Lecture Notes
Multibody Dynamics B, wb1413

A. L. Schwab & Guido M.J. Delhaes

Laboratory for Engineering Mechanics
Mechanical Engineering
Delft University of Technolgy
The Netherlands

April 9, 2009

Contents

Chapter 7

Numerical Integration of Ordinary Differential Equations

Up until now solved for $\ddot{\mathbf{q}}$ from

$$\bar{\mathbf{M}}\ddot{\mathbf{q}} = \bar{\mathbf{Q}}(\mathbf{q}, \dot{\mathbf{q}}, t) \tag{7.1}$$

or

$$\begin{pmatrix} \mathbf{M} & \mathbf{D}^T \\ \mathbf{D} & \mathbf{0} \end{pmatrix} \begin{pmatrix} \ddot{\mathbf{x}} \\ \boldsymbol{\lambda} \end{pmatrix} = \begin{pmatrix} \mathbf{f} \\ -\mathbf{D}^2\dot{\mathbf{x}}\dot{\mathbf{x}} \end{pmatrix} \tag{7.2}$$

But what we really want to know is $\mathbf{q}(t)$.
So given t_0, $\mathbf{q}_0$ and $\dot{\mathbf{q}}_0$ and $\bar{\mathbf{M}}\ddot{\mathbf{q}}_0 = \bar{\mathbf{Q}}(\mathbf{q}_0, \dot{\mathbf{q}}_0, t_0)$ we want to know $\mathbf{q}_1$ and $\dot{\mathbf{q}}_1$ for time $t_1 = t_0 + h$.
A natural, but naive, way would be,

$$\begin{aligned} \mathbf{q}_1 &= \mathbf{q}_0 + h\dot{\mathbf{q}}_0, \\ \dot{\mathbf{q}}_1 &= \dot{\mathbf{q}}_0 + h\ddot{\mathbf{q}}_0. \end{aligned}$$

Which finds its origin in the Taylor expansion of a function $\mathbf{f}(\mathbf{x})$.

$$\mathbf{f}(\mathbf{x}+h) = \mathbf{f}(\mathbf{x}) + h\mathbf{f}'(\mathbf{x}) + \frac{1}{2}h^2\mathbf{f}''(\mathbf{x}) + \cdots + \frac{1}{n!}h^n\mathbf{f}^n(\mathbf{x}) + O(h^{n+1}) \tag{7.3}$$

Knowing this you could raise the question if it were not better to use,

$$\mathbf{q}_1 = \mathbf{q}_0 + h\dot{\mathbf{q}}_0 + \frac{1}{2}h^2\ddot{\mathbf{q}}_0 \quad ?$$

An ordinary differential equation (ODE) is usually expressed as, $\mathbf{y}' = \mathbf{f}(\mathbf{x}, \mathbf{y})$ with initial values $\mathbf{y}(\mathbf{x}_0) = \mathbf{y}_0$. We have, $\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y})$ with the initial values at $t = t_0$, $\mathbf{y}(t_0) = \mathbf{y}_0$. Well actually we have a set of n second order differential equations,

$$\bar{\mathbf{M}}\ddot{\mathbf{q}} = \bar{\mathbf{Q}}(t, \mathbf{q}, \dot{\mathbf{q}}), \tag{7.4}$$

with, $\mathbf{q}(t_0) = \mathbf{q}_0$ and $\dot{\mathbf{q}}(t_0) = \dot{\mathbf{q}}_0$. Which we can transform into first order differential equations by substitution of $\mathbf{u} = \dot{\mathbf{q}}$ into (7.4),

$$\bar{\mathbf{M}}\dot{\mathbf{u}} = \bar{\mathbf{Q}}(t, \mathbf{q}, \mathbf{u}).$$

This lead to a set 0f $2n$ first order differential equations,

$$\begin{array}{ll} \dot{\mathbf{q}} = \mathbf{u} & \mathbf{q}(t_0) = \mathbf{q}_0 \\ \dot{\mathbf{u}} = \bar{\mathbf{M}}^{-1}\bar{\mathbf{Q}}(t, \mathbf{q}, \mathbf{u}) & u(t_0) = u_0, \end{array} \tag{7.5}$$

which we can write in the standard first order form

$$\dot{\mathbf{y}} = \mathbf{f}(t, \mathbf{y}) \quad \mathbf{y}(t_0) = \mathbf{y}_0 \quad \text{with} \quad \mathbf{y} = \begin{pmatrix} \mathbf{q} \\ \mathbf{u} \end{pmatrix}. \tag{7.6}$$

Lets look at a simple example, a mass-spring-damper system. The equation of

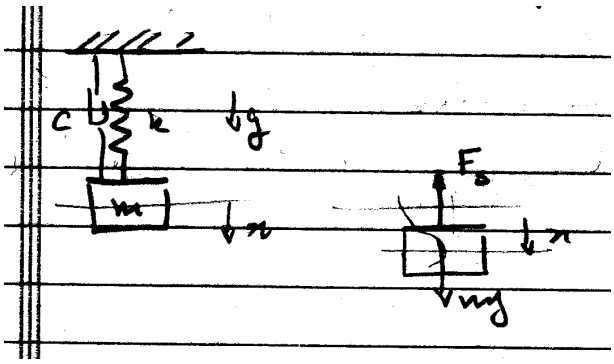

Figure 7.1: Sketch of a mass spring damper system

motion for this system with linear spring-damper is,

$$m\ddot{x} = mg - F_s - F_d = mg - kx - c\dot{x}.$$

Substitute $\dot{x} = u$ in the previous equation results in,

$$\dot{u} = g - \frac{k}{m}x - \frac{c}{m}u.$$

And rewritten in the standard first order form $\dot{\mathbf{y}} = \mathbf{g}(t, \mathbf{y})$,

$$\begin{pmatrix} \dot{x} \\ \dot{u} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -\frac{k}{m} & -\frac{c}{m} \end{pmatrix} \begin{pmatrix} x \\ u \end{pmatrix} + \begin{pmatrix} 0 \\ g \end{pmatrix}. \tag{7.7}$$

And finally with initial conditions for example starting from rest, $x = 0$ and $\dot{x} = 0$, or $\mathbf{y}_0 = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$.

7.1 Euler step

The first method, $\mathbf{y}_{n+1} = \mathbf{y}_n + h\mathbf{f}(t_n, \mathbf{y}_n)$,is known as an Euler step. A somewhat more refined method is,

$$\begin{aligned} \mathbf{y}^*_{n+1} &= \mathbf{y}_n + h\mathbf{f}(t_n, \mathbf{y}_n) \\ \mathbf{y}_{n+1} &= \mathbf{y}_n + \frac{h}{2}\left(\mathbf{f}(t_n, \mathbf{y}_n) + \mathbf{f}(t_{n+1}, \mathbf{y}^*_{n+1})\right) \end{aligned}$$

or Heun's method.
This method takes a sort of average of $\mathbf{f}(\mathbf{y}')$ into account. Note that predictor $\mathbf{y}^*_{n+1}$ is a normal Euler step. What do we mean by more refined? $\Rightarrow$ More

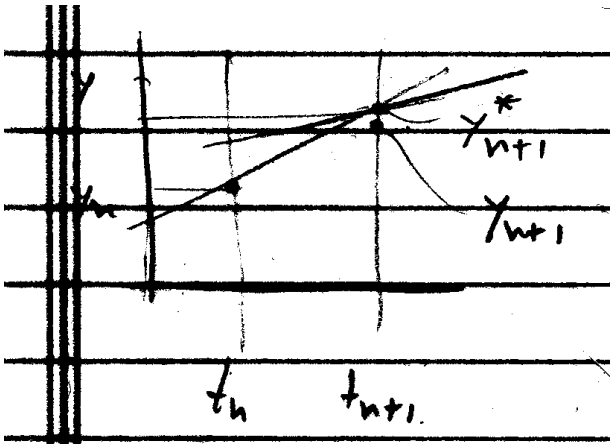

Figure 7.2: Sketch of the Heun integration method

accurate results! So lets look at the accuracy of $\mathbf{y}$ after one step, the so-called truncation error. The truncation error ϵ (after one step) for the two methods follows from the Taylor expansion and are,

$$\begin{array}{ll} \text{Euler} & \epsilon = O\left(h^2\right) \\ \text{Heun} & \epsilon = O\left(h^3\right) \end{array}$$

Far more important is the error in final result, or the so-called global error. This global truncation error E is the result after integration over the time-span $t = 0 \cdots T$, which is the sum of (T/h) local true errors,

$$E = \left(\frac{T}{h}\right) \epsilon$$

So the global truncation error for the two methods are,

$$\begin{array}{ll} \text{Euler} & E = O\left(h\right) \\ \text{Heun} & E = O\left(h^2\right) \end{array}$$

These are all order of magnitude things. In practise we usually do not know the value of this error and sometimes even not know the order of magnitude. But we can estimate the global error. Lets look at a simple one-dimensional problem. We assume that the approximate value is

$$y = \hat{y} + C_1 h^p, \tag{7.8}$$

with the true answer (which we do not know) $\hat{y}$ and the truncation error $C_1 h^p$. For Euler $p = 1$ and for Heun $p = 2$. This equation has two unknows, the true value $\hat{y}$ and the coefficients C_1. We can determine these values by integration the system twice with two different stepsizes. We first integrate from $t = 0$ to $t = T$ with step size h , call this result y_h. Next, we intergate again from $t = 0$

to $t = T$, but now with half the stepsize $h/2$ and we call this $y_{h/2}$. This results in the following two equations,

$$\begin{aligned} y_h &= \hat{y} + C_1 h^p \\ y_{h/2} &= \hat{y} + C_1 \left(\frac{h}{2}\right)^p. \end{aligned}$$

From which we can solve $\hat{y}$ and C_1. But since these are error bounds, plus or minus, we can only give a global truncation error on our best result $y_{h/2}$,

$$E = |\hat{y} - y_{h/2}| = \frac{1}{2^p - 1}|(y_{\frac{h}{2}} - y_h)| \tag{7.9}$$

So this was accuracy. Far more important is stability, or how do previous made errors propagate? To investigate this we need a test equation. We will again look at a simple one-dimensional case and use the most generic and archetypal differential equation,

$$\dot{y} = \lambda y. \tag{7.10}$$

The general solution to this equation is $y = ce^{\lambda t}$ and with complex $\lambda = a + bi$ and complex c we have the solutions of the form,

$$y = ce^{(a+bi)t} = ce^{at}e^{bit}.$$

Which we transform with the Euler identities $e^{i\phi} = \cos\phi + i\sin\phi$ into sines and cosines, as

$$y = ce^{at}\left(\cos(bt) + i\sin(bt)\right)$$

We see that e^{at} gives rise to growing or decaying solutions and that $\cos(bt) + i\sin(bt)$ is the oscillating part.

How does this connect to our mechanical systems?
Lets look at the mass-spring-damper system again. The equation of motion is:

$$m\ddot{x} + c\dot{x} + kx = f$$

Look at the reduced equations for the homogenous solution $m\ddot{x} + c\dot{x} + kx = 0$ and introduce the eigenfrequency ω_0, from $\omega_0^2 = k/m$, the relative damping β, from $2\beta\omega_0 = c/m$, which results in

$$\ddot{x} + 2\beta\omega_0\dot{x} + \omega_0^2 x = 0.$$

Transform this second order ODE into 2 first order equations with $v = \dot{x}$

$$\begin{pmatrix} \dot{x} \\ \dot{v} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -\omega_0^2 & -2\beta\omega_0 \end{pmatrix} \begin{pmatrix} x \\ v \end{pmatrix} \tag{7.11}$$

$$\dot{\mathbf{y}} = \mathbf{A}\mathbf{y}$$

assume solution of the form $\mathbf{y} = \mathbf{y}_0 e^{\boldsymbol{\lambda} t}$

$$\lambda \mathbf{y}_0 e^{\lambda t} = \mathbf{A}\mathbf{y}_0 e^{\lambda t} \text{ holds for any } t$$

$$(\mathbf{A} - \lambda\mathbf{I})\,\mathbf{y}_0 = 0 \Rightarrow |\mathbf{A} - \lambda\mathbf{I}| = 0$$

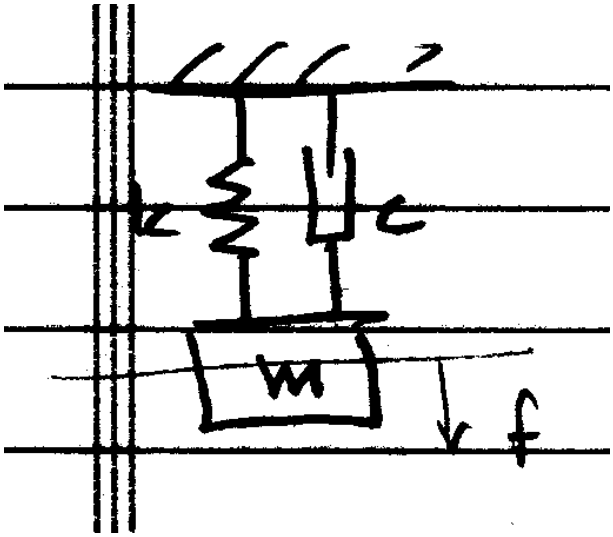

Figure 7.3: Sketch of a mass spring damper system

$$\begin{vmatrix} -\lambda & 1 \\ -\omega_0^2 & -\lambda - 2\beta\omega_0 \end{vmatrix} = \lambda^2 + 2\beta\omega_0\lambda + \omega_0^2 = 0 \tag{7.12}$$

$$\lambda = -\beta_0\omega_0 \pm \sqrt{1-\beta^2}i\omega_0$$

$$\lambda = a \pm bi$$

$\mathrm{Re}(\lambda) \rightarrow$ "Damping" $\mathrm{Im}(\lambda) \rightarrow$ "Oscillation"
So now we have an idea what type of motions are descirbed by the test equation $\dot{\mathbf{y}} = \lambda\mathbf{y}$.

After *one* numerical integration step we have,

$$\mathbf{y} = \hat{\mathbf{y}} + \boldsymbol{\epsilon},$$

with $\hat{\mathbf{y}}$ as the true solution and $\boldsymbol{\epsilon}$ as the truncation error of the method.

$$\dot{\hat{\mathbf{y}}} + \dot{\boldsymbol{\epsilon}} = \lambda\left(\hat{\mathbf{y}} + \boldsymbol{\epsilon}\right)$$

So the true solution $\dot{\hat{\mathbf{y}}} = \lambda\hat{\mathbf{y}}$ leads that the error can propagate:

$$\dot{\boldsymbol{\epsilon}} = \lambda\boldsymbol{\epsilon}$$

$\mathrm{Re}(\lambda) > 0$: Stable only in a fixed interval.
$\mathrm{Re}(\lambda) < 0$: Unconditionally stable.

Back to the test equation and the Method. Let's with the Euler method,

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h\mathbf{f}(t_n, \mathbf{y}_n) \text{ with } \mathbf{f}(t_n, \mathbf{y}_n) = \lambda\mathbf{y}_n$$

$$\begin{aligned} \mathbf{y}_{n+1} &= \mathbf{y}_n + h\lambda\mathbf{y}_n \\ \mathbf{y}_{n+1} &= (1 + h\lambda)\,\mathbf{y}_n \\ \mathbf{y}_{n+1} &= C\,(h\lambda)\,\mathbf{y}_n \end{aligned}$$

where $C(h\lambda)$ is called the amplification factor. For the Euler method we have $C(h\lambda) = (1 + h\lambda)$.
For absolute stability $C(h\lambda) < 1$, where λ can be complex, like in $h\lambda = a + bi$. So for stability we have to look at $|C(h\lambda)| < 1$, which for the Euler method results in $|1 + a + bi| < 1$. Definition stable: local errors do not propagate!

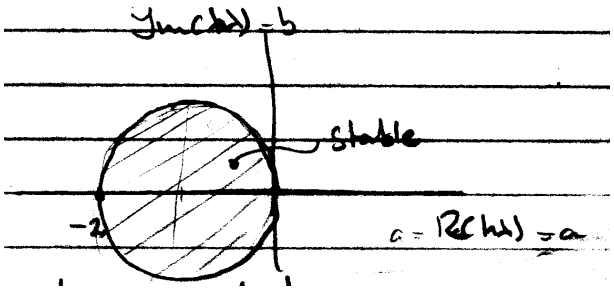

Figure 7.4: Euler stability area in the complex coordinates

Note that $h\lambda = bi$ is not in the stability region. Euler's method is unstable for pure oscillating systems. For example the solution of an undamped pendulum will become infinity over a period of time.

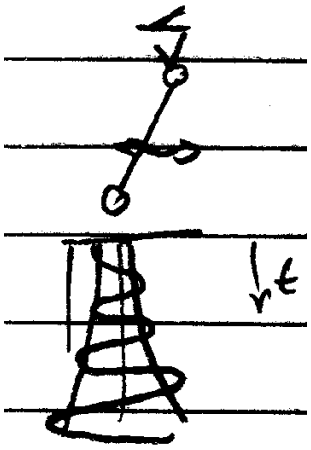

Figure 7.5: pendulum solution using Euler

7.2 Heun step

Let's look at Heun:

$$\begin{aligned}
\mathbf{y}^*_{n+1} = &\quad \mathbf{y}_n + h\lambda\mathbf{y}_n \\
\mathbf{y}_{n+1} = &\quad \mathbf{y}_n + \tfrac{h}{2}\left(\lambda\mathbf{y}_n + \lambda\left(\mathbf{y}_n + h\lambda\mathbf{y}_n\right)\right) \\
\mathbf{y}_{n+1} = &\quad \mathbf{y}_n + \tfrac{h\lambda}{2}\mathbf{y}_n + \tfrac{h\lambda}{2}\mathbf{y}_n + \tfrac{h^2\lambda^2}{2}\mathbf{y}_n \\
\mathbf{y}_{n+1} = &\quad \left(1 + (h\lambda) + \tfrac{1}{2}(h\lambda)^2\right)\mathbf{y}_n
\end{aligned}$$

where $C(h\lambda) = \left(1 + (h\lambda) + \frac{1}{2}(h\lambda)^2\right)$.
$z = h\lambda$ the condition becomes: $\left|1 + z + 1/2z^2\right| < 1$.
From the figure it is shown that Heun is also unstable for pure oscillating

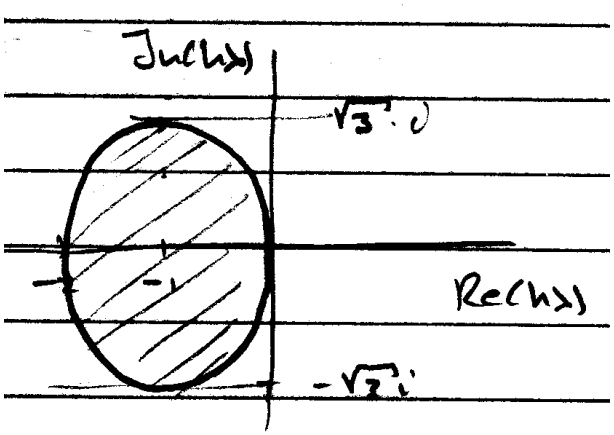

Figure 7.6: Heun stability area in the complex coordinates

systems.

7.3 Runge Kutta 4 step

A very accurate and efficient method is the so-called Runge Kutta method,

$$\begin{aligned}
k_1 = &\quad \mathbf{f}(t_n, \mathbf{y}_n) \\
k_2 = &\quad \mathbf{f}\left(t_n + \tfrac{h}{2}, \mathbf{y}_n + \tfrac{h}{2}k_1\right) \\
k_3 = &\quad \mathbf{f}\left(t_n + \tfrac{h}{2}, \mathbf{y}_n + \tfrac{h}{2}k_2\right) \\
k_4 = &\quad \mathbf{f}(t_n + h, \mathbf{y}_n + hk_3) \\
\mathbf{y}_{n+1} = &\quad \mathbf{y}_n + \tfrac{h}{6}(k_1 + 2k_2 + 2k_3 + k_4)
\end{aligned}$$

The local truncation error is $\epsilon = O\left(h^5\right)$ and the amplification factor for this method is: $C(\lambda h) = 1 + (h\lambda) + 1/2(h\lambda)^2 + 1/6(h\lambda)^3 + 1/24(h\lambda)^4$.
This method is stable for pure oscillating systems.
For stability:
$h\lambda < 2.8$ and with $\lambda = \omega_0$

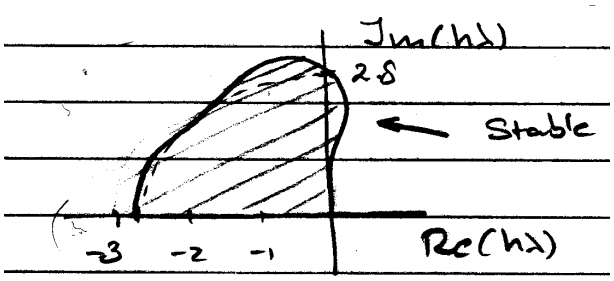

Figure 7.7: Runge Kutta stability area in the complex coordinates

$h < 2.8/\omega_0$ or with $T_0 = 2\pi/\omega_0 \rightarrow h < (2.8/2\pi)T_0 \rightarrow h < 0.4T_0$
These were all 1-step methods for systems first order differential equations.

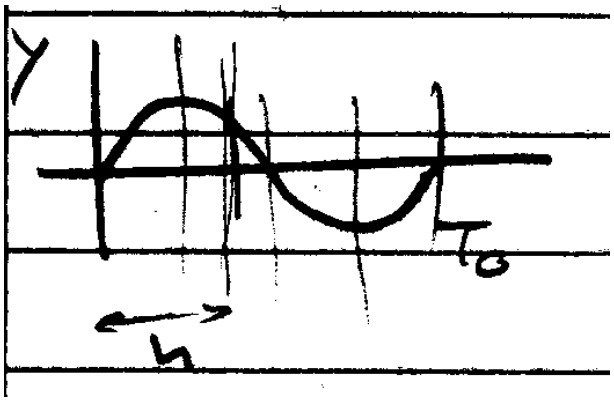

Figure 7.8: Runge Kutta maximum time step to guarantee stability

7.4 A method for second order differential equations

Now if you look at our system of differential equations half of them are pretty simple.

$$\begin{aligned} \dot{\mathbf{q}} = &\quad \mathbf{z} \\ \dot{\mathbf{z}} = &\quad \mathbf{f}\left(t, \mathbf{q}, \mathbf{z}\right) \end{aligned}$$

We can write this explicit in a very simple one (Euler 2)

$$\begin{aligned} k_1 = &\quad \mathbf{f}\left(t_n + \tfrac{h}{2}, \mathbf{q}_n + \tfrac{h}{2}\mathbf{z}_n, z_n\right) \\ \mathbf{q}_{n+1} = &\quad \mathbf{q}_n + h\mathbf{z}_n + \tfrac{1}{2}h^2 k_1 \\ \mathbf{z}_{n+1} = &\quad \mathbf{z}_n + hk_1 \end{aligned}$$

with $t_n + h/2$ is the midpoint and $\mathbf{z}_n$ not interesting, just pick $\mathbf{z}_n$. Note that only 1 function (k_1) calls to the differential equation at the midpoint.

The local truncation error is:

$$\boldsymbol{\epsilon} = \mathbf{O}\left(h^2\right)$$

but if $\mathbf{f}$ is weak in $\mathbf{z}$ (the velocities)

$$\boldsymbol{\epsilon} = \mathbf{O}\left(h^3\right)$$

To investigate the stability we use as a test equation the simplest second order equation that of a linear damped oscillatory system.

$$\ddot{q} + 2\beta\omega_0\dot{q} + \omega_0^2 q = 0$$

and with $\dot{q} = z$:

$$\begin{pmatrix} \dot{q} \\ \dot{z} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -\omega_0^2 & -2\beta\omega_0 \end{pmatrix} \begin{pmatrix} q \\ z \end{pmatrix} \tag{7.13}$$

which can be seen as a first order differential equation:$\dot{\mathbf{y}} = \lambda\mathbf{y}$.

$$\lambda_{1,2} = -\beta\omega_0 \pm \sqrt{1-\beta^2}\omega_0 i$$

The stability comes out as:
Stable for pure oscillating systems.

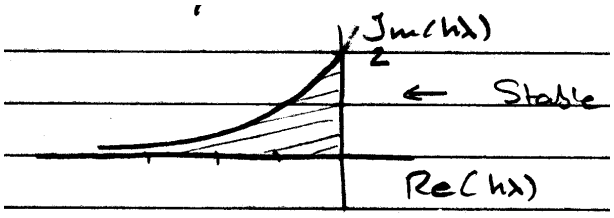

Figure 7.9: Runge Kutta stability detail

An "arcade" scheme, suitable for the entertainment industry, moderate accuracy but fast only 1 function call per step. Note that this function call is at the time increment $t_n + h/2$. So in the end we know the position q and the velocity z at the discrete points but the acceleration $\ddot{q}$ and all the forces etc. are only known at the midpoints.

7.5 Global Error Estimate Revisited

To get more accurate results you just decrease the step size h, but $\cdots$ we only looked at local truncation errors of the type $\epsilon = O\left(h^{p+1}\right)$ or $E = O\left(h^p\right)$. Since we work with finite word lengths in the computer to represent our numbers we also have round-off errors which re of the type $\epsilon = C_2 \leftarrow$ a very small number but since we take the $n = T/h$ steps the global or accumulated round-off error is:

$$E = \frac{T}{h}C_2 = O\left(\frac{1}{h}\right)$$

In refining our step size we would hit this. So the fatal global error is like:

$$E = C_1 h^p + \frac{C_2}{h}$$

For example Heun's method we have $E = C_1 h^2 + C_2/h$ which is shown in Figure (7.10), or better on a log-log scale in Figure (7.11).

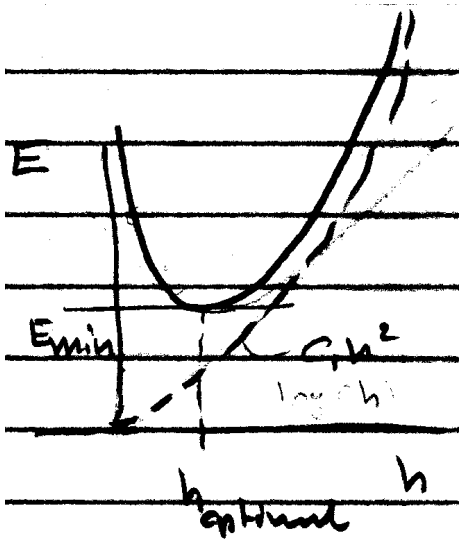

Figure 7.10: global error versus step size on linear scale

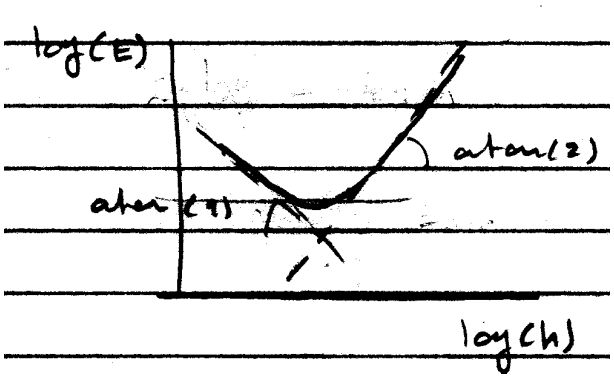

Figure 7.11: global error versus step size on logarithmic scale

7.6 Error estimate in practise

The global error estimate has the term:

$$y = \hat{y} + C_1 h^p + C_2 \frac{1}{h}$$

Now we expect to be either in the second term or in the third term to be dominant for the error estimate.
Dominant truncation: $y = \hat{y} + C_1 h^p$
Dominant round-off: $y = \hat{y} + C_2/h$
Take a number of successive steps with the step sizes $h = T/2^n$ and calculate T =timeperiod (usually start with $n = 6$ or 8) the differences at two successive solutions:

$$D_n = |y_n - y_{n-1}|$$

where y_n is $\int_0^T$ at step size $h = T/2^n$ and plot this on a log-log scale.

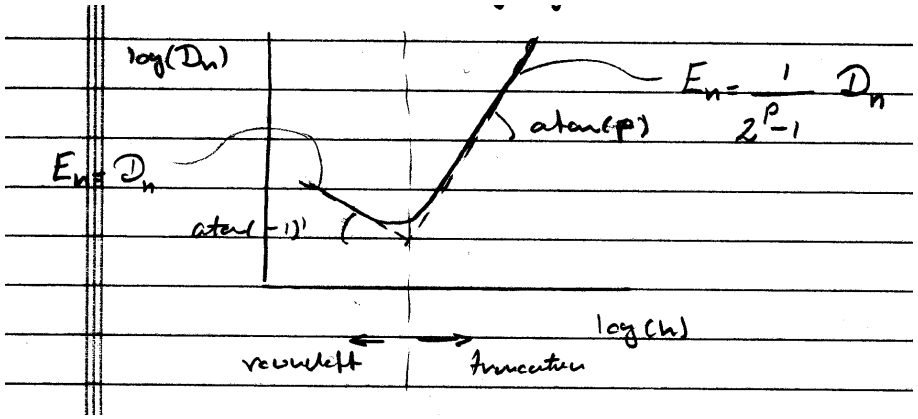

Figure 7.12: global error of the truncation and round-off effect versus step size on logarithmic scale

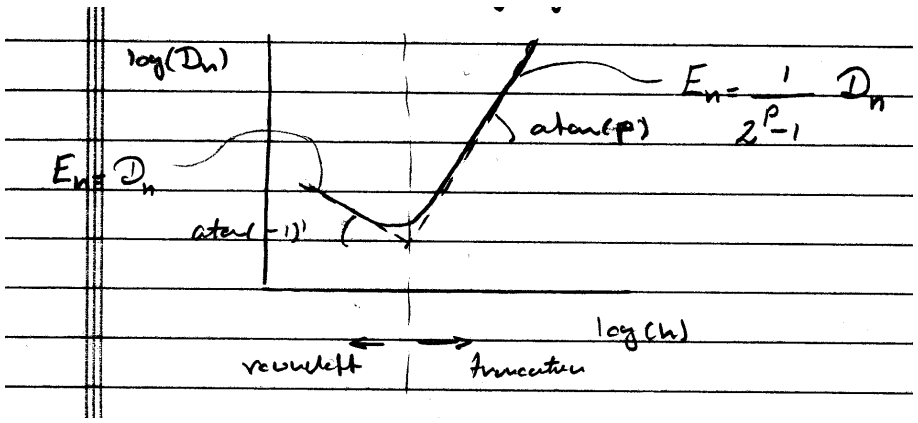

Figure 7.13: global error of the truncation and round-off effect versus step size on logarithmic scale

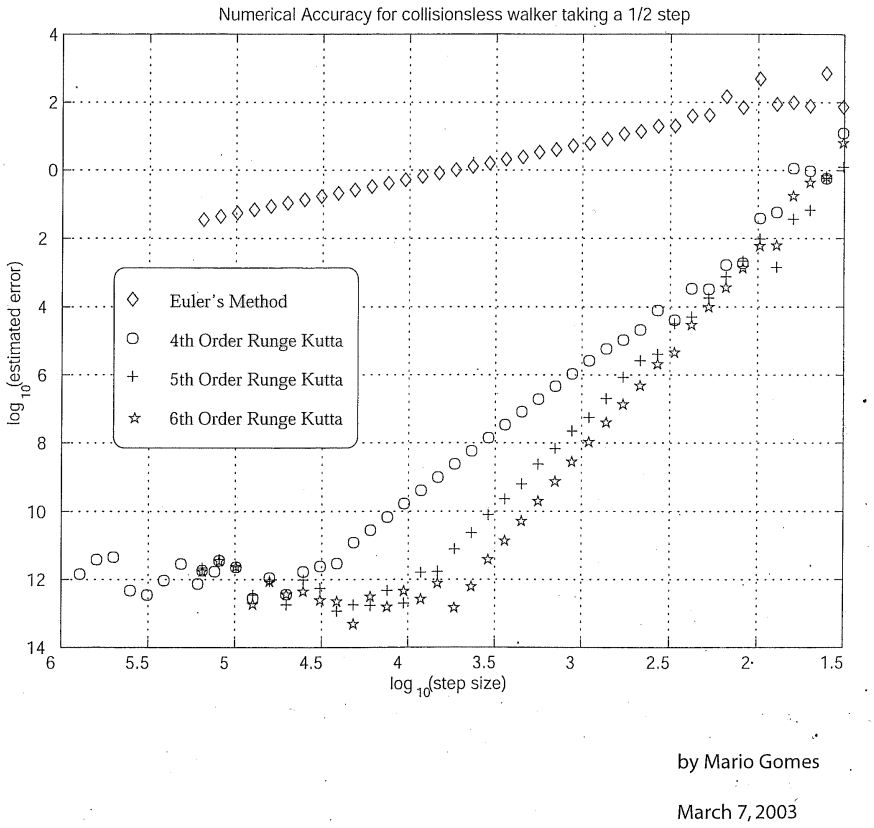

Figure 7.14: the truncation and round-off error for different integration methods versus step size on logarithmic scale

7.7 Implicit Methods

These were all explicit methods, recall Forward Euler $\mathbf{y}_{n+1} = \mathbf{y}_n + h\mathbf{f}(t_n, \mathbf{y}_n)$ in contrast to the Backward Euler $\mathbf{y}_{n+1} = \mathbf{y}_n + h\mathbf{f}(t_{n+1}, \mathbf{y}_{n+1})$.
We call this method implicit, in order to know $\mathbf{y}_{n+1}$ we need to evaluate $\mathbf{f}(t_{n+1}, \mathbf{y}_{n+1})$, so in general we need to solve these usually non-linear equations.

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

Why better? Accuracy and stability.

Accuracy: The local truncation error $\boldsymbol{\epsilon} = \mathbf{O}(h^2)$ is the same as for the forward Euler.

Stability: with the same test equation:

$$\dot{\mathbf{y}} = \lambda \mathbf{y}$$

We came up with:

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h\lambda \mathbf{y}_{n+1}$$

And for the error:

$$\begin{aligned} \boldsymbol{\epsilon}_{n+1} = &\quad \boldsymbol{\epsilon}_n + h\lambda \boldsymbol{\epsilon}_{n+1} \\ \boldsymbol{\epsilon}_{n+1} = &\quad \tfrac{1}{1-h\lambda}\boldsymbol{\epsilon}_n \end{aligned}$$

The amplification factor $\mathbf{C}(h\lambda) = 1/(1-h\lambda)$. Stability is expressed by

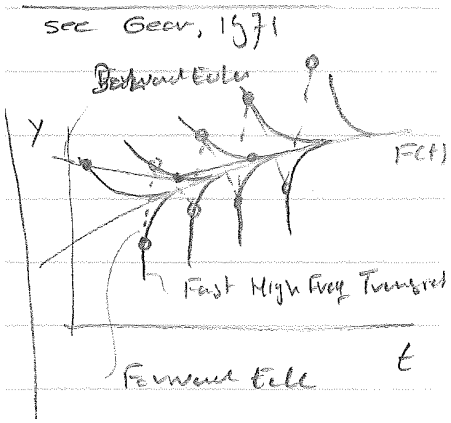

Figure 7.15: Forward versus Backward Euler following Gear

$|\mathbf{C}(h\lambda)| < 1$.
And with λ complex we get:
Even high frequency signals are not blown up! There are higher order implicit solutions which show the same stability behaviour.

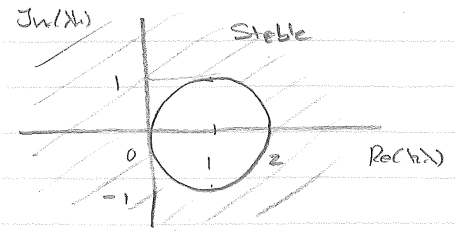

Figure 7.16: Backward Euler stability area

7.8 Linear Multi-step Methods

Even earlier than Runge-Kutta they were devised by J.C. Adams to solve a problem of F. Bashfort. The origin of the method has been dated back to 1855 when F. Bashfort made an application to the Royal Society for assistance from the Government grant. There he wrote:" $\cdots$ but I am indebted to Mr. Adams for a method of rewriting the differential equation $\cdots$ which gives the theoretical form of the drop with an accuracy exceeding that of the most refined measurements". Eventually the methods were published by Bashfort in 1883. What happened in 1855 in the United states of America? In 1855 Whitman published at his own expense a volume of 12 poems, Leaves of Grass. Smith & Wesson invents the revolver. It was 6 years before the civil war would started.

$$\dot{\mathbf{y}} = \mathbf{f}\left(t, \mathbf{y}\right)$$

This is the idea:

$$\mathbf{y}_1 = \mathbf{y}_0 + \int_{t_0}^{t_1} \mathbf{f}\left(t, \mathbf{y}\right) \mathrm{d}t$$

Describe $\mathbf{f}$ in a Langrange polynomial as a function of the past and $\int$ one step

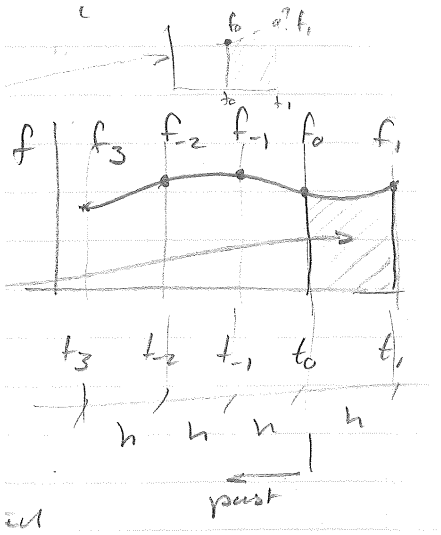

Figure 7.17: Linear multistep method principle of Bashfort

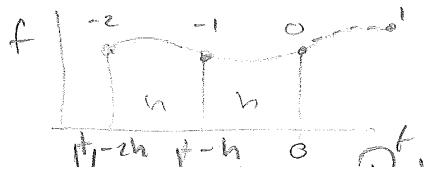

Figure 7.18: Fictive interpolation future value using Lagrange polynomial

in the fictive with the interpolated function value. For instance the Lagrange polynomial for 3 $\mathbf{f}$'s.

$$\mathbf{p}(t) = \mathbf{f}_{-2}\frac{(t+h)\,t}{(-h)\,(-2h)} + \mathbf{f}_{-1}\frac{(t+2h)\,t}{(h)\,(-h)} + \mathbf{f}_0\frac{(t+2h)\,(t+h)}{(2h)\,(h)}$$

Approximate
$\int_{t_0}^{t_1}\mathbf{f}(t,\mathbf{y})\,\mathrm{d}t$ by $\int_0^h \mathbf{p}(t)\,\mathrm{d}t$ which is $h\,(5/12\mathbf{f}_{-2} - 16/12\mathbf{f}_{-1} + 23/12\mathbf{f}_0)$. Thus we get $\mathbf{y}_{n+1} = \mathbf{y}_n + h/12\,(23\mathbf{f}_n - 16\mathbf{f}_{n-1} + 5\mathbf{f}_{n-2})$ which is the explicit Adams-Bashfort method. Now you could make an implicit scheme by adding $\mathbf{f}_{n+1}$ in the polynomial resulting in:

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \frac{h}{12}\,(5\mathbf{f}_{n+1} + 8\mathbf{f}_n - \mathbf{f}_{n-1})$$

The implicit Adams-Moulton method.
We do not like Implicit schemes because of the necessary iterations.

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \frac{h}{24}\,(9\mathbf{f}_{n+1} + 19\mathbf{f}_n - 5\mathbf{f}_{n-1} + \mathbf{f}_{n-2})$$

Now a very popular scheme is the following PECE (Shampine 1975)

Predict with $A-B$ order k	$\mathbf{y}_{n+1}^p$
Evaluate the differential equation	$\mathbf{f}_{n+1}^p\left(t_{n+1}, \mathbf{y}_{n+1}^p\right)$
Correct with $A-M$ order $k+1$	$\mathbf{y}_{n+1}$
Evaluate the differential equations	$\mathbf{f}_{n+1}\left(t_{n+1}, \mathbf{y}_{n+1}\right)$

This scheme uses only 2 future evaluations/step. This is the ode113 solver of Matlab!
Pro's:

- Only 2 Function Evaluations/step independent of the order
- Interpolated values of $\mathbf{y}$ for "free"

Con's:

- Startup is difficult
- Order $k > 0$ stability decreases

Change of step size h and order k?
Compare this with a carrace in reverse.

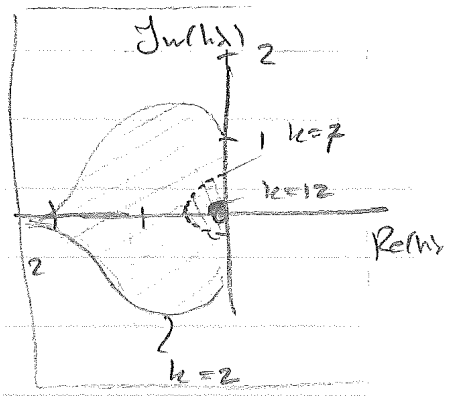

Figure 7.19: the stability area of the multi step integration method for different orders

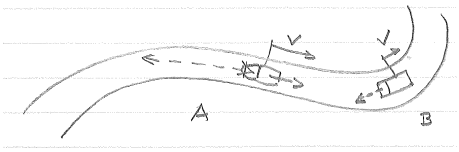

Figure 7.20: sketch of a "reverse" carrace

A: A smooth ride $\mathbf{v} > \rightarrow h >$
look far backward $\rightarrow k >$
B: A sudden break $\mathbf{v} < \rightarrow h <$
forget the past $\rightarrow k <$

Finally
There are BDF-methods Backward Differentiating made popular by Gear 1971 subscribing DIFSUB. Again we build a Lagrange polynomial but now on $\mathbf{y}_{-3}, \mathbf{y}_{-2}, \mathbf{y}_{-1}, \mathbf{y}_0$ and predict $\mathbf{y}_1$. But where is the differential equation? Differentiate the polynomial and set the result being $\mathbf{f}$. For instance for a BDF with constant step size h and order 2 has the form of:

$$\frac{3}{2}\mathbf{y}_1 - 2\mathbf{y}_0 + \frac{1}{2}\mathbf{f}_{-1} = h\mathbf{f}\left(t_1\mathbf{y}_n\right)$$

$$\mathbf{y}_1 = -\frac{1}{3}\mathbf{y}_{-1} + \frac{4}{3}\mathbf{y}_0 + \frac{2}{3}h\mathbf{f}\left(t_1\mathbf{y}_n\right)$$

Again, thus is an implicit scheme, and think of the startup problem. The methods are unconditionally stable up to order 6. This is ode15s with option BDF.

Most of you will be using the Matlab ode45 or ode23. There is a nice paper by L.F. Shampine and M.W. Reichelt, "The Matlab ODE Suite" SIAM J. Sci. Compt., 18(1): 1-22, 1997. I will put the pdf-file on the website. (Note some options have changed in time!)

Before we start let's introduce Auckland New Zealand 1987 the J.C. Butcher for a s-stage Runge-Kutta method:

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h\Sigma_{i=1}^{s} b_i k_i$$

with: $k_i = \mathbf{f}\left(t_n + c_i h, \mathbf{y}_n + h\Sigma_{j=1}^{s} a_{ij} k_j\right)$.
The method is characterised by s, the stages and the coefficients a_{ij}, b_i and c_i. These coefficients are usually written in the following scheme: Butcher arrays.

$$\begin{array}{c|c} c_1 & a_{11}\cdots a_{1s} \\ \vdots & \vdots \\ c_s & a_{s1}\cdots a_{ss} \\ \hline & b_1 \cdots b_s \end{array} \quad \text{or} \quad \begin{array}{c|c} \mathbf{c} & \mathbf{A} \\ \hline & \mathbf{b}^T \end{array}$$

So RK4 looks like

$$\begin{array}{ll} \mathbf{k}_1 & = \mathbf{f}\left(t_n, \mathbf{y}_n\right) \\ \mathbf{k}_2 & = \mathbf{f}\left(t_n + \frac{h}{2}, \mathbf{y}_n + \frac{h}{2}\mathbf{k}_1\right) \\ \mathbf{k}_3 & = \mathbf{f}\left(t_n + \frac{h}{2}, \mathbf{y}_n + \frac{h}{2}\mathbf{k}_2\right) \\ \mathbf{k}_4 & = \mathbf{f}\left(t_n + h, \mathbf{y}_n + h\mathbf{k}_3\right) \\ \mathbf{y}_{n+1} & = \mathbf{y}_n + \frac{h}{6}\left(\mathbf{k}_1 + 2\mathbf{k}_2 + 2\mathbf{k}_3 + \mathbf{k}_4\right) \end{array} \qquad \begin{array}{c|cccc} 0 & 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 \\ \hline & \frac{1}{6} & \frac{1}{3} & \frac{1}{3} & \frac{1}{6} \end{array}$$

Now if $a_{ij} = 0 j \geq i \rightarrow$ Explicit Method
and $a_{ij} = 0 j > i \rightarrow$ Diagonal Implicit
else $\rightarrow$ Implicit sensostrito
(The number c_i have to fulfill by: $c_i = \Sigma_{j=1}^{s} a_{ij}$ in order the integrated time dependent system which we make arbencunicus)

Methods with one stage, $s = 1$

$$\begin{array}{c|c} 0 & 0 \\ \hline & 1 \end{array} \text{ Forward Euler} \qquad \begin{array}{c|c} 1 & 1 \\ \hline & 1 \end{array} \text{ Backward Euler}$$

Explicit Methods with second stages, $s = 2$

$$\begin{array}{c|cc} 0 & 0 & 0 \\ 1 & 1 & 0 \\ \hline & \frac{1}{2} & \frac{1}{2} \end{array} \text{ Heun} \left(\text{but any } \begin{array}{c|cc} 0 & 0 & 0 \\ \frac{1}{2\alpha} & \frac{1}{2\alpha} & 0 \\ \hline & 1-\alpha & \alpha \end{array} \boldsymbol{\epsilon} = \mathbf{O}\left(h^3\right) \right)$$

Basis of the RKs-stage methods
ODE $\mathbf{y}' = \mathbf{f}\left(\mathbf{y}\right)$
Taylor expansion solution is:

$$\mathbf{y}\left(t+h\right) = \mathbf{y}\left(t\right) + h\mathbf{y}' + \frac{h^2}{2}\mathbf{y}'' + \cdots + \frac{h^p}{p!}\mathbf{y}^p + \mathbf{O}\left(h^{p+1}\right)$$

Runge-Kutta 2 stage method explicit (hope for $\boldsymbol{\epsilon} = \mathbf{O}\left(h^3\right)$)

$$\begin{array}{c|cc} 0 & 0 & 0 \\ b_2 & a_{21} & 0 \\ \hline & c_1 & c_2 \end{array} \rightarrow \begin{array}{c|cc} 0 & 0 & 0 \\ a & a & 0 \\ \hline & c_1 & c_2 \end{array}$$

$$\begin{array}{ll} \mathbf{k}_1 & = \mathbf{f}\left(\mathbf{y}_n\right) \\ \mathbf{k}_2 & = \mathbf{f}\left(\mathbf{y}_n + ah\mathbf{k}_1\right) \\ \mathbf{y}_{n+1} & = \mathbf{y}_n + h\left(c_1\mathbf{k}_1 + c_2\mathbf{k}_2\right) \end{array} \qquad \left(\begin{array}{c|cc} 0 & 0 & 0 \\ \frac{1}{2\alpha} & \frac{1}{2\alpha} & 0 \\ \hline & 1-\alpha & \alpha \end{array} \right)$$

Taylor expansion:

$$\mathbf{k}_2 = \mathbf{f}(\mathbf{y}_n) + ah\mathbf{k}_1\mathbf{f}'(\mathbf{y}_n) + \frac{(ah\mathbf{k}_1)^2}{2}\mathbf{f}''(\mathbf{y}_n) + \cdots$$

$$\begin{aligned}\mathbf{y}_{n+1} &= \mathbf{y}_n + hc_1\mathbf{f}(\mathbf{y}_n) + hc_2\mathbf{f}(\mathbf{y}_n) + hc_2ah\mathbf{k}_1\mathbf{f}'(\mathbf{y}_n) + \mathbf{O}\left(h^3\right)\\ \mathbf{y}_{n+1} &= \mathbf{y}_n + h(c_1+c_2)\mathbf{f}(\mathbf{y}_n) + h^2c_2a\mathbf{k}_1\mathbf{f}'(\mathbf{y}_n) + \mathbf{O}\left(h^3\right)\end{aligned}$$

Taylor expansion solution:

$$\mathbf{y}_{n+1} = \mathbf{y}_n + h\mathbf{y}'_n + \frac{h^2}{2}\mathbf{y}''_n + \mathbf{O}\left(h^3\right)$$

$$\mathbf{y}'' = \frac{\partial \mathbf{f}}{\partial \mathbf{y}}\frac{\partial \mathbf{y}}{\partial t} = \mathbf{f}'\mathbf{y}' = \mathbf{f}'\mathbf{f}$$

$$\left.\begin{array}{l} h(c_1+c_2) = h \\ h^2c_2a = \frac{1}{2}h^2 \end{array}\right\} \begin{array}{l} c_1+c_2 = 1 \\ c_2a = \frac{1}{2}\end{array}$$

2 equations but 3 constants (c_1, c_2, a) so 1 equation missing. Is coming from:

$$\begin{array}{c|cc} 0 & 0 & 0 \\ \frac{1}{2\alpha} & \frac{1}{2\alpha} & 0 \\ \hline & 1-\alpha & \alpha \end{array}$$

Now lets try and connect this to ODE23 (print font 10 batch)

Explicit 3-stage methods

$$\begin{array}{ll|c|ccc} b_1+b_2+b_3 = 1 & 0 & 0 & 0 & 0 \\ b_2c_2+b_3c_3 = \frac{1}{2} & x & x & 0 & 0 \\ b_2c_2^2+b_3c_3^2 = \frac{1}{3} & x & x & x & 0 \\ b_3a_{32}a_{21} = \frac{1}{6} & & x & x & x \end{array} \boldsymbol{\epsilon} = \mathbf{O}\left(h^4\right)$$

6 coefficients and 4 equations $\rightarrow$ 2 free parameters a 3^{rd} order method. Max # coefficients = 0.

$$\begin{array}{c|ccc} 0 & 0 & 0 & 0 \\ \frac{1}{3} & \frac{1}{3} & 0 & 0 \\ \frac{2}{3} & 0 & \frac{2}{3} & 0 \\ \hline & \frac{1}{4} & 0 & \frac{3}{4} \end{array} \quad \text{RK3} \quad \begin{array}{c|ccc} 0 & 0 & 0 & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & 0 \\ \frac{3}{4} & 0 & \frac{3}{4} & 0 \\ \hline & \frac{2}{9} & \frac{1}{3} & \frac{4}{9} \end{array} \quad \text{"ODE23"}$$

But when you read closely you see the following Butcher arrays.

$$\begin{array}{c|c|c} A = \begin{pmatrix} 0 \\ \frac{1}{2} \\ \frac{3}{4} \\ 1 \end{pmatrix} & \begin{pmatrix} 0 & 0 & 0 & 0 \\ \frac{1}{2} & 0 & 0 & 0 \\ 0 & \frac{3}{4} & 0 & 0 \\ \frac{2}{9} & \frac{1}{3} & \frac{4}{9} & 0 \end{pmatrix} = B^T & h\mathbf{B} = h.*\mathbf{B} \\ \hline & \begin{pmatrix} -\frac{5}{72} & \frac{1}{12} & \frac{1}{9} & -\frac{1}{8} \end{pmatrix} = E^T & \end{array}$$

$$\mathbf{y}_{n+1} = \mathbf{y}_n + \mathbf{f} * h\mathbf{B}(:,3)$$

$\mathbf{f}(:,\mathbf{y}) = \mathbf{f}(\mathbf{y}_{n+1})$ this $\mathbf{f}_y$ is used for local error estimate.

```
function varargout = ode(ode,tspan,y0,options,varargin)
% ODE23 Solve non-stiff differential equations, low order method.
% [T,Y] = ODE23(ODEFUN,TSPAN,Y0) with TSPAN = [T0 TFINAL] integrates the
% ... cut ...
% Initialize method parameters.
pow = 1/3;
A = [1/2; 3/4; 1];
B = [
 1/2   0      2/9
 0     3/4    1/3
 0     0      4/9
 0     0      0];
E = [-5/72; 1/12; 1/9; -1/8];
f = zeros(neq,4);
hmin = 16*eps*abs(t);
% ... cut ...
% LOOP FOR ADVANCING ONE STEP.
nofailed = true;       % no failed attemps
while true
hB = h * B;
f(:,2) = feval(ode,t+h*A(1),y+f*hB(:,1),args:);
f(:,3) = feval(ode,t+h*A(2),y+f*hB(:,2),args:);
tnew = t + h*A(3);
if done
tnew = tfinal;    % Hit end point exactly.
end
h = tnew - t;    % Purify h.

ynew = y + f*hB(:,3);
f(:,4) = feval(ode,tnew,ynew,args:);
stats.nfevals = stats.nfevals + 3;

% Estimate the error.
if normcontrol
normynew = norm(ynew);
err = absh * (norm(f * E) / max(max(normy,normynew),threshold));
else
err = absh * (norm(f * E) ./ max(max(abs(y),abs(ynew)),threshold),inf);
end

% Accept the solution only if the weighted error is no more than the tolerance rtol. Estimate an h that will yield an error of rtol on the next step or the next try at taking this step, as the case may be, and use 0.8 of this value to avoid failures.
if err > rtol       % Failed step
% ... cut ...

function yinterp = ntrp23(tinterp,t,y,tnew,ynew,h,f)
%NTRP23 Interpolation helper function for ODE23.
```

BI = [
 1 −4/3 5/9
 0 1 −2/3
 0 4/3 −8/9
 0 −1 1];
s = ((tinterp - t) / h)'; % may be a row vector
yinterp = y(:,ones(length(tinterp),1)) + f*(h*BI)*cumprod(s(ones(3,1),:));

RK3 ("ODE23")

$$\begin{array}{ll} \mathbf{k}_1 & = \mathbf{f}\left(t_n, \mathbf{y}_n\right) \\ \mathbf{k}_2 & = \mathbf{f}\left(t_n + \frac{h}{2}, \mathbf{y}_n + \frac{h}{2}\mathbf{k}_1\right) \\ \mathbf{k}_3 & = \mathbf{f}\left(t_n + \frac{3h}{4}, \mathbf{y}_n + \frac{3h}{4}\mathbf{k}_2\right) \\ \mathbf{y}_{n+1} & = \mathbf{y}_n + \frac{h}{9}\left(2\mathbf{k}_1 + \mathbf{k}_2 + 4\mathbf{k}_3\right) \end{array}$$

$\boldsymbol{\epsilon} = \text{abs}h * \text{norm}\left(\mathbf{f} * \mathbf{E}\right)$ abs: back iterative norm: multi distinctive
Local truncation error estimate:

$$\boldsymbol{\epsilon} = h\left(-\frac{5}{72}\mathbf{k}_1 + \frac{1}{12}\mathbf{k}_2 + \frac{1}{9}\mathbf{k}_3 - \frac{1}{8}\mathbf{k}_4\right)$$

If the step is accepted then: ...
f(:,1) = f(:,4) FSAL (First Same As Last)
So apparently 4 stage but actually 3-stage method.

Finally, for dense output or event interpolation we want to interpolate accurately between $\mathbf{y}_n$ and $\mathbf{y}_{n+1}$ without extra function evaluations to $\mathbf{f}$.

$$\begin{array}{rl} \mathbf{y}_{n+\alpha} = \mathbf{y}_n + & h\mathbf{k}_1\left(\alpha - \frac{4}{3}\alpha^2 + \frac{5}{9}\alpha^3\right) + \\ & h\mathbf{k}_2\left(\alpha^2 - \frac{2}{3}\alpha^3\right) + \\ & h\mathbf{k}_3\left(\frac{4}{3}\alpha^2 - \frac{8}{9}\alpha^3\right) + \\ & h\mathbf{k}_4\left(-\alpha^2 + \alpha^3\right)\left(+\mathbf{O}\left(h^4\right)??\right) \end{array}$$

So we have two extra things here:

- Local truncation error estimate and step size adaption!
- Accurate interpolation results.

Now look for yourself at ode45

function varargout = ode45(ode,tspan,y0,options,varargin)
%ODE45 Solve non-stiff differential equations, medium order method.
% [T,Y] = ODE45(ODEFUN,TSPAN,Y0) with TSPAN = [T0 TFINAL] integrates the
% ... cut ...
% Initialize method parameters.
pow = 1/5;
A = [1/5; 3/10; 4/5; 8/9; 1; 1];
B = [

```
 1/5   3/40   44/45     19372/6561     9017/3168      35/384
 0     9/40   −56/15    −25360/2187    −355/33        0
 0     0      32/9      64448/6561     46732/5247     500/1113
 0     0      0         −212/729       49/176         125/192
 0     0      0         0              −5103/18656    −2187/6784
 0     0      0         0              0              11/84
 0     0      0         0              0              0];
E = [71/57600; 0; -71/16695; 71/1920; -17253/339200; 22/525; -1/40];
f = zeros(neq,7);
hmin = 16*eps*abs(t);
% ... cut ...
% LOOP FOR ADVANCING ONE STEP.
nofailed = true;        % no failed attemps
while true
hA = h * A;
hB = h * B;
f(:,2) = feval(ode,t+hA(1),y+f*hB(:,1),args:);
f(:,3) = feval(ode,t+hA(2),y+f*hB(:,2),args:);
f(:,4) = feval(ode,t+hA(3),y+f*hB(:,3),args:);
f(:,5) = feval(ode,t+hA(4),y+f*hB(:,4),args:);
f(:,6) = feval(ode,t+hA(5),y+f*hB(:,5),args:);

tnew = t + hA(6);
if done
tnew = tfinal;        % Hit end point exactly.
end
h = tnew - t;        % Purify h.

ynew = y + f*hB(:,6);
f(:,7) = feval(ode,tnew,ynew,args:);
stats.nfevals = stats.nfevals + 6;

% Estimate the error.
if normcontrol
normynew = norm(ynew)
err = absh * (norm(f * E) / max(max(normy,normynew),threshold));
else
err = absh * norm((f * E) ./ max(max(abs(y),abs(ynew)),threshold),inf); end

% Accept the solution only if the weighted error is no more than the
% tolerance rtol. Estimate an h that will yield an error of rtol on
% the next step or the next try at taking this step, as the case may be,
% and use 0.8 of this value to avoid failures.
if err > rtol        % Failed step
% ... cut ...

function yinterp = ntrp45(tinterp,t,y,tnew,ynew,h,f)
%NTRP45 Interpolation helper function for ODE45.
BI = [
```

```
 1   −183/64     37/12          −145/128
 0   0           0              0
 0   1500/371    −1000/159      −1000/371
 0   −125/32     125/12         −375/64
 0   9477/3392   −729/106       25515/6784
 0   −11/7       11/3           −55/28
 0   3/2         −4             5/2];
s = ((tinterp -t) / h)';       % may be a row vector
yinterp = y(:,ones(length(tinterp),1)) + f*(h*BI)*cumprod(s(ones(4,1),1:));
```

Bibliography

[1] P. Henrici. *Discrete variable methods in ordinary differential equations*. Wiley, New York, 1962.

[2] Charles William Gear. *Numerical initial value problems in ordinary differential equations*. Prentice-Hall, Englewood Cliffs, N.J., 1971.

[3] L. F. Shampine and M. K. Gordon. *Computer solution of ordinary differential equations: the initial value problem*. W. H. Freeman, San Francisco, 1975.

[4] John Charles Butcher. *The numerical analysis of ordinary differential equations: Runge-Kutta and general linear methods*. J. Wiley, Chichester, New York, 1987.

[5] John H. Hubbard and Beverly H. West. *Differential Equations: A Dynamical System Approach, Part 1, Ordinary Differential Equations*. Number 5 in Texts in Applied Mathematics. Springer, New York, 1991.

[6] E. Hairer, S. P. Nørsett, and G. Wanner. *Solving Ordinary Differential Equations I: Nonstiff Problems*. Number 8 in Springer Series in Computational Mathematics. Springer-Verlag, Berlin Heidelberg, second revised edition, 1993.

[7] Lawrence F. Shampine. *Numerical solution of ordinary differential equations*. Chapman & Hall, New York, 1994.

[8] Edda Eich-Soellner and Claus Führer. *Numerical Methods in Multibody Dynamics*. European Consortium for Mathematics in Industry. B.G.Teubner, Stuttgart, 1998.

[9] Reinhold von Schwerin. *Multibody System Simulation: Numerical Methods, Algorithms, and Software*. Number 7 in Lecture Notes in Computational Science and Engineering. Springer-Verlag, 1999.

[10] Cleve B. Moler. *Numerical Computing with Matlab*. SIAM, 2004. (Free at: http://www.mathworks.com/moler/chapters.html).