Basics and Euler Integrators

Numerical Simulation of Physical Systems

Markus Andres, Thomas Schmitt

FHV
University of Applied Sciences

April 27, 2016

Content

Solvers? What for?

FHV

Remember: Modeling and Simulation

Mathematical Modeling of Physical (Dynamic) Systems
Here, we basically try to describe a real (technical) system as accurate as needed to gain information about, e.g. the dynamic behavior.

Numerical Simulation of Dynamic Systems
Thus, we need appropriate numerical integration methods, better known as solvers, that basically numerically solve our model, i.e. system of differential equations.

What we Gain when we "Model" a System

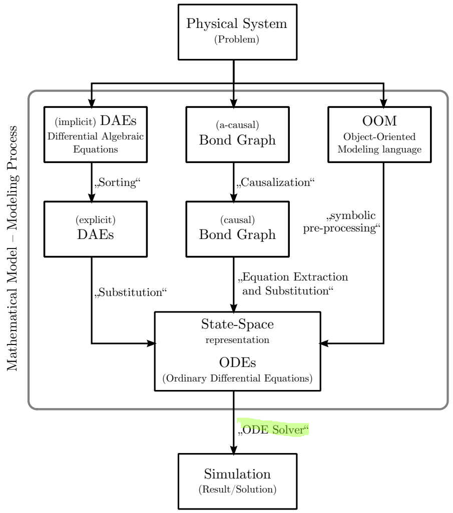
When **modeling**, we come up with some **mathematical description** of a physical system, that describes the behavior of the system. There are different ways to get there e.g.

- Classical Modeling using differential algebraic equations
- Bond-Graphs
- Object-Oriented Modeling
- and some more...

Modeling

All modeling techniques help us to come up with some representation of the system we want to simulate. Still this is not part of this course.

Some ways of Modeling



What we Gained by Modeling

State-Space Representation

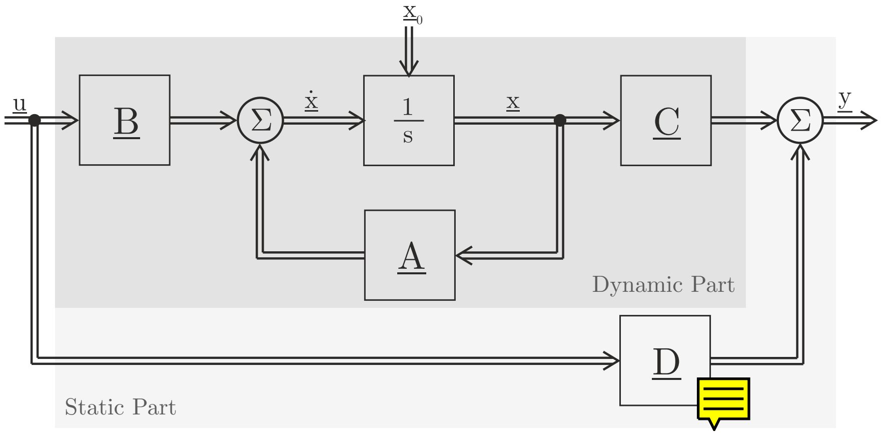
So basically what we gained through our modeling efforts are four matrices represented by

$$\underline{\dot{x}}(t) = \underline{A} \cdot \underline{x}(t) + \underline{B} \cdot \underline{u}(t), \quad \underline{x}(0) = \underline{x}_0 \tag{1}$$

$$\underline{y}(t) = \underline{C} \cdot \underline{x}(t) + \underline{D} \cdot \underline{u}(t) \tag{2}$$

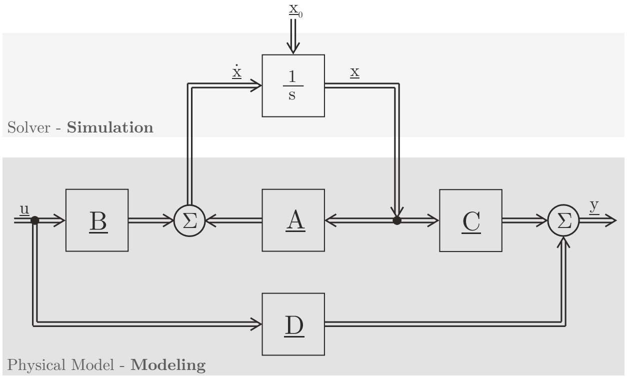
Not very helpful, is it?

The State-Space Representation



That is what we usually see when we get a graphical version of the state-space representation.

A Rearranged State-Space Representation



Modeling AND Simulation

By utilizing an integrator that computes x from $\dot{x}$, we can compute the output depending on the system dynamics and time-varying inputs.

The $\frac{1}{s}$ Magic

The $\boxed{\frac{1}{s}}$ block on the last slide computes the values of states in the state-space model. In the state-space representation this block represents an ideal (analytical) integrator.

- Ideal integration is exactly what is done when **solving** differential equations **by hand** or in a symbolic fashion by a computer.
- It gets **very cumbersome** or even impossible for relatively basic technical systems.
- Non-linearities are very hard to handle.
- Discrete events are not very funny either.

Discretization

To be able to compute any continuous values in a numerical fashion, we have to discretize something.

There are two possibilities:

- Time
- State Variables

Discretizing time is very common and usually comes to the mind of nearly every engineer. It is basically the one of the two possibilities that is used throughout all areas of simulation. Still there is an alternative: **QSS**

Quantized State Simulation (QSS)

All of the algorithms presented from now on are based on the discretization of time. Recent research suggests that it could be beneficial to discretize the states rather than the time. Basically, a new simulation step is then carried out when the **state is expected to change** more than a predefined band allows it to.

Still this is a very young field of research and therefore **no commercial implementation** is expected in the upcoming years.

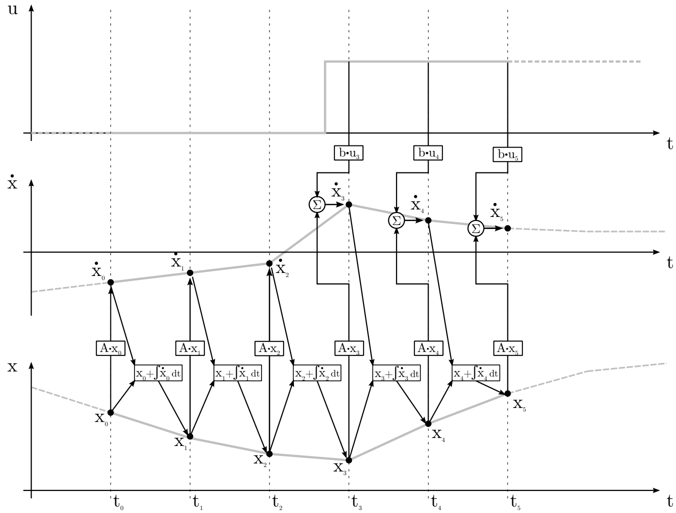
We will not further investigate on QSS. If you are interested in its benefits take a look at [1] and the tool PowerDEVS.

Why does Time have to be Discrete?

If we could solve the model equation analytically we could compute the solution at any point in time without any error. Still it usually is inefficient to solve complex models analytically.

- The "thing" we approximate is the integrator.
- The approximation is done by making the simulation time discrete.
- There will be a difference between the analytical (correct) solution and the approximation
- The smaller the steps are, the smaller the resulting difference (the error) will be.

The Process of Simulation with Discretized Time



Numerical Integration

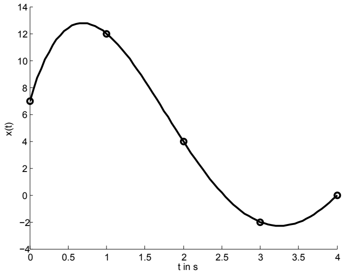
To discretize time we utilize the **Taylor Series** to approximate the trajectory of the states.

$$x_i(t+h) = x_i(t) + \underbrace{\frac{dx_i(t)}{dt}}_{\dot{x}(t)} \cdot h + \underbrace{\frac{d^2x_i(t)}{dt^2}}_{\ddot{x}(t)} \cdot \frac{h^2}{2!} + \cdots \tag{3}$$

with h being the time between the start point and the point that we want to describe.

An Interpretation of the Taylor Series

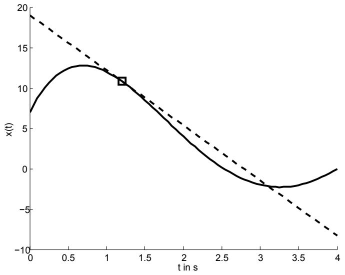
We quickly repeat what the Taylor Series does for us. As an example we try to describe a polynomial of 4^{th} order by a Taylor Series.



Now we will compute the Taylor Series with a rising amount of terms.

An Interpretation of the Taylor Series

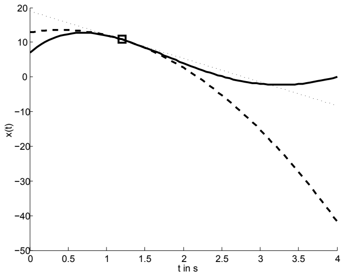
If we truncate the Taylor Series after the linear term we come up with:



$$x_i(t+h) = x(t) = x_i(t)\big|_{t=1.2} + \underbrace{\left.\frac{dx_i(t)}{dt}\right|_{t=1.2}}_{\dot{x}(t)|_{t=1.2}} \cdot h \tag{4}$$

An Interpretation of the Taylor Series

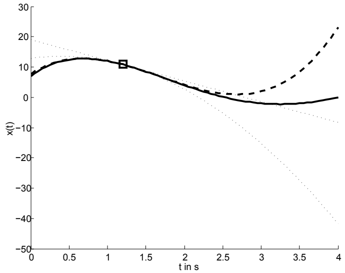
Truncating after the quadratic term leads to:



$$x(t) = x_i(t)\big|_{t=1.2} + \left.\frac{dx_i(t)}{dt}\right|_{t=1.2} \cdot h + \left.\frac{d^2x_i(t)}{dt^2}\right|_{t=1.2} \cdot \frac{h^2}{2!} \tag{5}$$

An Interpretation of the Taylor Series

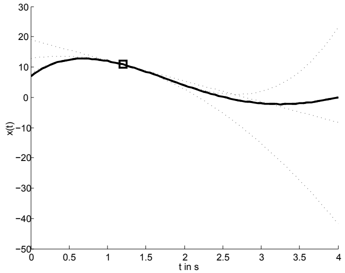
Truncation after the cubic term:



$$x(t) = x_i(t)\big|_{t=1.2} + \left.\frac{dx_i(t)}{dt}\right|_{t=1.2} \cdot h + \left.\frac{d^2x_i(t)}{dt^2}\right|_{t=1.2} \cdot \frac{h^2}{2!} + \left.\frac{d^3x_i(t)}{dt^3}\right|_{t=1.2} \cdot \left.\frac{h^3}{3!}\right|_{t=1.2} \qquad (6)$$

An Interpretation of the Taylor Series

Truncation after the fourth order term:



$$x(t) = x_i(t)|_{t=1.2} + \left.\frac{dx_i(t)}{dt}\right|_{t=1.2} \cdot h + \left.\frac{d^2x_i(t)}{dt^2}\right|_{t=1.2} \cdot \frac{h^2}{2!} + \left.\frac{d^3x_i(t)}{dt^3}\right|_{t=1.2} \cdot \left.\frac{h^3}{3!}\right|_{t=1.2} + \left.\frac{d^4x_i(t)}{dt^4}\right|_{t=1.2} \cdot \frac{h^4}{4!} \quad (7)$$

Taylor Series Conclusions

So what do we get from that?

- For a function that we can **differentiate**, we can **predict** how it will behave in the future.
- The **more often** we can differentiate the more precise the prediction will be.
- For a **polynomial** of n-th order the result will be a perfect prediction after n derivatives.
- For functions like **trigonometric** or **exponential functions** we will need an infinitive amount derivatives to describe it perfectly.

Using a Model with the Taylor Series

Plugging

$$\dot{x}(t) = f(x(t), u(t), t) = f(t) \tag{8}$$

$$f(t) = A \cdot x(t) + b \cdot u(t) \tag{9}$$

Das ist das Modell

in the Taylor Series,

$$x_i(t+h) = x_i(t) + \underbrace{\frac{dx_i(t)}{dt}}_{\dot{x}(t)} \cdot h + \underbrace{\frac{d^2 x_i(t)}{dt^2}}_{\ddot{x}(t)} \cdot \frac{h^2}{2!} + \cdots \tag{10}$$

results in:

$$x_i(t+h) = x_i(t) + f_i(t) \cdot h + \frac{df_i(t)}{dt} \cdot \frac{h^2}{2!} + \cdots \tag{11}$$

The model computes derivatives

It is important to remember, that the model computes the first derivative of the actual states from the actual states and the input. This is how the Taylor Series is combined with our model.

Autonomous Systems

For the following observations, it is sufficient to use **autonomous linear time-invariant systems**. Therefore we will use

$$f(x, t) = A \cdot x(t) \tag{12}$$

When implementing the solvers it is important to have an input. Then all the derivations shown in these slides can be done by replacing the upper equation with

$$f(x, t) = A \cdot x(t) + b \cdot u \tag{13}$$

This should not cause too much trouble.

Forward Euler

For the first solver (or numerical integrator) we will truncate the Taylor Series after the first term. This is rather obvious as we have all terms available when we have a model of the system.

$$x_i(t+h) = x_i(t) + f_i(t) \cdot h + \frac{df_i(t)}{dt} \cdot \frac{h^2}{2!} + \cdots \tag{14}$$

Resulting in:

$$x_i(t+h) = x_i + f_i(t) \cdot h \tag{15}$$

This is **Forward Euler**.

Forward Euler

We can reformulate this equation in a couple of ways.

$$x(t+h) = x(t) + \dot{x}(t) \cdot h \tag{16}$$

$$x(t+h) = x(t) + f(x,t) \cdot h \tag{17}$$

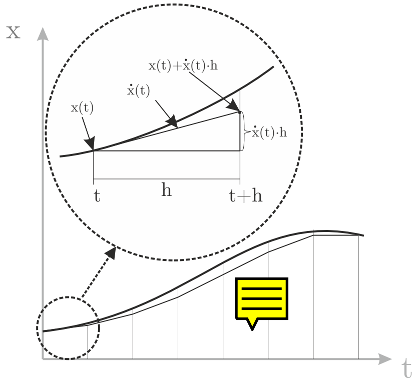
$$x(t+h) = x(t) + A \cdot x(t) \cdot h \tag{18}$$

Simplicity

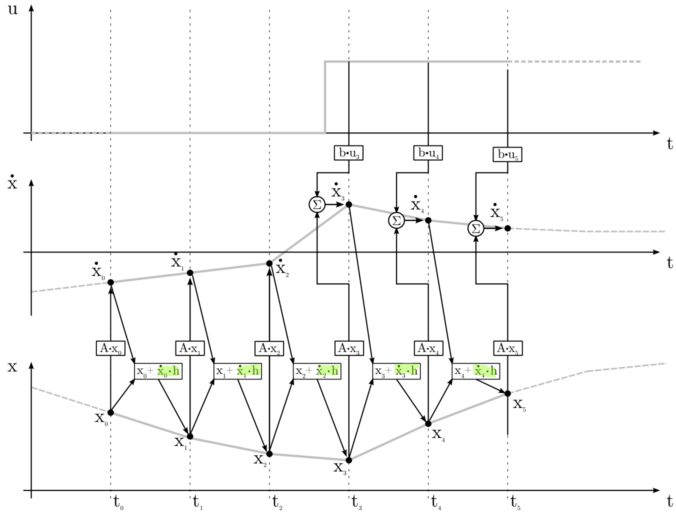
What we found is the Forward Euler (FE) algorithm. The biggest advantage of the FE is, that no additional information is needed and it is simple to understand and predict its behavior.

Forward Euler

A graphical representation of the Forward Euler equation is shown below.



Simulating with Forward Euler



Discretization

There is a **discrete form** of the state-space representation. The analytical form of the equation is

$$\underline{\dot{x}}(t) = \underline{A} \cdot \underline{x}(t) + \underline{B} \cdot \underline{u}(t), \quad \underline{x}(0) = \underline{x}_0 \tag{19}$$

$$\underline{y}(t) = \underline{C} \cdot \underline{x}(t) + \underline{D} \cdot \underline{u}(t) \tag{20}$$

whereas the discrete one is

$$\underline{x}(t+h) = \underline{F} \cdot \underline{x}(t) + \underline{G} \cdot \underline{u}(t), \quad x(0) = x_0 \tag{21}$$

$$\underline{y}(t) = \underline{H} \cdot \underline{x}(t) + \underline{I} \cdot \underline{u}(t) \tag{22}$$

Discretization

For the **transformation** from the analytical to the discrete state-space form, we basically have to plug in the integration algorithm and bring the equations in the form that is shown above. This should usually not be too complicated as we will see on the following slides.

One Important Difference

. . . between the analytical and the discrete form is, that the analytical version computes the derivatives of the state variables, whereas the discrete one computes the real values of the states, i.e. the next values of the state variables for a given step-size h.

Discretization

Remembering the equation for the Forward Euler and applying some formula manipulations we can recognize the following.

$$x(t+h) = x(t) + \dot{x}(t)\cdot h \tag{23}$$
$$x(t+h) = x(t) + A\cdot x(t)\cdot h \tag{24}$$
$$x(t+h) = \underbrace{(\mathbb{1} + A\cdot h)}_{F}\cdot x(t) \tag{25}$$

Therefore and with some knowledge about **z-Transformation** we can make a good guess about the stability region of the Forward Euler[1].

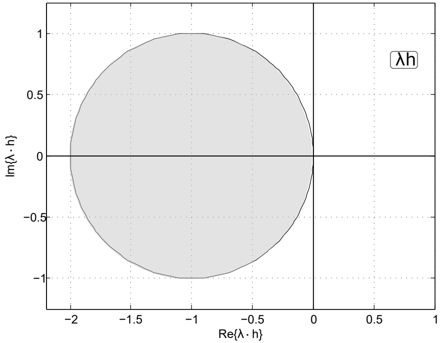
Replacing the Integrator

The term $\dot{x}(t)\cdot h$ in (23) is the replacement (approximation) for the analytical integrator.

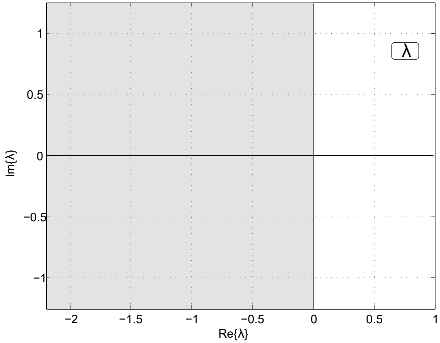
[1] A less intuition based approach will be presented in the next set of slides.

Forward Euler Stability

Stability domain of the **Forward Euler**:



Analytical stability domain:



For:

$$x(t+h) = x(t) + A \cdot x(t) \cdot h \tag{26}$$

$$x(t+h) = \underbrace{(\mathbb{1} + A \cdot h)}_{F} \cdot x(t) \tag{27}$$

Where does the $\lambda \cdot h$ come from?

If we again take a look at the equations

$$x(t+h) = x(t) + A \cdot x(t) \cdot h \tag{28}$$

$$x(t+h) = \underbrace{(\mathbb{1} + A \cdot h)}_{F} \cdot x(t) \tag{29}$$

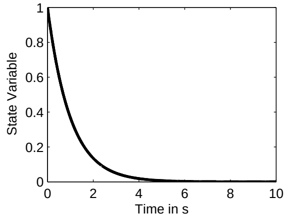
we can see, that A is multiplied with h. Therefore, the properties of the F matrix, like the eigenvalues, are scaled linearly with h. Therefore, we are now able to **move the poles** in the $\lambda \cdot h$ plane. This is not possible in the λ plane because there the poles solely describe properties of the system, whereas in the $\lambda \cdot h$ plane, they describe poles of the **discretized system**.

Testing Stability

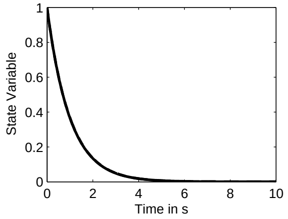
We use the code for a simple system implemented in MATLAB to verify our theory.

```
% Paramters
A = -1;
c = 1;

% Initial Conditions
x0 = 1;

% Simulation Properties
t_start = 0;
t_end = 10;
h = 1e-3;

% Artificial Variables
i = 1;
x = zeros(size(t_start:h:t_end));
x(i) = x0;

for t = t_start:h:t_end
    % FE
    dx_dt(i) = A*x(i);
    x(i+1) = x(i) + dx_dt(i)*h;

    % for Plotting
    t_vec(i) = t;
    i = i+1;
end
```

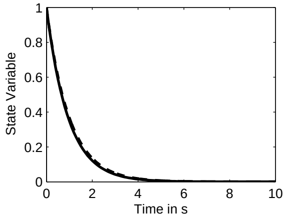
Result of the Forward Euler M-Code



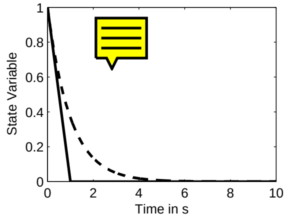
(a) h = 1ms



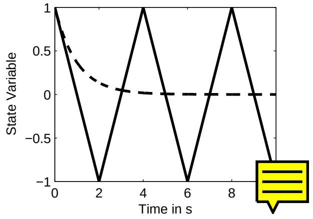
(b) h = 10ms



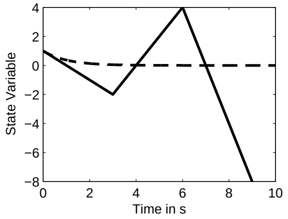
(c) h = 100ms



(d) h = 1000ms



(e) h = 2000ms



(f) h = 3000ms

This was the First Integration Algorithm

We found a very popular although simple integration algorithm. Let us now search for alternatives.

Backward Euler

Looking at the formula there is just a minor difference between Forward and Backward Euler. It is "just" the point in time when the derivative is computed.

$$x(t+h) \approx x(t) + \dot{x}(t+h) \cdot h \tag{30}$$

$$x(t+h) \approx x(t) + f(x(t+h), t+h) \cdot h \tag{31}$$

$$x(t+h) \approx x(t) + A \cdot x(t+h) \cdot h \tag{32}$$

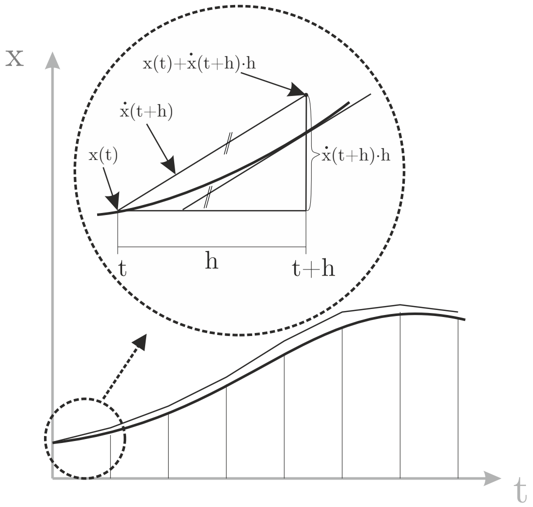
$$x(t+h) \approx (\mathbb{1} - A \cdot h)^{-1} \cdot x(t) \tag{33}$$

This last step is just possible if the system is a linear time-invariant system.

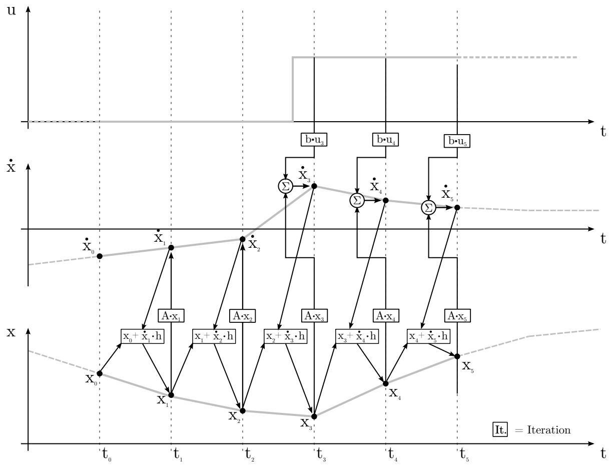
The Main Difference

Still this has major influence on the properties of the solving algorithm.

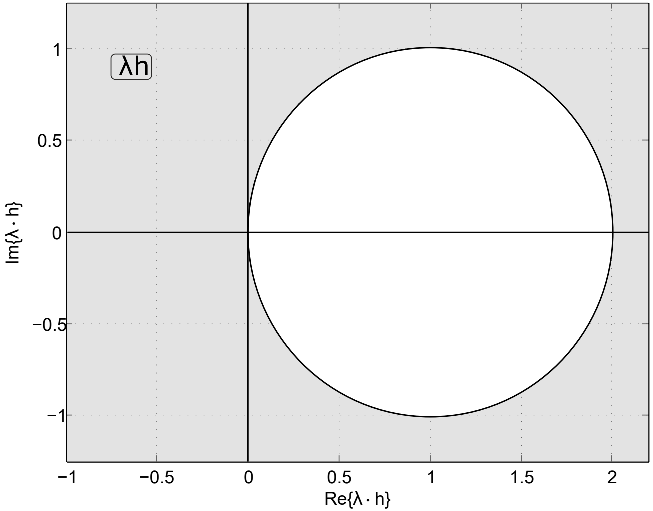
A Graphical Description



Simulating a Linear System with Backward Euler



Backward Euler Stability

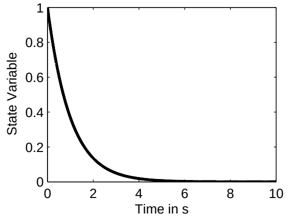


Backward Euler Example Code

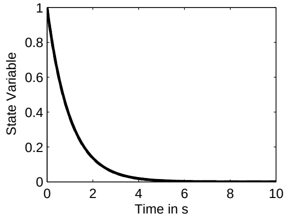
This is the example code for the linear case.

```
for t = t_start:h:t_end
    % BE
    x(i+1) = (1-A*h)^-1 * x(i);

    % for Plotting
    t_vec(i) = t;
    i = i+1;
end
```

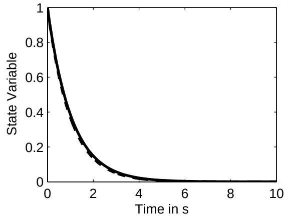
Backward Euler Results



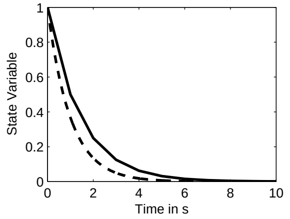
(g) h = 1ms



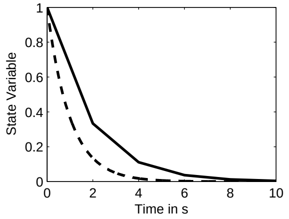
(h) h = 10ms



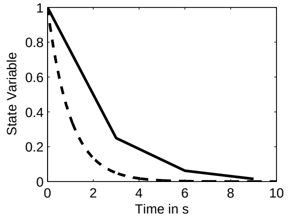
(i) h = 100ms



(j) h = 1000ms



(k) h = 2000ms



(l) h = 3000ms

Implicit vs. Explicit

As mentioned before this simple solution is just possible for linear systems. For non-linear systems there are several techniques to solve the problem faced when using Backward Euler[2].

The Difference

There is a distinction that classifies solvers. If they use future information - and therefore will need to do an iteration to find a solution - they are called **implicit** algorithms. If they don't depend on future data they are **explicit** ones.

The stability domain strongly depends on this property.

[2]Or any other algorithm that uses information from the future.

The Non-Linear Case

For non-linear[3] models the situation gets more complicated in a number of ways.

- The analytical poles move in the λ plane depending on the state of the system.
- This can also result in a system getting numerically unstable as the poles move out of the solvers stability region.
- The formula manipulation done from equation (32) to (33) on slide 35 is not possible anymore.
- For implicit solvers an iteration is necessary to approximate the state variables in the future.

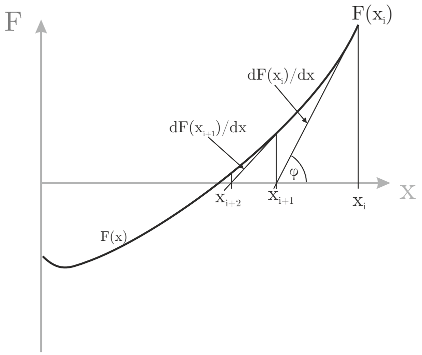
[3] "Non-linear" in this case refers to model equations that can not be solved explicitly by the solver and therefore have to be solved by any kind of iteration. This is a different definition than the one for technical non-linearity. Still we use it here as Dymola uses it this way as well.

Newton Iteration

We take a look at the **Newton Iteration** to better understand what this means.

The Newton Iteration **searches for a zero crossing** of an arbitrary function. If the function doesn't naturally have a zero crossing at the desired position, we have to manipulate it in order to do so.

Newton Iteration



It is not about time

Note that the variables on the x-axis are the states (in our special case) not the time! x_i is the start value for the iteration.

Newton Iteration

From the figure we get

$$\tan(\varphi) = \frac{\partial F^i}{\partial x} = \frac{F^i - 0}{x^i - x^{i+1}} \tag{34}$$

with i being the counting variable of the iteration. Formula manipulation leads to the next value of the iteration.

$$x^{i+1} = x^i - \frac{F^i}{\partial F^i / \partial x} \tag{35}$$

F^i describes the deviation of zero (= distance from the x-axis) of the function at the current iteration value. Therefore, it should get as close to zero as necessary.

$$F \overset{!}{=} 0 \tag{36}$$

In addition to that, we need the derivative $\partial F / \partial x$.

Newton Iteration

We compute F and its derivative by **reformulating the BE** to get an equation that has a zero crossing that the Newton Iteration can search for.

$$x(t+h) = x(t) + h \cdot f(x(t+h)),\ x^i = x(t+h) \tag{37}$$

$$x^i = x(t) + h \cdot f(x^i) \tag{38}$$

$$0 = x(t) + h \cdot f(x^i) - x^i \quad \text{=F} \tag{39}$$

We now have everything needed for the Newton Iteration:

$$F^i = x(t) + h \cdot f(x^i) - x^i \tag{40}$$

$$\frac{\partial F^i}{\partial x} = h \cdot \frac{\partial f(x^i)}{\partial x} - 1 \tag{41}$$

More Newton Iteration

Plugging these results in the equation for x^{i+1} yields the iteration equation. We start with a guess-value (on the x-axis) that is plugged in this equation and execute it in a loop until $|F| \leq \varepsilon$. When this condition is fulfilled, the new state $x(t+h)$ is found.

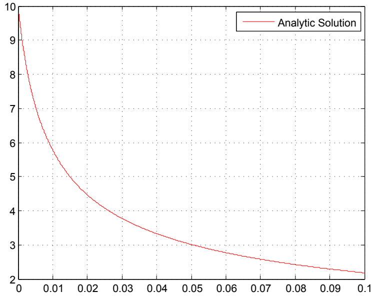
$$x^{i+1} = x^i - \frac{\overbrace{x(t) + h \cdot f(x^i) - x^i}^{F^i}}{\underbrace{h \cdot \partial f(x^i)/\partial x - 1}_{\frac{\partial F^i}{\partial x}}} \tag{42}$$

An Example

Let us apply the Newton iteration to the following non-linear function:

$$f(x,t) = -x^3 \qquad (43)$$

Let us assume an initial value of $x(t=0)=10$.
The analytic solution is shown in the figure below:



Let us further assume that $h = 0.01$, $\varepsilon = 0.01$ and the initial guess value $x^i = 5$.

An Example

Now, we have to insert the values into equation (42).
Let us start with the numerator:

$$F^i = x(t) + h \cdot f(x^i) - x^i$$
$$F^i = 10 + 0.01 \cdot (-5^3) - 5 = 3.75$$

For the denominator we get:

$$\frac{\partial F^i}{\partial x} = h \cdot \frac{\partial f(x^i)}{\partial x} - 1$$
$$\frac{\partial F^i}{\partial x} = 0.01 \cdot (-3 \cdot 5^2) - 1 = -1.75$$

An Example

Hence, our next value for the Newton iteration will be:

$$x^{i+1} = x^i - \frac{F^i}{\frac{\partial F^i}{\partial x}}$$

$$x^{i+1} = 5 - \frac{3.75}{-1.75} = 7.1429$$

This was the first iteration. Continue the iteration until $|F| \leq \varepsilon$.

An Example

Let us apply the Newton iteration once more using $x^i = 7.1429$

$$F^i = x(t) + h \cdot f(x^i) - x^i$$
$$F^i = 10 + 0.01 \cdot (-7.1429^3) - 7.1429 = -0.7873$$

For the denominator we get:

$$\frac{\partial F^i}{\partial x} = h \cdot \frac{\partial f(x^i)}{\partial x} - 1$$
$$\frac{\partial F^i}{\partial x} = 0.01 \cdot (-3 \cdot 7.1429^2) - 1 = -2.5306$$

An Example

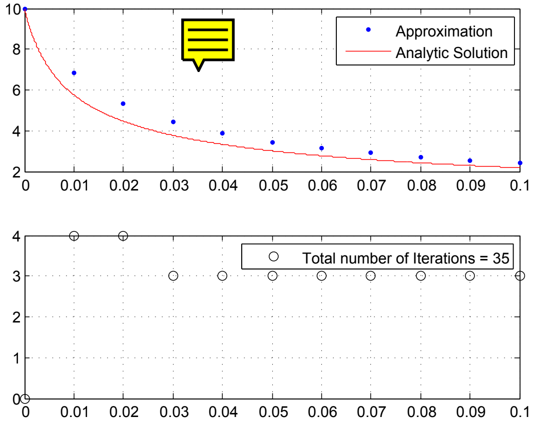
Hence, our next guess value will be:

$$x^{i+1} = x^{i} - \frac{F^{i}}{\frac{\partial F^{i}}{\partial x}}$$
$$x^{i+1} = 7.1429 - \frac{-0.7873}{-2.5306} = 6.8318$$

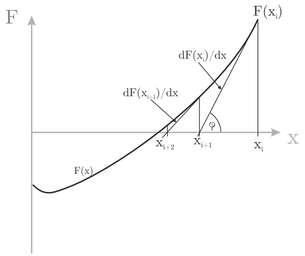
This was the second iteration. Continue the iteration until $|F| \leq \varepsilon$.

An Example

If we simulate the model for $t = 0.1s$ we get the following result:



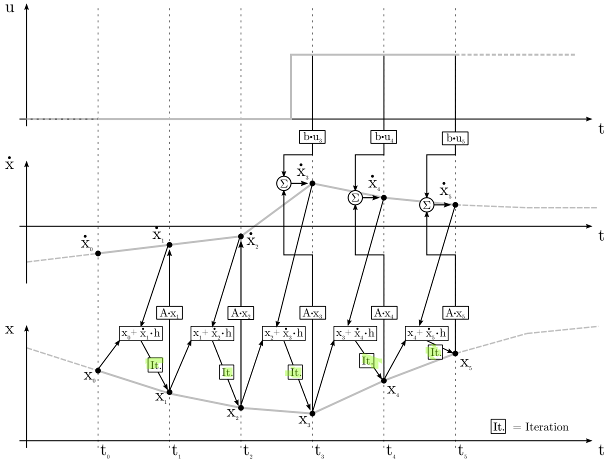
Some Notes regarding the Newton Iteration



If the function is strongly non-linear there is **no guarantee** that we find a zero crossing (or we can as well find a wrong one). This strongly depends on the starting value used for the iteration.

The iteration for usual solvers with a reasonable step-size and tolerance takes about 3-5 times to converge.

Simulating a Non-Linear System with Backward Euler



The Jacobi Matrix

One term that often comes up when talking about simulation topics. The Jacobi matrix is defined as

$$J = \frac{\partial f}{\partial x} = \begin{pmatrix} \partial f_1/\partial x_1 & \partial f_1/\partial x_2 & \cdots & \partial f_1/\partial x_n \\ \partial f_2/\partial x_1 & \partial f_2/\partial x_2 & \cdots & \partial f_2/\partial x_n \\ \vdots & \vdots & \ddots & \vdots \\ \partial f_n/\partial x_1 & \partial f_n/\partial x_2 & \cdots & \partial f_n/\partial x_n \end{pmatrix} \tag{44}$$

For the linear case $f = A \cdot x + B \cdot u$ it simplifies to $J = A$.

For the non-linear case it changes during simulation time and replaces the system matrix A of the state-space description.

The Hessian Matrix

The Hessian Matrix is especially useful for the Newton Iteration. Note that it has F rather than f in its nominator.

$$H = \frac{\partial F}{\partial x} = \begin{pmatrix} \partial F_1/\partial x_1 & \partial F_1/\partial x_2 & \cdots & \partial F_1/\partial x_n \\ \partial F_2/\partial x_1 & \partial F_2/\partial x_2 & \cdots & \partial F_2/\partial x_n \\ \vdots & \vdots & \ddots & \vdots \\ \partial F_n/\partial x_1 & \partial F_n/\partial x_2 & \cdots & \partial F_n/\partial x_n \end{pmatrix} \tag{45}$$

This Hessian Matrix is mainly used in the Newton Iteration. It has nothing to do with modeling directly.

Jacobi, Hessian and Newton Iteration

Using these matrices we can formulate the Newton Iteration in a very compact fashion.

$$
\begin{aligned}
x_{t+h}^{i+1} &= x_{t+h}^{i} - \frac{x_t + h \cdot f(x_{t+h}^i) - x_{t+h}^i)}{h \cdot \underbrace{\partial f(x_{t+h}^i)/\partial x}_{J_{t+h}^i} - 1} = \\
&= x_{t+h}^{i} - \frac{\overbrace{x_t + h \cdot f(x_{t+h}^i) - x_{t+h}^i}^{F^i}}{\underbrace{(h \cdot J_{t+h}^i - 1)}_{H^i = \partial F^i/\partial x}} = \\
&= x_{t+h}^{i} - {H^i}^{-1} \cdot F^i
\end{aligned} \tag{46}
$$

Some M-Code

The corresponding MATLAB code:

```
while (Fi > 1e-5)
{
    Fi = x(t) + h * J*xi - xi
    H = h * J - 1
    xi_1 = xi - H^-1 * F
    xi = xi_1
}
```

Conclusion

- We usually discretize time to create a numerical integrator
- The explicit Forward Euler truncates the Taylor Series after the first term and therefore the next state variables directly by applying a simple formula
- The implicit Backward Euler uses future values to compute the next state variables and therefore has to use (Newton) iteration to find the next state values.
- Discretization changes stability properties compared to their analytical counterparts. The numerical properties can be influenced by the step size.
- Newton iteration does not necessarily converge what will make any solver relying on this iteration fail.

Demonstration

- Simple Pendulum in Dymola
- Oscillating System in Simulink

References I

[1] Francois Cellier and Ernesto Kofman. *Continuous System Simulation*. Springer, 2006.