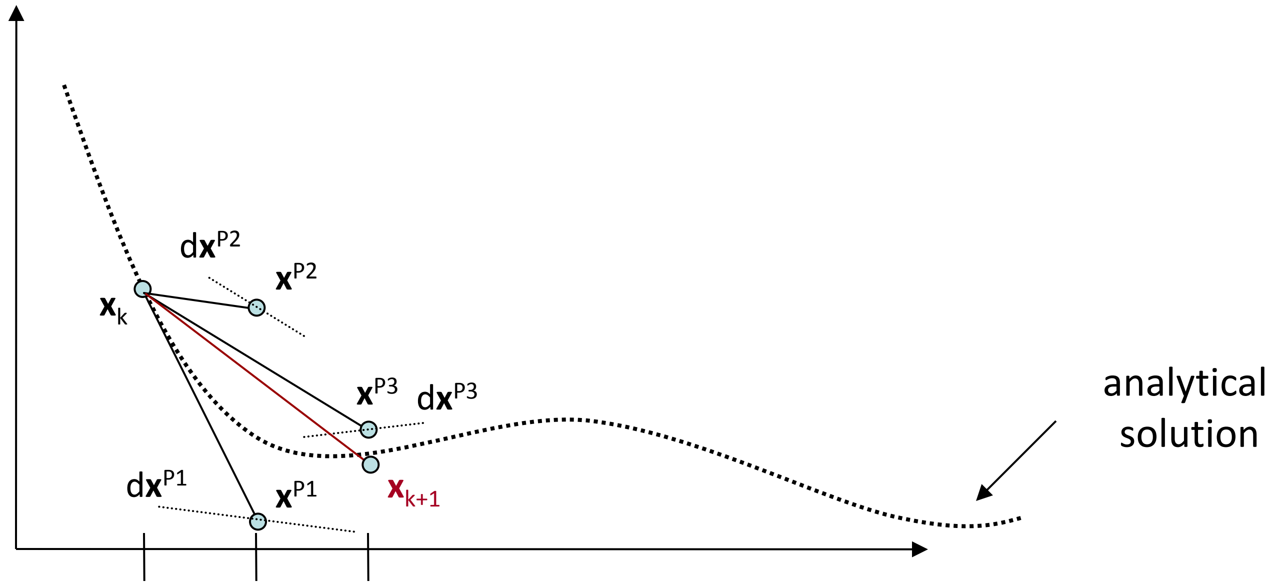
Virtual Physics
Equation-Based Modeling

TUM, December 16, 2014

Higher-Order Methods



Dr. Dirk Zimmer

German Aerospace Center (DLR), Robotics and Mechatronics Centre

Heun

Let us look at an extension of Forward Euler:

- We start by making a prediction step using forward Euler:

prediction: $\mathrm{d}\mathbf{x}_k = f(\mathbf{x}_k, t_k)$ (We will now omit /dt in dx/dt)

$\mathbf{x}^P{}_{k+1} = \mathbf{x}_k + h \cdot \mathrm{d}\mathbf{x}_k$

- Using $\mathbf{x}^P$, we make a second guess of $\mathrm{d}\mathbf{x}/\mathrm{d}t$ and finally make a correction step, combining both guesses.

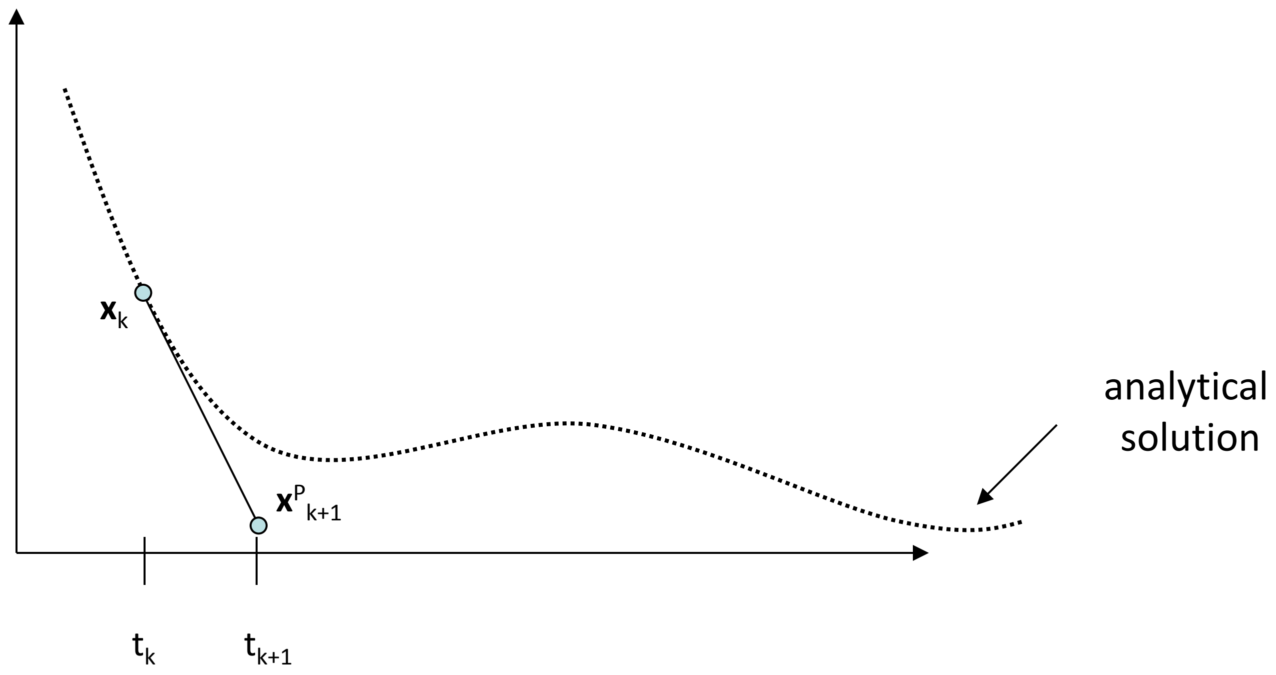
correction: $\mathrm{d}\mathbf{x}^P{}_{k+1} = f(\mathbf{x}^P{}_{k+1}, t_k + h)$

$\mathbf{x}^C{}_{k+1} = \mathbf{x}_k + h/2 \cdot (\mathrm{d}\mathbf{x}_k + \mathrm{d}\mathbf{x}^P{}_{k+1})$

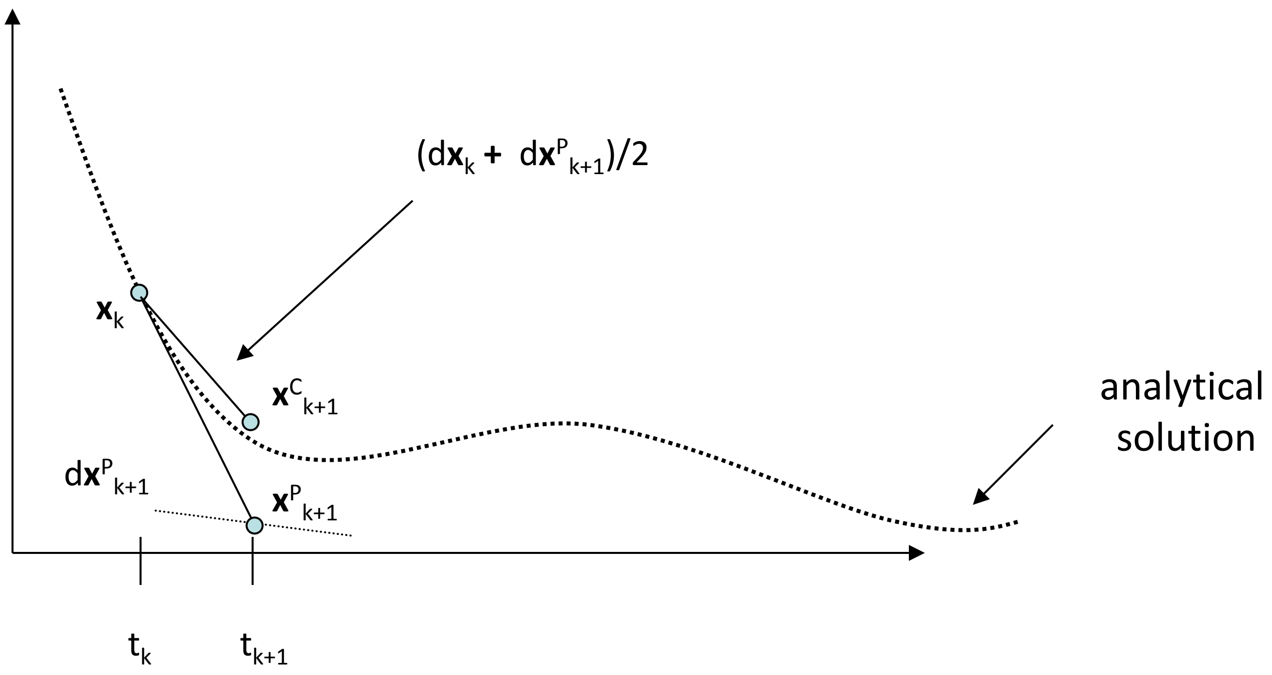
- This method is known as Heun‘s method.

- It involves two evaluations of f() but it is second order accurate.

Heun: Illustration



Heun: Illustration



Heun: Accuracy

Robotics and Mechatronics Centre

Why is this method second order accurate?

- Let us develop a 2nd order Taylor-Approximation for f().

$$\mathbf{x}_{k+1} = \mathbf{x}_k + h \cdot f(\mathbf{x}_k,t_k) + h^2/2 \cdot df(\mathbf{x}_k,t_k)/dt + O(h^3)$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k + h \cdot f(\mathbf{x}_k,t_k) + h^2/2 \cdot (\partial f(\mathbf{x}_k,t_k)/\partial x \cdot d\mathbf{x}_k/dt + \partial f(\mathbf{x}_k,t_k)/ \partial t) + O(h^3)$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k + h \cdot f(\mathbf{x}_k,t_k) + h^2/2 \cdot (\partial f(\mathbf{x}_k,t_k)/\partial x \cdot f(\mathbf{x}_k,t_k) + \partial f(\mathbf{x}_k,t_k)/ \partial t) + O(h^3)$$

Heun: Accuracy

Robotics and Mechatronics Centre

Why is this method second order accurate?

- If we reformulate Heun just in terms of $\mathbf{x}_k$, $\mathbf{x}_{k+1}$ and f() we get:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + h/2 \cdot (\, f(\mathbf{x}_k + h \cdot f(\mathbf{x}_k,t_k)\,, t_k+h) + f(\mathbf{x}_k,t_k)\,)$$

- We now express $f(\mathbf{x}_k + h \cdot f(\mathbf{x}_k,t_k)\,, t_k+h)$ as linear 2D-Taylor Approximation of $f(\mathbf{x},t)$ around $f(\mathbf{x}_k,t_k)$:

$$f(\mathbf{x}_k + h \cdot f(\mathbf{x}_k,t_k)\,, t_k+h) = f(\mathbf{x}_k,t_k) + h \cdot f(\mathbf{x}_k,t_k) \cdot \partial f(\mathbf{x}_k,t_k)/\partial x + h \cdot \partial f(\mathbf{x}_k,t_k)/\partial t$$

- Plugging into the original equation yields:

$$\mathbf{x}_{k+1} = \mathbf{x}_k + h/2 \cdot (f(\mathbf{x}_k,t_k) + h \cdot f(\mathbf{x}_k,t_k) \cdot \partial f(\mathbf{x}_k,t_k)/\partial x + h \cdot \partial f(\mathbf{x}_k,t_k)/\partial t + f(\mathbf{x}_k,t_k)\,)$$

$$\mathbf{x}_{k+1} = \mathbf{x}_k + h \cdot f(\mathbf{x}_k,t_k) + h^2/2 \cdot (\partial f(\mathbf{x}_k,t_k)/\partial x \cdot f(\mathbf{x}_k,t_k) + \partial f(\mathbf{x}_k,t_k)/\partial t)$$

- This is identical to the 2nd order Taylor approximation!

Butcher tableau

Robotics and Mechatronics Centre

Is Heun the only 2nd order Method with two evaluations of f()?

- Let us generalize this procedure by introducing coefficients α and β:

prediction:
$$d\mathbf{x}_k = f(\mathbf{x}_k, t_k)$$
$$\mathbf{x}^P = \mathbf{x}_k + h \cdot \beta_{11}\, d\mathbf{x}_k$$

correction:
$$d\mathbf{x}^P = f(\mathbf{x}^P, t_k + \alpha_1 h)$$
$$\mathbf{x}_{k+1} = \mathbf{x}_k + h \cdot (\beta_{21} d\mathbf{x}_k + \beta_{22} d\mathbf{x}^P)$$

- The coefficients are typically arranged in the Butcher tableau

$$\begin{array}{c|cc} 0 & 0 & 0 \\ \alpha_1 & \beta_{11} & 0 \\ \hline 1 & \beta_{21} & \beta_{22} \end{array} \qquad \begin{array}{c|cc} 0 & 0 & 0 \\ 1 & 1 & 0 \\ \hline 1 & 0.5 & 0.5 \end{array}$$

- On the right, you see the Butcher tableau of Heun.

Midpoint Rule

Robotics and Mechatronics Centre

- If we repeat the analysis for the Heun formula with the generalized coefficients, we get:

$$
\begin{aligned}
\mathbf{x}_{k+1} = \mathbf{x}_k \\
&+ h \cdot (\beta_{21} + \beta_{22}) \cdot f(\mathbf{x}_k,t_k) \\
&+ h^2/2 \cdot (2\, \beta_{11}\beta_{22} \partial f(\mathbf{x}_k,t_k)/\partial x \cdot f(\mathbf{x}_k,t_k) + 2\, \alpha_1\beta_{22}\, \partial f(\mathbf{x}_k,t_k)/ \partial t)
\end{aligned}
$$

- In order to be 2nd order accurate, the following equations must hold:

$$\beta_{21} + \beta_{22} = 1$$
$$2\beta_{11}\beta_{22} = 1$$
$$2\alpha_1\beta_{22} = 1$$

- This is evidently true for Heun, but there is a second solution. **The midpoint rule:**

$$
\begin{array}{c|cc}
0 & 0 & 0 \\
0.5 & 0.5 & 0 \\
\hline
1 & 0 & 1
\end{array}
$$

Higher-Order RK Methods

Heun contains 2 sub-steps. If we allow *n* steps, we get more coefficients and can derive even higher-order methods:

step 0: $d\mathbf{x}^{P0} = f(\mathbf{x}_k, t_k)$

…

step i: $\mathbf{x}^{Pi} = \mathbf{x}_k + h \cdot (\beta_{i1} d\mathbf{x}^{P0} + \beta_{i2} d\mathbf{x}^{P1} + \ldots + \beta_{ii} d\mathbf{x}^{P(i-1)})$

$d\mathbf{x}^{Pi} = f(\mathbf{x}^{Pi}, t_k + \alpha_i h)$

…

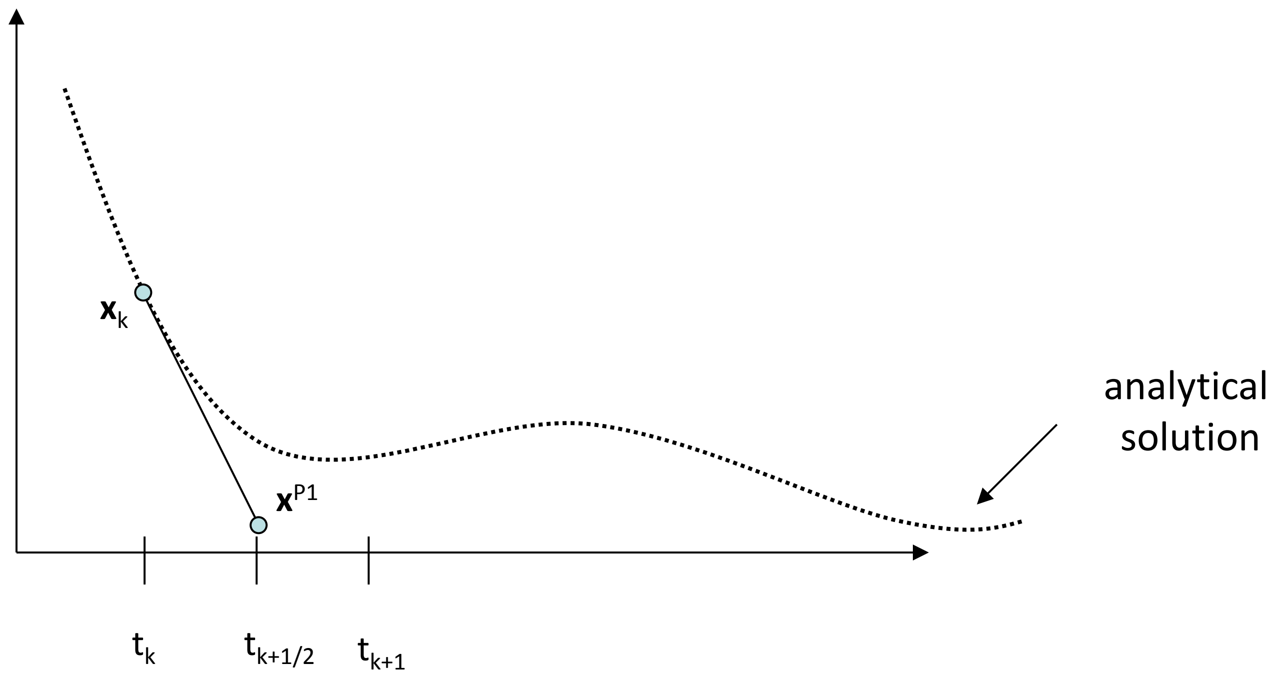
final step n: $\mathbf{x}_{k+1} = \mathbf{x}_k + h \cdot (\beta_{n1} d\mathbf{x}^{P0} + \beta_{n2} d\mathbf{x}^{P1} + \ldots + \beta_{nn} d\mathbf{x}^{P(n-1)})$

RK4

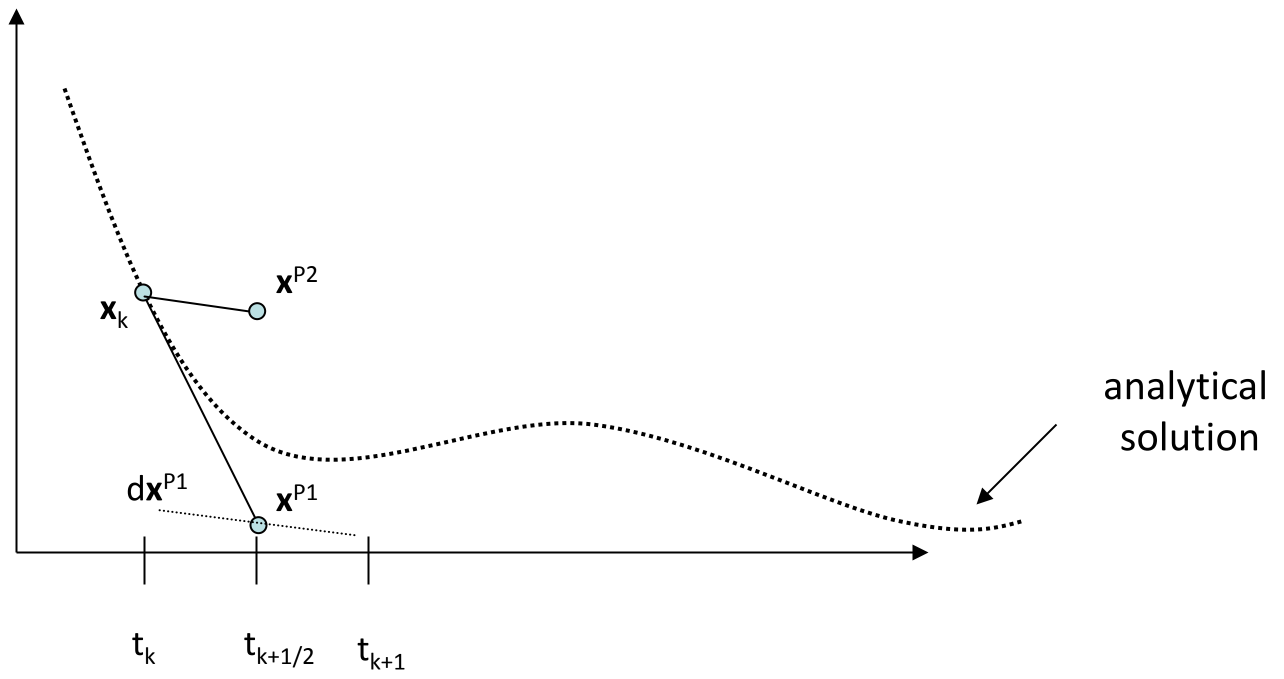
The best known algorithm from this class of numerical integration methods is the 4th-order accurate Runge-Kutta (RK4) algorithm characterized by the following Butcher tableau:

$$\begin{array}{c|cccc}
0 & 0 & 0 & 0 & 0 \\
1/2 & 1/2 & 0 & 0 & 0 \\
1/2 & 0 & 1/2 & 0 & 0 \\
1 & 0 & 0 & 1 & 0 \\
\hline
1 & 1/6 & 1/3 & 1/3 & 1/6
\end{array}$$

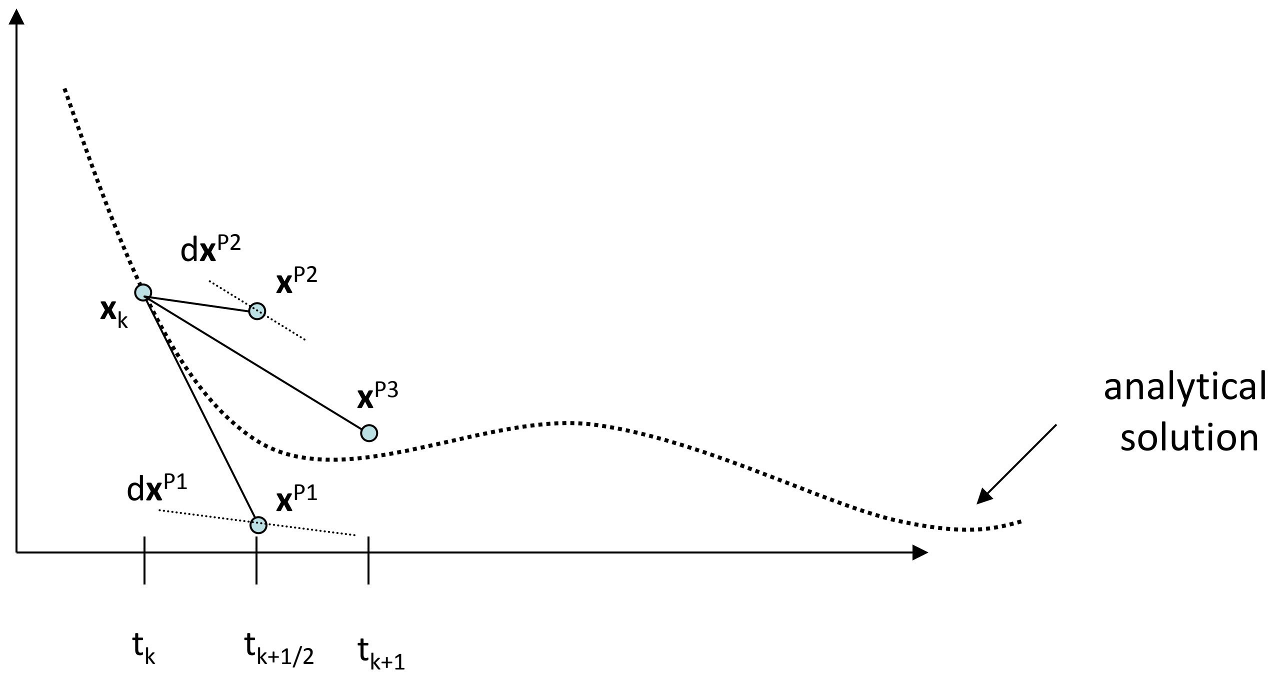
RK4: Illustration



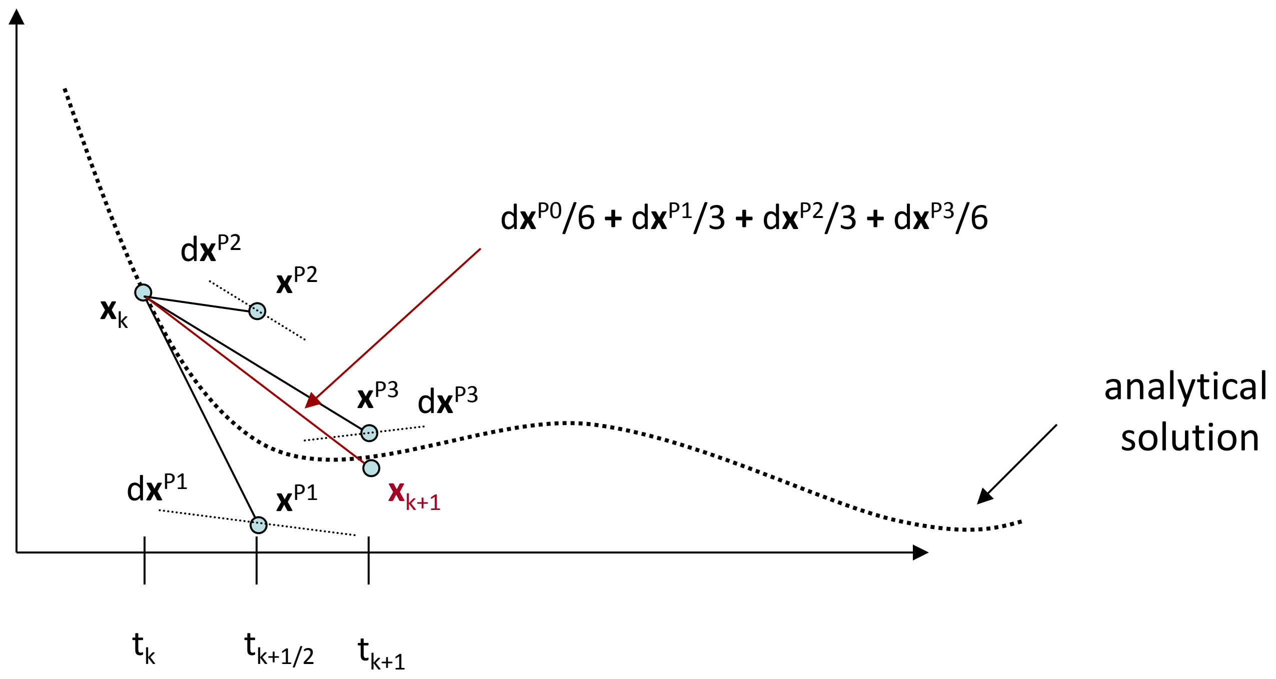
RK4: Illustration



RK4: Illustration



RK4: Illustration



Various RK Methods

Over time, many RK methods have been developed:

Developer	Year	Order	# of Stages
Euler	1768	1	1
Runge	1895	4	4
Heun	1900	2	2
Kutta	1901	5	6
Huta	1956	6	8
Shanks	1966	7	9
Curtis	1970	8	11

- The number of non-linear equations grows rapidly with the order of the methods. Already for RK methods of order 5, there no longer exists a solution in 5 stages. More stages must be added in order to increase the number of parameters.
- In recent years, a sequence of yet higher-order RK methods were developed quite rapidly using computer algebra methods (Maple, Mathematica).

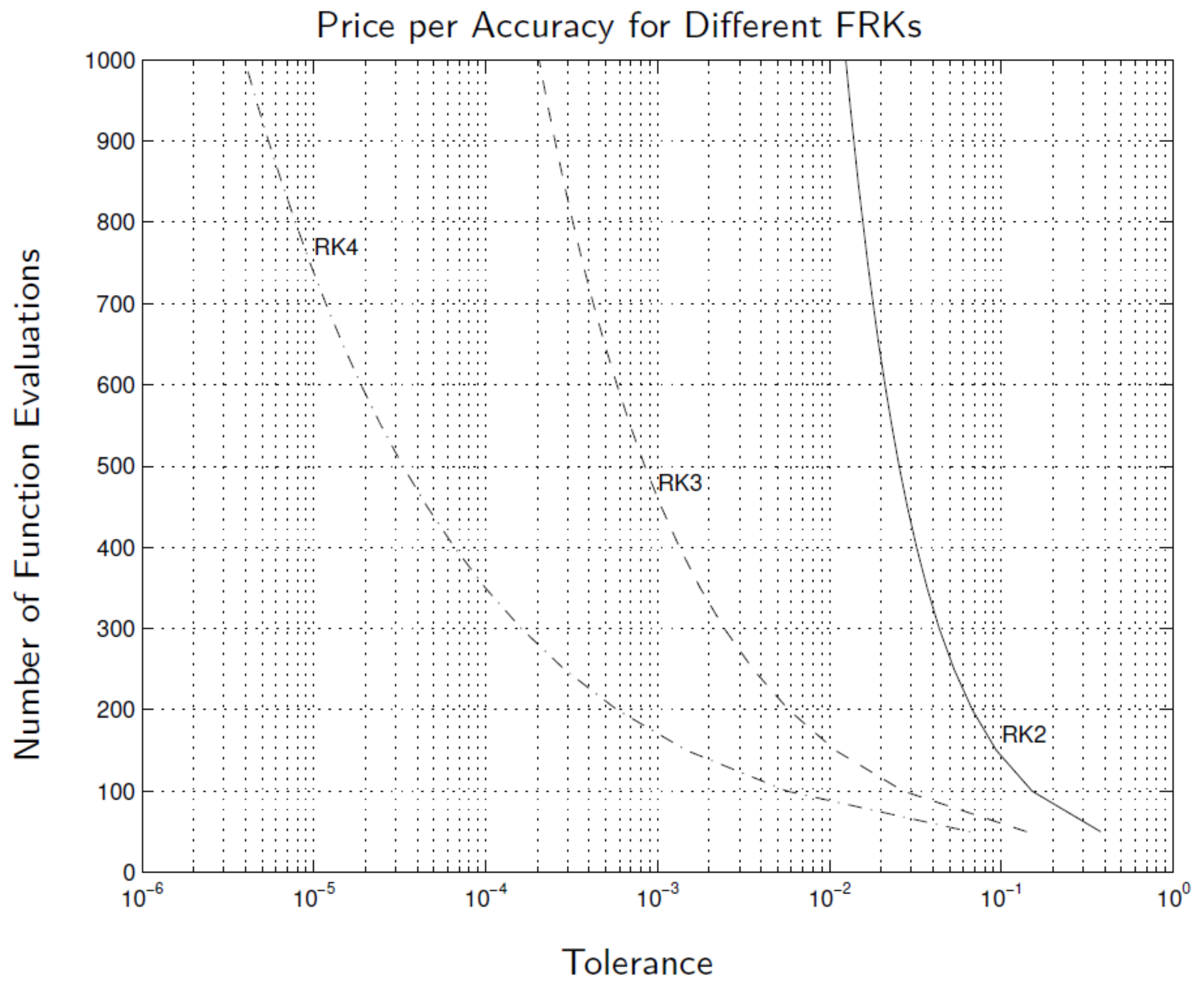
RK: Computational Effort

Robotics and Mechatronics Centre

A *n*th-order method is more precise than a first-order method but it involves also more function evaluations per integration step. Is it worth its price?

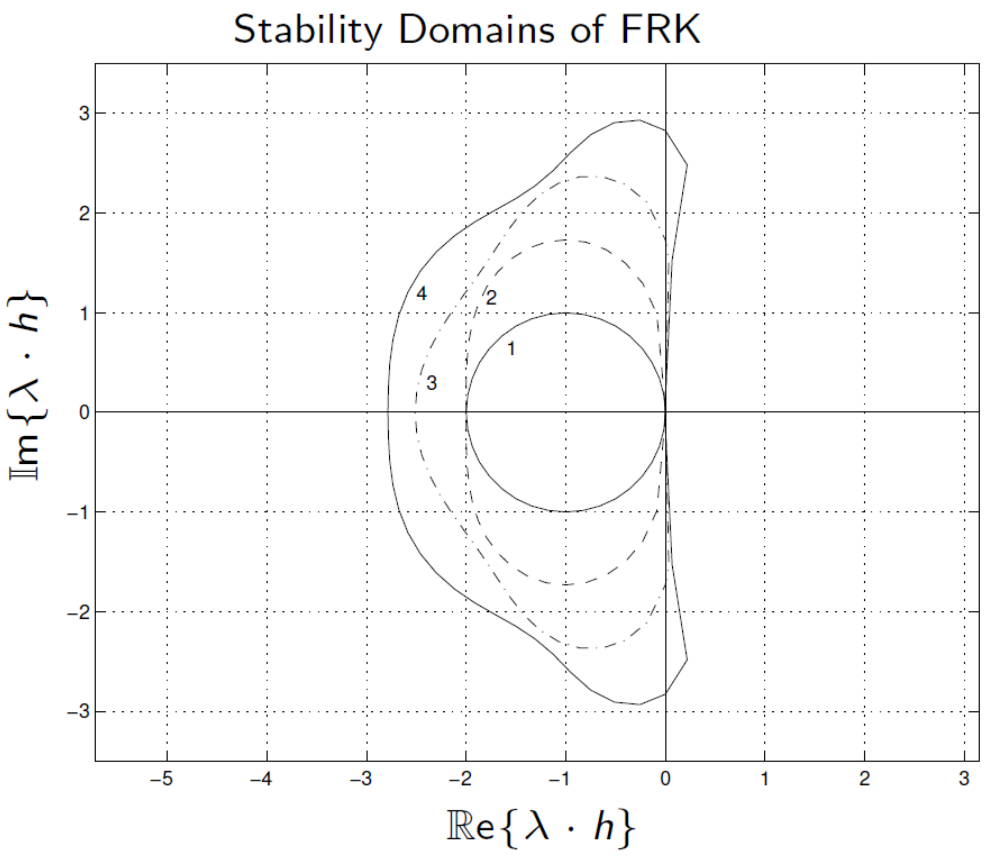
- The answer is evident:

  YES!
  It is worth every penny!



RK: Stability

What about the stability?

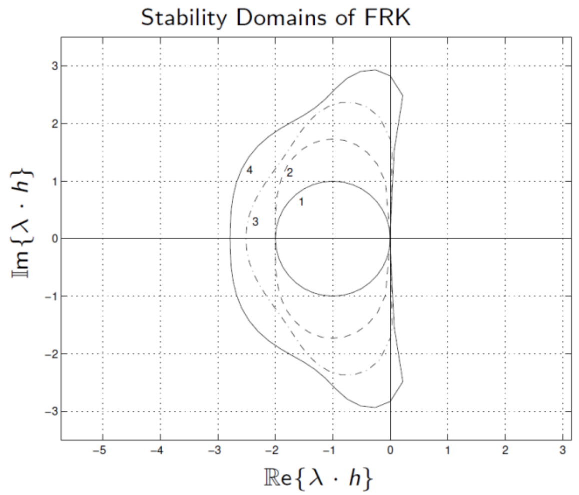


RK: Stability

Robotics and Mechatronics Centre

What about the stability?

- nth-order methods in n stages have the same stability domain. For orders higher than 5 the stability domain depends on the concrete Butcher tableau.



- Although, higher order methods gain a lot in precision, the stability domain grows rather modestly.

- For stiff systems, all RK-methods are almost as bad as FE. We are still bound to use very small step-sizes.

- However, for oscillating systems (with eigenvalues near the imaginary axis), the situation improves significantly from RK2 on.

Local Integration Error Estimation

In order to control the step-size, we would like to have an estimate of the local integration error for each step.

- One way, is to perform the same step twice using two different integration methods.
- Using their results ($\mathbf{x}_1$ and $\mathbf{x}_2$) we may estimate the relative error $\varepsilon_{\mathrm{rel}}$

$$\varepsilon_{rel} = |\mathbf{x}_1 - \mathbf{x}_2| \,/\, \max(|\mathbf{x}_1|, |\mathbf{x}_2|, \delta)$$

δ is a small fudge value > 0

- We can now compare ε_{rel} with the desired tolerance tol_{rel} and control the step-size accordingly

RKF 4/5

It is not efficient to perform the same step twice. Hence, Fehlberg managed to integrate an RK4 method into an RK5 method.

- Here is the Butcher Tableau for Runge-Kutte-Fehlberg 4/5:

0	0	0	0	0	0	0
1/4	1/4	0	0	0	0	0
3/8	3/32	9/32	0	0	0	0
12/13	1932/2197	-7200/2197	7296/2197	0	0	0
1	439/216	-8	3680/513	-845/4104	0	0
1/2	-8/27	2	-3544/2565	1859/4104	-11/40	0
$\mathbf{x}_1$	25/216	0	1408/2565	2197/4104	−1/5	0
$\mathbf{x}_2$	16/135	0	6656/12825	28561/56430	−9/50	2/55

- Now the estimation of the local integration error is virtually for free.

Step-Size Control

Using $\varepsilon_{rel} = |\mathbf{x}_1 - \mathbf{x}_2| \,/\, \max(|\mathbf{x}_1|,|\mathbf{x}_2|,\delta)$, we can develop a step-size control for RKF 4/5

- $\mathbf{x}_1$ is a 4th order approximation whereas $\mathbf{x}_2$ is a 5th order approximation
- Hence the relative error ε_{rel} is proportional to h^5
- It is therefore meaningful to state:

$$\varepsilon_{rel} / \mathrm{tol}_{rel} = (h_{\mathrm{old}} / h_{\mathrm{new}})^5$$

$$\Rightarrow \quad \mathrm{h}_{\mathrm{new}} = \sqrt[5]{\mathrm{tol}_{rel}/\varepsilon_{rel}}\; h_{\mathrm{old}}$$

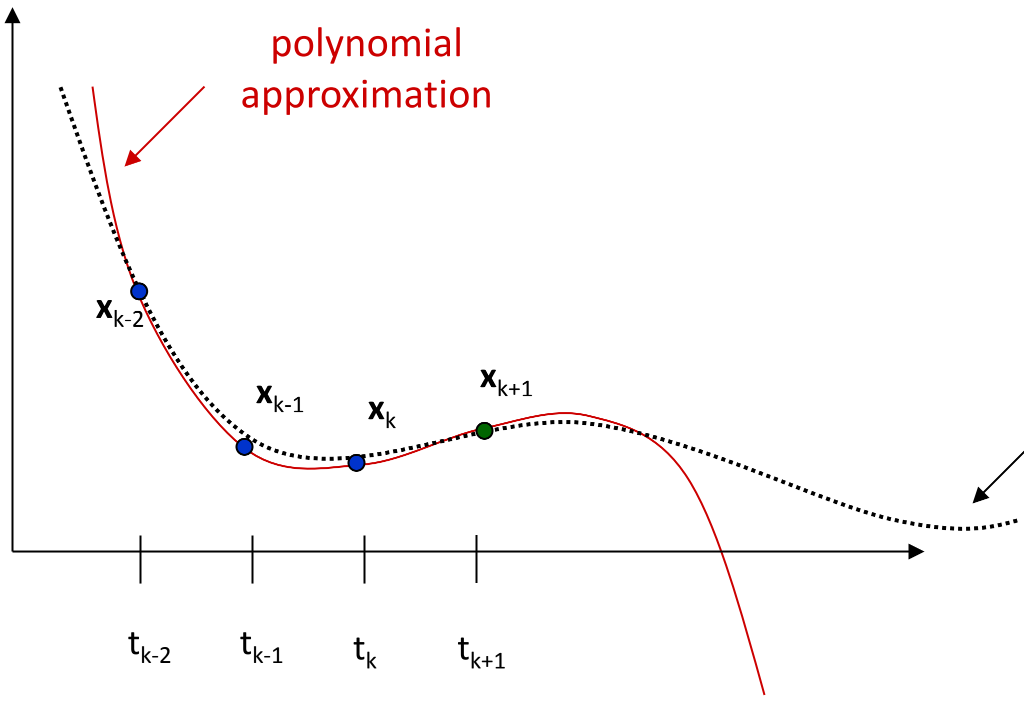
- In this way, if the error is too large, the next step is reduced, and if the error is unnecessarily small, the next step is increased.
- This is an ***optimistic strategy*** since steps are never repeated, even if the error is excessively large

Single-Step

- So far, every integration step was independent from its preceding integrations steps.
- At each step, we have performed a single-step (with some sub-steps). The information contained in the previous integration steps has been discarded.
- Hence the RK methods are called ***single-step methods***.
- But there are also ***multi-step methods***….

Multi-Step

- The idea behind every multi-step method is to generate a higher-order method by a polynomial approximation of the previous steps.
- The figure below illustrates this principle:



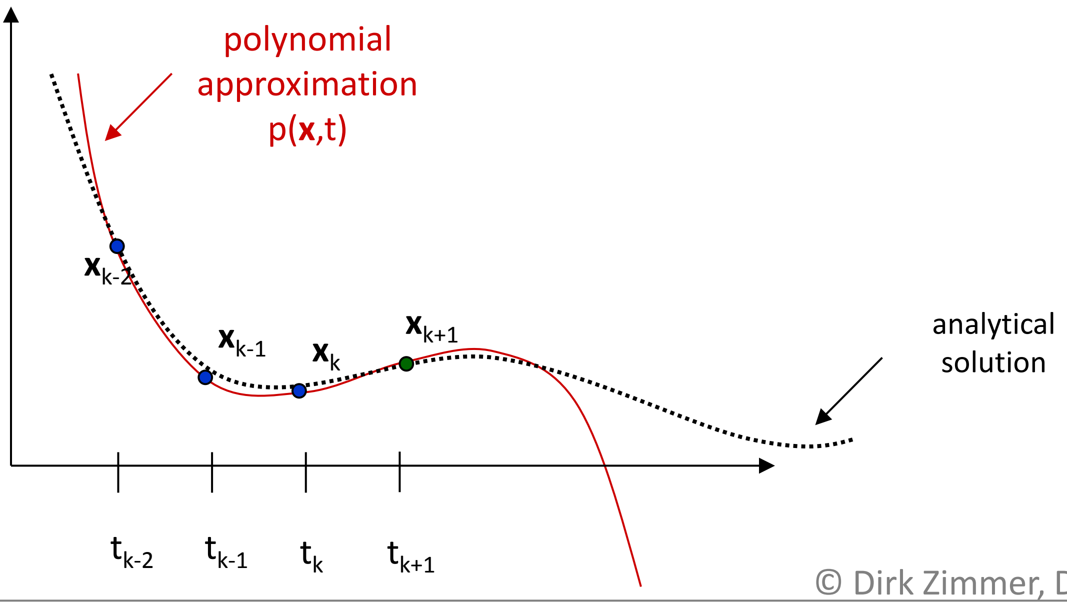
Backwards Difference Formula

Robotics and Mechatronics Centre

- Let p() be the polynomial that approximates the last n-1 steps: $\mathbf{x}_{k+1}$, $\mathbf{x}_{k}$ … $\mathbf{x}_{k-n+1}$
- We can then generate a nth-order method by solving the following equation for $\mathbf{x}_{k+1}$:

$$f(\mathbf{x}_{k+1},t_{k+1}) = dp(\mathbf{x}_{k+1},t_{k+1})/dt$$

- Hence this represents an implicit method (like BE).



Backwards Difference Formula

Robotics and Mechatronics Centre

- Let p() be the polynomial that approximates the last n-1 steps: $\mathbf{x}_{k+1}$, $\mathbf{x}_k$ … $\mathbf{x}_{k-n+1}$
- We can then generate a n^th-order method by solving the following equation for $\mathbf{x}_{k+1}$:

$$f(\mathbf{x}_{k+1},t_{k+1}) = dp(\mathbf{x}_{k+1},t_{k+1})/dt$$

- Hence this represents an implicit method (like BE).
- **Remark:** Since this is an implicit method, it can also be directly applied to the implicit DAE form 0 = F(d**x**/dt,**x**,**u**,t):

$$0 = F(dp(\mathbf{x}_{k+1},t_{k+1})/dt,\ \mathbf{x}_{k+1},\mathbf{u}_{k+1}t_{k+1})$$

These kind of solvers can also be applied on systems that have not undergone index-reduction (but it is not necessarily an efficient way to do so..)

Backwards Difference Formula

Robotics and Mechatronics Centre

- If we perform the integration with a fixed step-size, all points for the polynomial approximation are equally-distant spaced in time. This enables the usage of Newton-Gregory polynomials for p().

- $dp(\mathbf{x}_{k+1},t_{k+1})/dt$ can be expressed as weighted sum of $\mathbf{x}_k$:

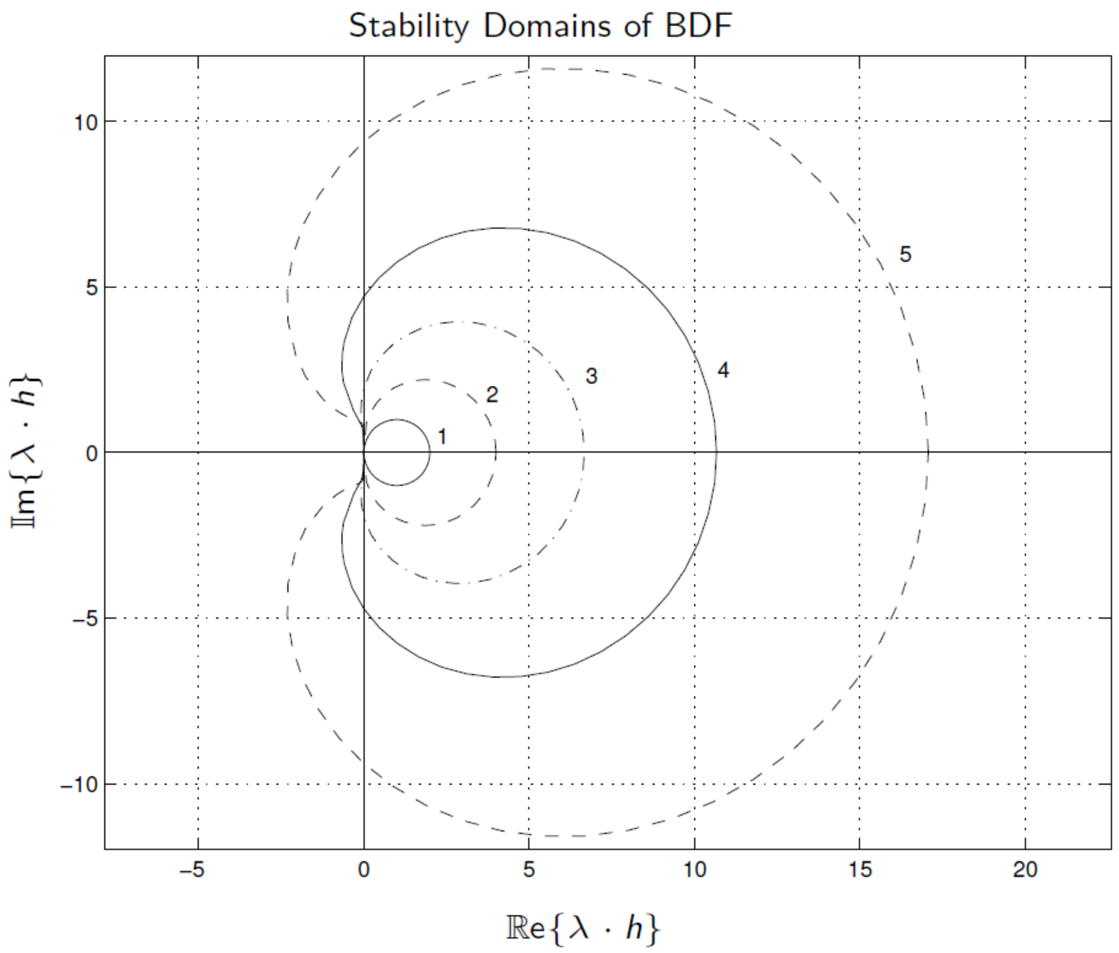
$$dp(\mathbf{x}_{k+1},t_{k+1})/dt \cdot h = \alpha_1 \mathbf{x}_{k+1} + \alpha_2 \mathbf{x}_k + \ldots + \alpha_n \mathbf{x}_{k-n+1}$$

- This leads to the famous BDF methods:

	α_1	α_2	α_3	α_4	α_5	α_6
BDF 1	1	-1				
BDF 2	3/2	-2	1/2			
BDF 3	11/6	-3	3/2	-1/3		
BDF 4	25/12	-4	3	-4/3	1/4	
BDF 5	137/60	-5	5	-10/3	5/4	-1/5

BDF: Stability

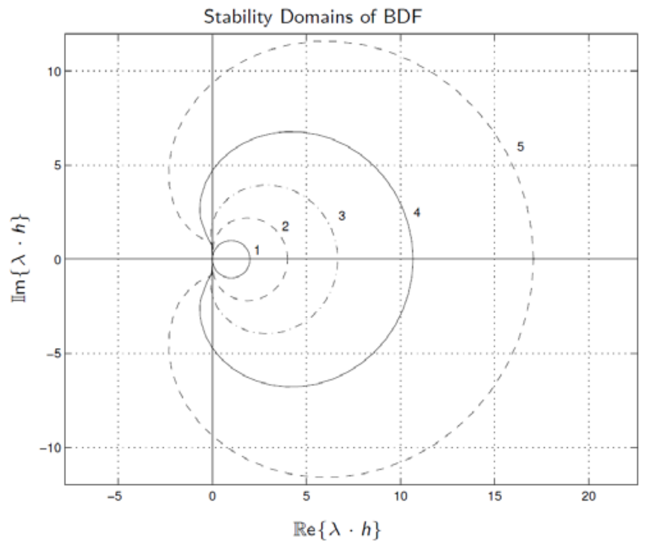
What about the stability?



BDF: Stability

What about the stability?

- BDF 1 is Backward Euler
- BDF 2 is still maintains the analytical stability
- The region of unstability grows significantly with the number of the order.



- From BDF 3 on, the unstable region overlaps the imaginary axis.

- Hence the higher order BDF methods (from 5 on) behave strangely for oscillatory systems.

- Thanks to their suitability for the simulation of stiff systems and due to their simplicity, the BDF algorithms are among the most widely used numerical ODE solvers for the simulation of dynamic systems.

Multi-Step: Startup

- There are some extra problems involved with multi-step methods in general

- One of them is the startup problem.

- We simply assumed, that there for an n^{th}-order method, there are n-1 past values available. At start, this assumption is obviously violated.

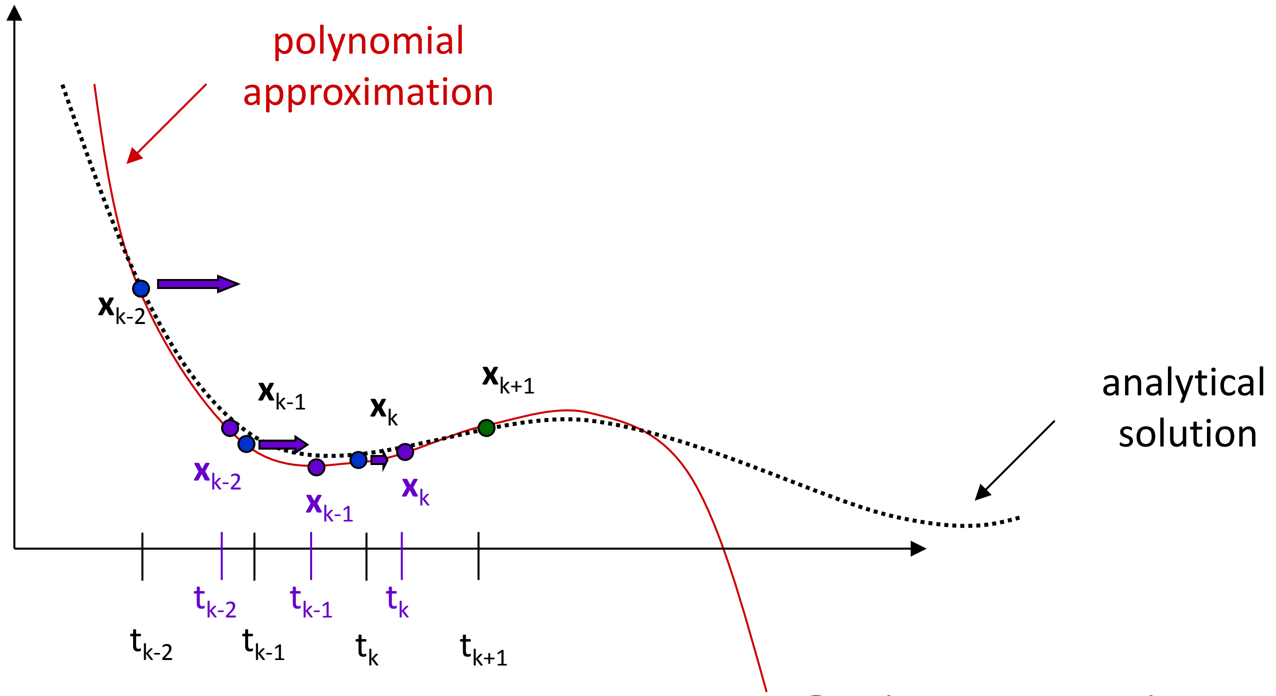
- Essentially, there are two solutions:

  - Work yourself up: Start with BDF 1, then continue with BDF 2, BDF 3 and so on… Unfortunately, the usage of BDF 1 may enforce small step-sizes initially.
  - Kick-start using a single-step method of the appropriate order.
    For instance use 3 steps of RK4 to start BDF 4. However, since RK4 cannot cope with stiff systems, small step-sizes might again be enforced.
  - Most probably the best choice is to use and implicit Runge-Kutta (see later on) method as Kick-start

Multi-Step: Startup

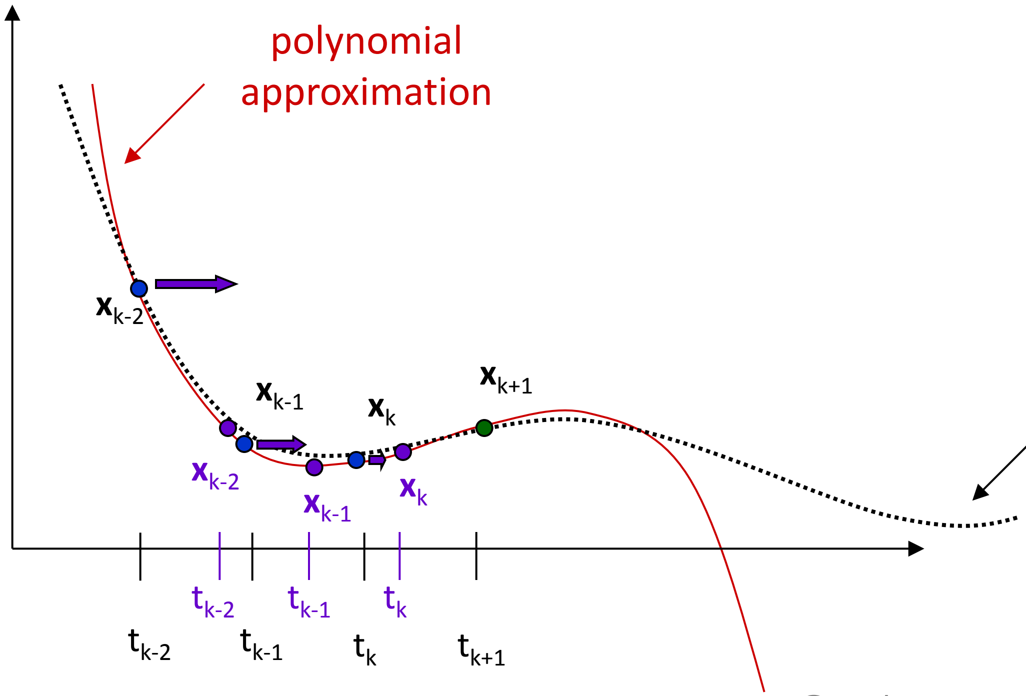
Robotics and Mechatronics Centre

- The other problem is step-size control.
- We have assumed a fixed step-size so far.
- If we want to adapt the step-size, we have to “relocate” our past integration values $\mathbf{x}_k$, $\mathbf{x}_{k-1}$, … $\mathbf{x}_{k-n+1}$



Multi-Step: Startup

- This relocation of $\mathbf{x}_k$, $\mathbf{x}_{k-1}$, … $\mathbf{x}_{k-n+1}$ can be performed with linear operations only and does not involve any loss of precision since the polynomial g() remains unchanged.
- However, the adaption of step-size is relatively costly. Controlling the step-size for each step is not recommendable. Consequently, a more conservative step-size has to be chosen so that overhasty changes can be avoided and the same step-size can be maintained over larger time-spans



Overview

So far, we have investigated two major classes of integration methods

	Explicit	Implicit
Single-Step	Runge-Kutta	
Multi-Step		BDF (DASSL)

Overview

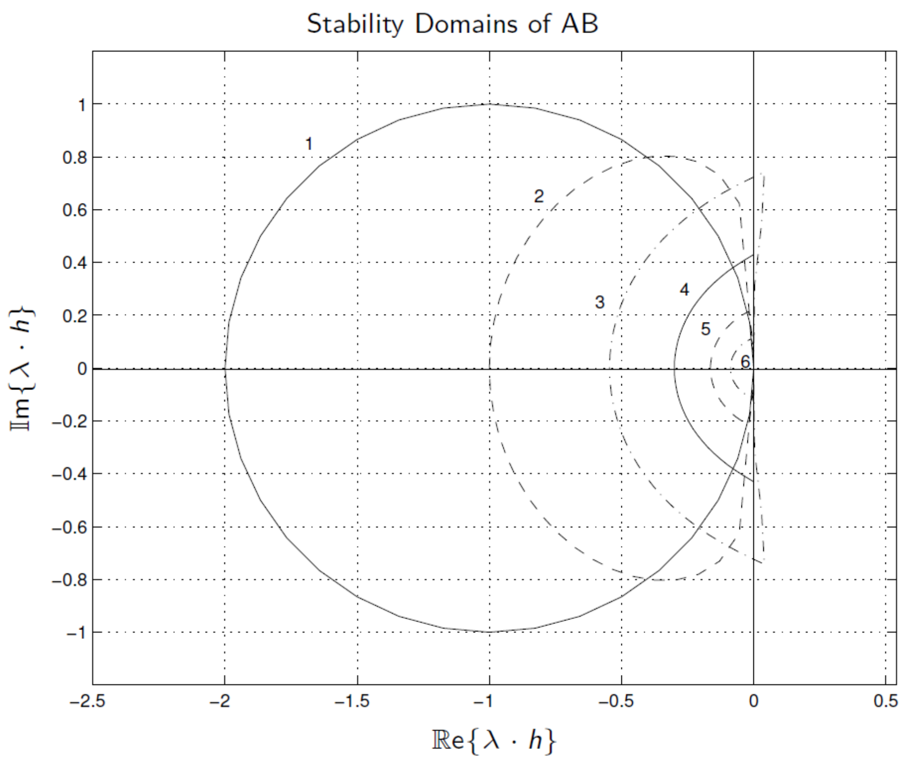
What about explicit multi-step methods. They would be great for real-time simulation or?

	Explicit	**Implicit**
Single-Step	Runge-Kutta	
Multi-Step	?	BDF (DASSL)

Adams-Bashforth

There are explicit multi-step methods available. There are called, for instance, Adams-Bashforth (AB).

- Whereas the implicit multi-step methods are based on polynomial interpolation, the explicit methods are based on polynomial extrapolation. This is potentially dangerous.
- It is therefore no surprise, that AB performs very poorly with respect to stability.
- Indeed, the numerical stable region shrinks for higher-order methods
- AB is practically **only** used for non-stiff, linear systems.



Overview

Wouldn't implicit single step methods be very suited for the simulation of stiff systems?

	Explicit	Implicit
Single-Step	Runge-Kutta	?
Multi-Step	Adams-Bashforth	BDF (DASSL)

Implicit RK

The classic RK methods enabled an explicit computation since the Butcher Tableau does not contain entries at or above the diagonal.

- Here is the Butcher Tableau of 3rd order Radau IIa. It requires an implicit solver.

$$\begin{array}{c|cc} 1/3 & 5/12 & -1/12 \\ 1 & 3/4 & 1/4 \\ \hline 1 & 3/4 & 1/4 \end{array}$$

$\mathbf{dx}^{P0} = f(\mathbf{x}_k + 5h/12\mathbf{dx}^{P0} - h/12\mathbf{dx}^{P1}\ , t_k + h/3)$

$\mathbf{dx}^{P1} = f(\mathbf{x}_k + 3h/4\mathbf{dx}^{P0} + h/4\mathbf{dx}^{P1}, t_k + h)$

$\Rightarrow \mathbf{x}_{k+1} = \mathbf{x}_k + 3h/4\mathbf{dx}^{P0} + h/4\mathbf{dx}^{P1}$

$\mathbf{dx}^{P0}$ and $\mathbf{dx}^{P1}$ need to be found by an iterative solver. Hence, Radau IIa (3rd order) requires twice as many iteration variables as BE and a single step is though (roughly) $2^3=8$ times more expensive as an BDF step of the same order.

Implicit RK

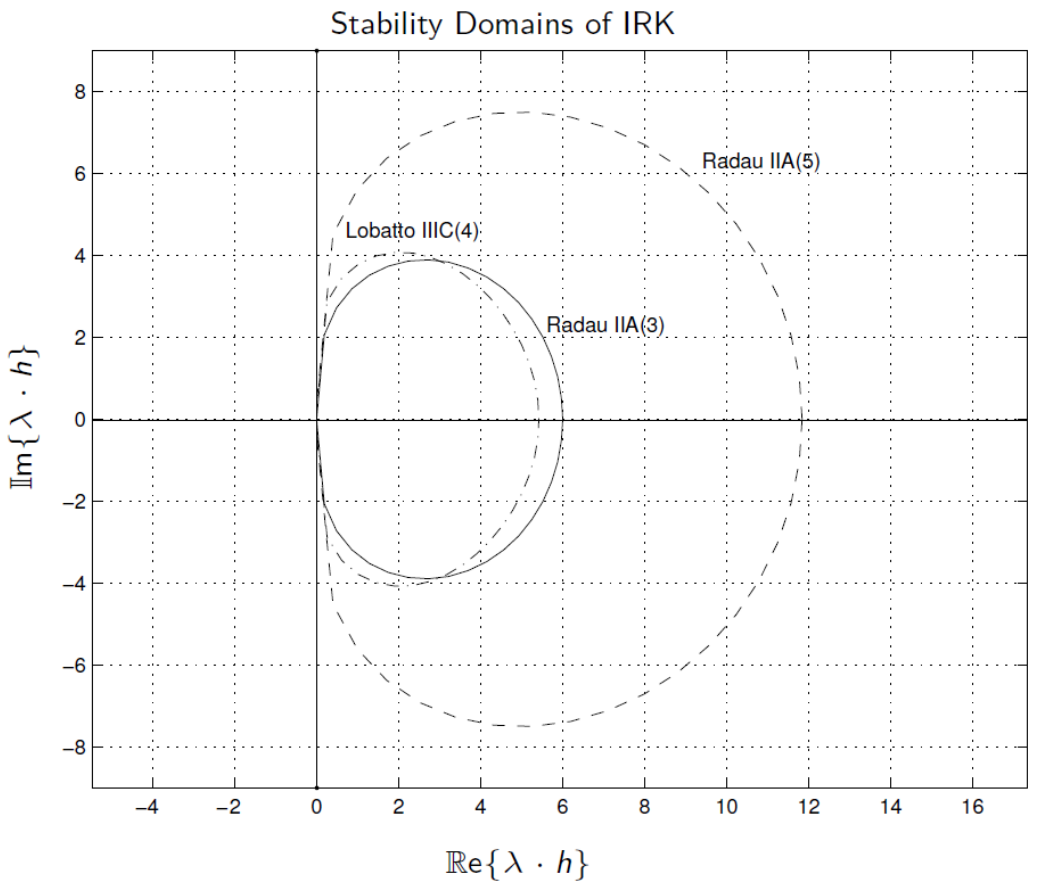
Robotics and Mechatronics Centre

The classic RK methods enabled an explicit computation since the Butcher Tableau does not contain entries at or above the diagonal.

- Here is the Butcher Tableau of 3rd order Radau IIa. It requires an implicit solver.

$$\begin{array}{c|cc} 1/3 & 5/12 & -1/12 \\ 1 & 3/4 & 1/4 \\ \hline 1 & 3/4 & 1/4 \end{array}$$

- Implicit RK methods behave excellently w.r.t. stability
- However, a step of IRK is significantly more expensive than a step of BDF of the same order.
- However, IRK feature a more flexible step-size control and can compensate for this in highly non-linear or discon-tinuous systems



Overview

Robotics and Mechatronics Centre

So.. this is what we have learned today:

	Explicit	**Implicit**
Single-Step	Runge-Kutta	Implicit Runge-Kutta (Radau IIa)
Multi-Step	Adams-Bashforth	BDF (DASSL)

Questions ?