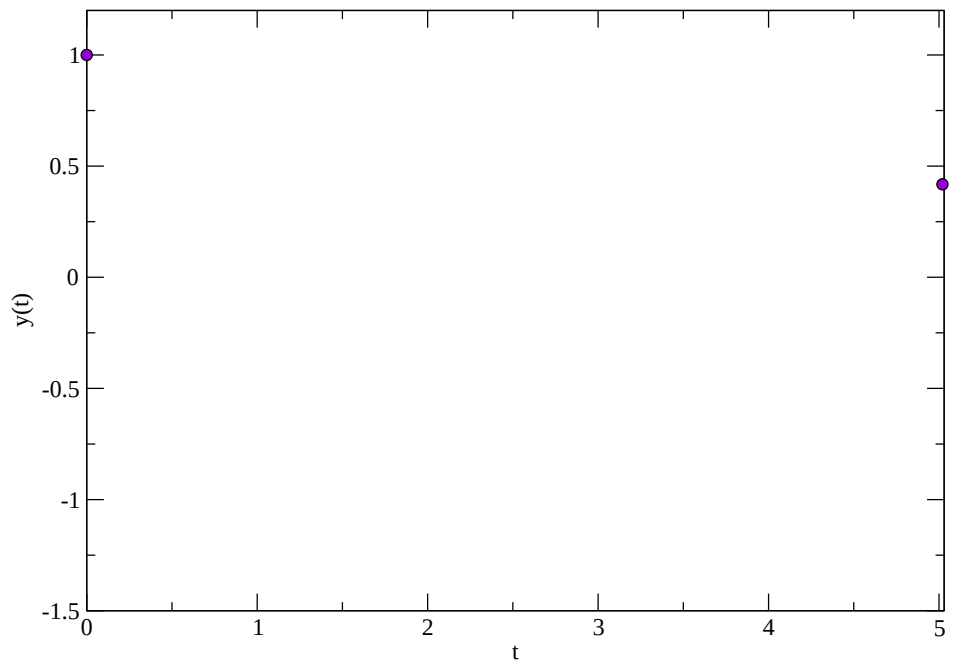
BVP: shooting method

Take again $\ddot{y} = -\gamma\dot{y} - \omega_0^2 y$, $\gamma = 0.3$, $\omega_0 = 1$

BVP: shooting method

Take again $\ddot{y} = -\gamma\dot{y} - \omega_0^2 y$, $\gamma = 0.3$, $\omega_0 = 1$

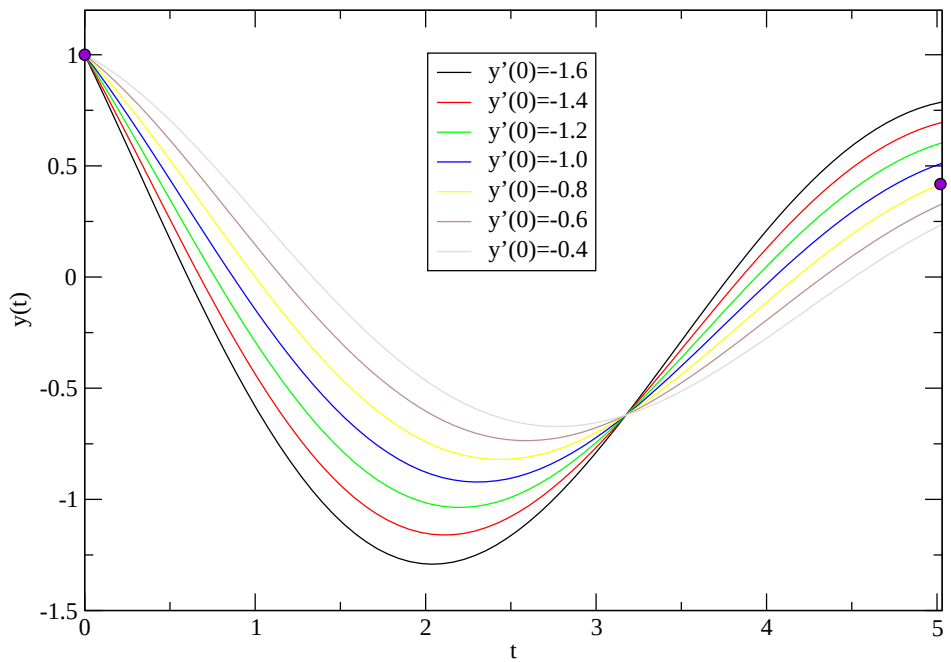
Boundaries are set at $y(0) = 1$, $y(5.02) = 0.418$.



BVP: shooting method

Take again $\ddot{y} = -\gamma\dot{y} - \omega_0^2 y$, $\gamma = 0.3$, $\omega_0 = 1$

Boundaries are set at $y(0) = 1$, $y(5.02) = 0.418$. Change the value of $\dot{y}(0)$ and integrate, until the desired value for $y(5.02)$ is obtained: in this case, $\dot{y}(0) = -0.8$.



BVP: relaxation

Take for example $y'' = f(x, y, y')$, with boundaries $y(x_0) = g_0$ and $y(x_n) = g_n$. Assume a mesh of n points in x, spaced h. Then the equation becomes

$$\frac{y_{n-1} - 2y_n + y_{n+1}}{h^2} = f(x_n, y_n, \frac{y_{n+1} - y_{n-1}}{2h}) \equiv f_n$$

with y_0 and y_n given by the BV.
This has the general form (please, try to remember the structure above!)

BVP: relaxation

$$\frac{1}{h^2}\begin{bmatrix} -2 & 1 & 0 & & \dots & 0 \\ 1 & -2 & 1 & & & \vdots \\ \vdots & & & & & \\ & & & 1 & -2 & 1 \\ 0 & \dots & & 0 & 1 & -2 \end{bmatrix}\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ \\ y_{n-1} \end{bmatrix} =$$

$$\begin{bmatrix} f_1 \\ f_2 \\ \vdots \\ \\ f_{n-1} \end{bmatrix} - \begin{bmatrix} g_0/h^2 \\ 0 \\ \vdots \\ 0 \\ g_n/h^2 \end{bmatrix}$$

BVP: relaxation

which can be written as

$$\frac{1}{h^2}\mathbf{A}Y = \mathbf{F}(Y) + \mathbf{G}$$

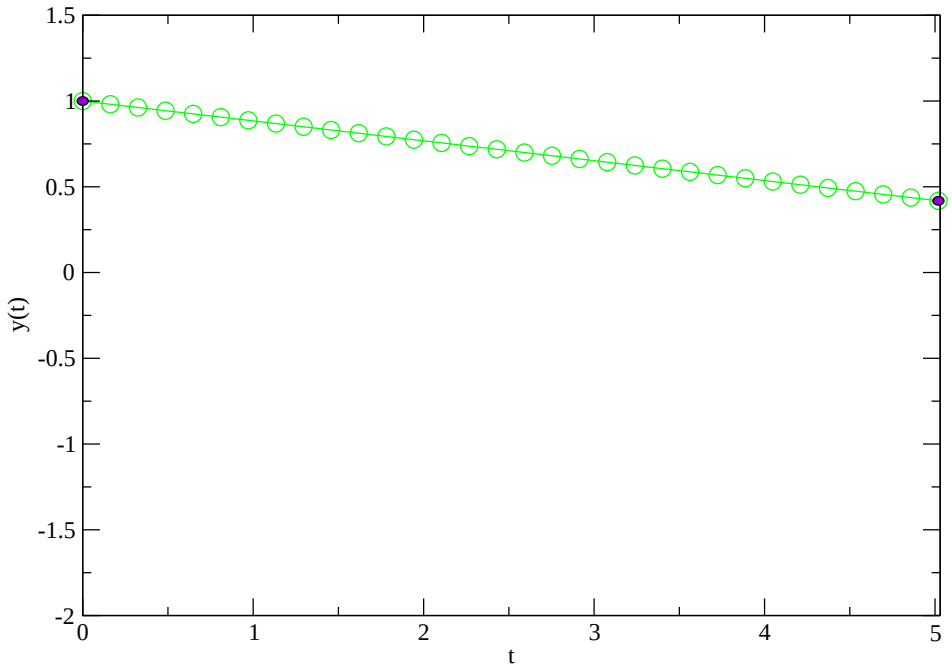
It can be solved using Newton method, after linearising $F(Y)$.

$$\mathbf{y}^{(m+1)} = \mathbf{y}^{(m)} - \left[\frac{\mathbf{A}}{h^2} - \frac{\partial \mathbf{F}_i}{\partial y_j}\right]^{-1} [\frac{\mathbf{A}y^{(m)}}{h^2} - \mathbf{F}(y^{(m)}) - \mathbf{G}]$$

Take an initial guess for $y^{(0)}$ and iterate. It is vital that we used Newton method, otherwise the method would not converge!

BVP: relaxation

Take again $\ddot{y} = -\gamma\dot{y} - \omega_0^2 y$, $\gamma = 0.3$, $\omega_0 = 1$, boundaries are $y(0) = 1$, $y(5.02) = 0.418$. Initial guess is a straight line.



BVP: relaxation

Seven iterations, and the solution is readly found. We used a mesh of 30 points (which means that $h \approx 0.17$).