Symbolical and Numerical Approaches for Solving Nonlinear Systems

Bernhard Bachmann
Kaja Balzereit, Willi Braun, Jan Hagemann,
Lennart Ochel, Vitalij Ruge, Patrick-Marcel Täuber

FH Bielefeld
University of
Applied Sciences

02. February 2015

Contents

Transformation steps for simulation

$$\underline{0} = \underline{f}(\underline{x}(t), \underline{\dot{x}}(t), \underline{y}(t), \underline{u}(t), \underline{p}, t)$$

FH Bielefeld
University of
Applied Sciences

Transformation steps for simulation

$$\underline{0} = \underline{f}(\underline{x}(t), \underline{\dot{x}}(t), \underline{y}(t), \underline{u}(t), \underline{p}, t)$$

$$\Downarrow$$

$$\underline{0} = \underline{f}(\underline{x}(t), \underline{z}(t), \underline{u}(t), \underline{p}, t),\ \underline{z}(t) = \begin{pmatrix} \underline{\dot{x}}(t) \\ \underline{y}(t) \end{pmatrix}$$

FH Bielefeld
University of
Applied Sciences

Transformation steps for simulation

$$\underline{0} = \underline{f}(\underline{x}(t), \underline{\dot{x}}(t), \underline{y}(t), \underline{u}(t), \underline{p}, t)$$

$$\Downarrow$$

$$\underline{0} = \underline{f}(\underline{x}(t), \underline{z}(t), \underline{u}(t), \underline{p}, t),\ \underline{z}(t) = \begin{pmatrix} \underline{\dot{x}}(t) \\ \underline{y}(t) \end{pmatrix}$$

$$\Downarrow$$

$$\underline{z}(t) = \begin{pmatrix} \underline{\dot{x}}(t) \\ \underline{y}(t) \end{pmatrix} = \underline{g}(\underline{x}(t), \underline{u}(t), \underline{p}, t)$$

$$\Downarrow$$

$$\underline{\dot{x}}(t) = \underline{h}(\underline{x}(t), \underline{u}(t), \underline{p}, t)$$

$$\underline{y}(t) = \underline{k}(\underline{x}(t), \underline{u}(t), \underline{p}, t)$$

Transformation example

$$\begin{aligned} f_2(z_2) &= 0 \\ f_4(z_1, z_2) &= 0 \\ f_3(z_2, z_3, z_5) &= 0 \\ f_5(z_1, z_3, z_5) &= 0 \\ f_1(z_3, z_4) &= 0 \end{aligned}$$

Algebraic loop (SCC)

$$\begin{aligned} f_3(z_2, z_3, z_5) &= 0 \\ f_5(z_1, z_3, z_5) &= 0 \end{aligned}$$

Transformation steps for simulation

$$\underline{0} = \underline{f}(\underline{x}(t), \underline{\dot{x}}(t), \underline{y}(t), \underline{u}(t), \underline{p}, t)$$

$$\Downarrow$$

$$\underline{0} = \underline{f}(\underline{x}(t), \underline{z}(t), \underline{u}(t), \underline{p}, t),\ \underline{z}(t) = \begin{pmatrix} \underline{\dot{x}}(t) \\ \underline{y}(t) \end{pmatrix}$$

$$\Downarrow$$

$$\underline{z}(t) = \begin{pmatrix} \underline{\dot{x}}(t) \\ \underline{y}(t) \end{pmatrix} = \underline{g}(\underline{x}(t), \underline{u}(t), \underline{p}, t)$$

$$\Downarrow$$

$$\underline{\dot{x}}(t) = \underline{h}(\underline{x}(t), \underline{u}(t), \underline{p}, t)$$
$$\underline{y}(t) = \underline{k}(\underline{x}(t), \underline{u}(t), \underline{p}, t)$$

Transformation steps for simulation

$$\underline{0} = \underline{f}(\underline{x}(t), \underline{\dot{x}}(t), \underline{y}(t), \underline{u}(t), \underline{p}, t)$$

$$\Downarrow$$

$$\underline{0} = \underline{f}(\underline{x}(t), \underline{z}(t), \underline{u}(t), \underline{p}, t),\ \underline{z}(t) = \begin{pmatrix} \underline{\dot{x}}(t) \\ \underline{y}(t) \end{pmatrix}$$

$$\Downarrow$$

$$\underline{z}(t) = \begin{pmatrix} \underline{\dot{x}}(t) \\ \underline{y}(t) \end{pmatrix} = \underline{g}(\underline{x}(t), \underline{u}(t), \underline{p}, t)$$

$$\Downarrow$$

$$\begin{aligned} \underline{\dot{x}}(t) &= \underline{h}(\underline{x}(t), \underline{u}(t), \underline{p}, t) \\ \underline{y}(t) &= \underline{k}(\underline{x}(t), \underline{u}(t), \underline{p}, t) \end{aligned}$$

Transformation example

$$\begin{aligned} f_2(z_2) &= 0 \\ f_4(z_1, z_2) &= 0 \\ f_3(z_2, z_3, z_5) &= 0 \\ f_5(z_1, z_3, z_5) &= 0 \\ f_1(z_3, z_4) &= 0 \end{aligned}$$

Algebraic loop (SCC)

$$\begin{aligned} f_3(z_2, z_3, z_5) &= 0 \\ f_5(z_1, z_3, z_5) &= 0 \end{aligned}$$

Transformation steps for simulation

$$\underline{0} = \underline{f}(\underline{x}(t), \underline{\dot{x}}(t), \underline{y}(t), \underline{u}(t), \underline{p}, t)$$

$$\Downarrow$$

$$\underline{0} = \underline{f}(\underline{x}(t), \underline{z}(t), \underline{u}(t), \underline{p}, t),\ \underline{z}(t) = \begin{pmatrix} \underline{\dot{x}}(t) \\ \underline{y}(t) \end{pmatrix}$$

$$\Downarrow$$

$$\underline{z}(t) = \begin{pmatrix} \underline{\dot{x}}(t) \\ \underline{y}(t) \end{pmatrix} = \underline{g}(\underline{x}(t), \underline{u}(t), \underline{p}, t)$$

$$\Downarrow$$

$$\begin{aligned} \underline{\dot{x}}(t) &= \underline{h}(\underline{x}(t), \underline{u}(t), \underline{p}, t) \\ \underline{y}(t) &= \underline{k}(\underline{x}(t), \underline{u}(t), \underline{p}, t) \end{aligned}$$

Transformation example

$$\begin{aligned} f_2(z_2) &= 0 \\ f_4(z_1, z_2) &= 0 \\ f_3(z_2, z_3, z_5) &= 0 \\ f_5(z_1, z_3, z_5) &= 0 \\ f_1(z_3, z_4) &= 0 \end{aligned}$$

Algebraic loop (SCC)

$$\begin{aligned} f_3(z_2, z_3, z_5) &= 0 \\ f_5(z_1, z_3, z_5) &= 0 \end{aligned}$$

Algebraic loops in OpenModelica:

- Generated code (functionODE, functionAlgebraics, initialization, etc.)
- Efficient handling of (non-)linear equation(s)
- Proper scaling of iteration variables and equations
- Iteration schemes need good starting values
- Handling of non-convergence
- Dealing with non-valid values of iteration variables during solution process

Transformation example

$$
\begin{aligned}
f_2(z_2) &= 0\\
f_4(z_1, z_2) &= 0\\
f_3(z_2, z_3, z_5) &= 0\\
f_5(z_1, z_3, z_5) &= 0\\
f_1(z_3, z_4) &= 0
\end{aligned}
$$

Algebraic loop (SCC)

$$
\begin{aligned}
f_3(z_2, z_3, z_5) &= 0\\
f_5(z_1, z_3, z_5) &= 0
\end{aligned}
$$

Introduction

Important Aspects with Algebraic Loops!

Algebraic loops in OpenModelica:

- Generated code (functionODE, functionAlgebraics, initialization, etc.)
- Efficient handling of (non-)linear equation(s)
  - Solution of single equations, Tearing of equation systems, Good solver implementation, Generation of symbolic Jacobians, etc.
- Proper scaling of iteration variables and equations
- Iteration schemes need good starting values
- Handling of non-convergence
- Dealing with non-valid values of iteration variables during solution process

Transformation example

$$
\begin{aligned}
f_2(z_2) &= 0 \\
f_4(z_1, z_2) &= 0 \\
f_3(z_2, z_3, z_5) &= 0 \\
f_5(z_1, z_3, z_5) &= 0 \\
f_1(z_3, z_4) &= 0
\end{aligned}
$$

Algebraic loop (SCC)

$$
\begin{aligned}
f_3(z_2, z_3, z_5) &= 0 \\
f_5(z_1, z_3, z_5) &= 0
\end{aligned}
$$

Introduction

Important Aspects with Algebraic Loops!

Algebraic loops in OpenModelica:

- Generated code (functionODE, functionAlgebraics, initialization, etc.)
- Efficient handling of (non-)linear equation(s)
  - Solution of single equations, Tearing of equation systems, Good solver implementation, Generation of symbolic Jacobians, etc.
- Proper scaling of iteration variables and equations
- Iteration schemes need good starting values
- Handling of non-convergence
- Dealing with non-valid values of iteration variables during solution process

Transformation example

$$
\begin{aligned}
f_2(z_2) &= 0\\
f_4(z_1, z_2) &= 0\\
f_3(z_2, z_3, z_5) &= 0\\
f_5(z_1, z_3, z_5) &= 0\\
f_1(z_3, z_4) &= 0
\end{aligned}
$$

Algebraic loop (SCC)

$$
\begin{aligned}
f_3(z_2, z_3, z_5) &= 0\\
f_5(z_1, z_3, z_5) &= 0
\end{aligned}
$$

Introduction

Important Aspects with Algebraic Loops!

FH Bielefeld
University of
Applied Sciences

Algebraic loops in OpenModelica:

- Generated code (functionODE, functionAlgebraics, initialization, etc.)
- Efficient handling of (non-)linear equation(s)
  - Solution of single equations, Tearing of equation systems, Good solver implementation, Generation of symbolic Jacobians, etc.
- Proper scaling of iteration variables and equations
- Iteration schemes need good starting values
- Handling of non-convergence
- Dealing with non-valid values of iteration variables during solution process

Transformation example

$$
\begin{aligned}
f_2(z_2) &= 0\\
f_4(z_1, z_2) &= 0\\
f_3(z_2, z_3, z_5) &= 0\\
f_5(z_1, z_3, z_5) &= 0\\
f_1(z_3, z_4) &= 0
\end{aligned}
$$

Algebraic loop (SCC)

$$
\begin{aligned}
f_3(z_2, z_3, z_5) &= 0\\
f_5(z_1, z_3, z_5) &= 0
\end{aligned}
$$

Introduction

Important Aspects with Algebraic Loops!

FH Bielefeld
University of
Applied Sciences

Algebraic loops in OpenModelica:

- Generated code (functionODE, functionAlgebraics, initialization, etc.)
- Efficient handling of (non-)linear equation(s)
  - Solution of single equations, Tearing of equation systems, Good solver implementation, Generation of symbolic Jacobians, etc.
- Proper scaling of iteration variables and equations
- Iteration schemes need good starting values
- Handling of non-convergence
- Dealing with non-valid values of iteration variables during solution process

Transformation example

$$
\begin{aligned}
f_2(z_2) &= 0\\
f_4(z_1, z_2) &= 0\\
f_3(z_2, z_3, z_5) &= 0\\
f_5(z_1, z_3, z_5) &= 0\\
f_1(z_3, z_4) &= 0
\end{aligned}
$$

Algebraic loop (SCC)

$$
\begin{aligned}
f_3(z_2, z_3, z_5) &= 0\\
f_5(z_1, z_3, z_5) &= 0
\end{aligned}
$$

Introduction

Important Aspects with Algebraic Loops!

Algebraic loops in OpenModelica:

- Generated code (functionODE, functionAlgebraics, initialization, etc.)
- Efficient handling of (non-)linear equation(s)
  - Solution of single equations, Tearing of equation systems, Good solver implementation, Generation of symbolic Jacobians, etc.
- Proper scaling of iteration variables and equations
- Iteration schemes need good starting values
- Handling of non-convergence
  - Different heuristics possible (varying starting values, nominal values, accuracy, etc.)
- Dealing with non-valid values of iteration variables during solution process

Transformation example

$$
\begin{aligned}
f_2(z_2) &= 0\\
f_4(z_1, z_2) &= 0\\
f_3(z_2, z_3, z_5) &= 0\\
f_5(z_1, z_3, z_5) &= 0\\
f_1(z_3, z_4) &= 0
\end{aligned}
$$

Algebraic loop (SCC)

$$
\begin{aligned}
f_3(z_2, z_3, z_5) &= 0\\
f_5(z_1, z_3, z_5) &= 0
\end{aligned}
$$

Introduction

Important Aspects with Algebraic Loops!

Algebraic loops in OpenModelica:

- Generated code (functionODE, functionAlgebraics, initialization, etc.)
- Efficient handling of (non-)linear equation(s)
  - Solution of single equations, Tearing of equation systems, Good solver implementation, Generation of symbolic Jacobians, etc.
- Proper scaling of iteration variables and equations
- Iteration schemes need good starting values
- Handling of non-convergence
  - Different heuristics possible (varying starting values, nominal values, accuracy, etc.)
- Dealing with non-valid values of iteration variables during solution process

Transformation example

$$
\begin{aligned}
f_2(z_2) &= 0 \\
f_4(z_1, z_2) &= 0 \\
f_3(z_2, z_3, z_5) &= 0 \\
f_5(z_1, z_3, z_5) &= 0 \\
f_1(z_3, z_4) &= 0
\end{aligned}
$$

Algebraic loop (SCC)

$$
\begin{aligned}
f_3(z_2, z_3, z_5) &= 0 \\
f_5(z_1, z_3, z_5) &= 0
\end{aligned}
$$

Algebraic loops in OpenModelica:

- Generated code (functionODE, functionAlgebraics, initialization, etc.)
- Efficient handling of (non-)linear equation(s)
  - Solution of single equations, Tearing of equation systems, Good solver implementation, Generation of symbolic Jacobians, etc.
- Proper scaling of iteration variables and equations
- Iteration schemes need good starting values
- Handling of non-convergence
  - Different heuristics possible (varying starting values, nominal values, accuracy, etc.)
- Dealing with non-valid values of iteration variables during solution process

Transformation example

$$
\begin{aligned}
f_2(z_2) &= 0\\
f_4(z_1, z_2) &= 0\\
f_3(z_2, z_3, z_5) &= 0\\
f_5(z_1, z_3, z_5) &= 0\\
f_1(z_3, z_4) &= 0
\end{aligned}
$$

Algebraic loop (SCC)

$$
\begin{aligned}
f_3(z_2, z_3, z_5) &= 0\\
f_5(z_1, z_3, z_5) &= 0
\end{aligned}
$$

Default mixed-solver strategy:

- Determine proper starting values
  - Check validity of starting values with respect to regular Jacobian and asserts of function calls
- Start damped Newton algorithm
- 1^{st} Fallback case: newly developed Homotopy solver
- 2^{nd} Fallback case: hybrid solver (`-nls hybrid`)

Nonlinear problem

$$\underline{F}(\underline{x}) = \underline{0}$$

Start vector: $\underline{x}_0 \in \mathbb{R}^n$.

Newton iteration step:

$$\begin{aligned} J_{\underline{F}}\left(\underline{x}^{(k)}\right) \cdot \underline{s}^{(k)} &= -\underline{F}\left(\underline{x}^{(k)}\right) \\ \underline{x}^{(k+1)} &= \underline{x}^{(k)} + \tau^{(k)} \underline{s}^{(k)} \end{aligned}$$

Damping parameter:

$$\tau^{(k)} \in [0, 1].$$

FH Bielefeld
University of
Applied Sciences

Default mixed-solver strategy:

- Determine proper starting values
  - Check validity of starting values with respect to regular Jacobian and asserts of function calls
- Start damped Newton algorithm
- 1^{st} Fallback case: newly developed Homotopy solver
- 2^{nd} Fallback case: hybrid solver (`-nls hybrid`)

Nonlinear problem

$$\underline{F}(\underline{x}) = \underline{0}$$

Start vector: $\underline{x}_0 \in \mathbb{R}^n$.

Newton iteration step:

$$\begin{aligned} J_{\underline{F}}\left(\underline{x}^{(k)}\right) \cdot \underline{s}^{(k)} &= -\underline{F}\left(\underline{x}^{(k)}\right) \\ \underline{x}^{(k+1)} &= \underline{x}^{(k)} + \tau^{(k)}\underline{s}^{(k)} \end{aligned}$$

Damping parameter:

$$\tau^{(k)} \in [0, 1].$$

Default mixed-solver strategy:

- Determine proper starting values
  - Check validity of starting values with respect to regular Jacobian and asserts of function calls
- Start damped Newton algorithm
  - Damping strategy based on expected function decrease and validity of iteration step
- 1^{st} Fallback case: newly developed Homotopy solver
- 2^{nd} Fallback case: hybrid solver (`-nls hybrid`)

Nonlinear problem

$$\underline{F}(\underline{x}) = \underline{0}$$

Start vector: $\underline{x}_0 \in \mathbb{R}^n$.

Newton iteration step:

$$\begin{aligned} J_{\underline{F}}\left(\underline{x}^{(k)}\right) \cdot \underline{s}^{(k)} &= -\underline{F}\left(\underline{x}^{(k)}\right) \\ \underline{x}^{(k+1)} &= \underline{x}^{(k)} + \tau^{(k)} \underline{s}^{(k)} \end{aligned}$$

Damping parameter:

$$\tau^{(k)} \in [0, 1].$$

Introduction

Current Status in OpenModelica

FH Bielefeld
University of
Applied Sciences

Default mixed-solver strategy:

- Determine proper starting values
  - Check validity of starting values with respect to regular Jacobian and asserts of function calls
- Start damped Newton algorithm
  - Damping strategy based on expected function decrease and validity of iteration step
- 1^{st} Fallback case: newly developed Homotopy solver
- 2^{nd} Fallback case: hybrid solver (`-nls hybrid`)

Nonlinear problem

$$\underline{F}(\underline{x}) = \underline{0}$$

Start vector: $\underline{x}_0 \in \mathbb{R}^n$.

Newton iteration step:

$$\begin{aligned} J_{\underline{F}}\left(\underline{x}^{(k)}\right) \cdot \underline{s}^{(k)} &= -\underline{F}\left(\underline{x}^{(k)}\right) \\ \underline{x}^{(k+1)} &= \underline{x}^{(k)} + \tau^{(k)} \underline{s}^{(k)} \end{aligned}$$

Damping parameter:

$$\tau^{(k)} \in [0, 1].$$

Introduction

Current Status in OpenModelica

Default mixed-solver strategy:

- Determine proper starting values
  - Check validity of starting values with respect to regular Jacobian and asserts of function calls
- Start damped Newton algorithm
  - Damping strategy based on expected function decrease and validity of iteration step
- 1^{st} Fallback case: newly developed Homotopy solver
  - Already robust prototype
- 2^{nd} Fallback case: hybrid solver (`-nls hybrid`)

Nonlinear problem

$$\underline{F}(\underline{x}) = \underline{0}$$

Start vector: $\underline{x}_0 \in \mathbb{R}^n$.

Newton iteration step:

$$\begin{aligned} J_{\underline{F}}\left(\underline{x}^{(k)}\right) \cdot \underline{s}^{(k)} &= -\underline{F}\left(\underline{x}^{(k)}\right) \\ \underline{x}^{(k+1)} &= \underline{x}^{(k)} + \tau^{(k)} \underline{s}^{(k)} \end{aligned}$$

Damping parameter:

$$\tau^{(k)} \in [0, 1].$$

Introduction

Current Status in OpenModelica

FH Bielefeld
University of
Applied Sciences

Default mixed-solver strategy:

- Determine proper starting values
  - Check validity of starting values with respect to regular Jacobian and asserts of function calls
- Start damped Newton algorithm
  - Damping strategy based on expected function decrease and validity of iteration step
- 1^{st} Fallback case: newly developed Homotopy solver
  - Already robust prototype
- 2^{nd} Fallback case: hybrid solver (-nls hybrid)

Nonlinear problem

$$\underline{F}(\underline{x}) = \underline{0}$$

Start vector: $\underline{x}_0 \in \mathbb{R}^n$.

Newton iteration step:

$$\begin{aligned} J_{\underline{F}}\left(\underline{x}^{(k)}\right) \cdot \underline{s}^{(k)} &= -\underline{F}\left(\underline{x}^{(k)}\right) \\ \underline{x}^{(k+1)} &= \underline{x}^{(k)} + \tau^{(k)} \underline{s}^{(k)} \end{aligned}$$

Damping parameter:

$$\tau^{(k)} \in [0, 1].$$

Introduction

Current Status in OpenModelica

FH Bielefeld
University of
Applied Sciences

Default mixed-solver strategy:

- Determine proper starting values
  - Check validity of starting values with respect to regular Jacobian and asserts of function calls
- Start damped Newton algorithm
  - Damping strategy based on expected function decrease and validity of iteration step
- 1^{st} Fallback case: newly developed Homotopy solver
  - Already robust prototype
- 2^{nd} Fallback case: hybrid solver (`-nls hybrid`)
  - Robust solver including several additional heuristics

Nonlinear problem

$$\underline{F}(\underline{x}) = \underline{0}$$

Start vector: $\underline{x}_0 \in \mathbb{R}^n$.

Newton iteration step:

$$\begin{aligned} J_{\underline{F}}\left(\underline{x}^{(k)}\right) \cdot \underline{s}^{(k)} &= -\underline{F}\left(\underline{x}^{(k)}\right) \\ \underline{x}^{(k+1)} &= \underline{x}^{(k)} + \tau^{(k)} \underline{s}^{(k)} \end{aligned}$$

Damping parameter:

$$\tau^{(k)} \in [0, 1].$$

Default mixed-solver strategy:

- Determine proper starting values
  - Check validity of starting values with respect to regular Jacobian and asserts of function calls
- Start damped Newton algorithm
  - Damping strategy based on expected function decrease and validity of iteration step
- 1^{st} Fallback case: newly developed Homotopy solver
  - Already robust prototype
- 2^{nd} Fallback case: hybrid solver (`-nls hybrid`)
  - Robust solver including several additional heuristics

Nonlinear problem

$$\underline{F}(\underline{x}) = \underline{0}$$

Start vector: $\underline{x}_0 \in \mathbb{R}^n$.

Newton iteration step:

$$\begin{aligned} J_{\underline{F}}\left(\underline{x}^{(k)}\right) \cdot \underline{s}^{(k)} &= -\underline{F}\left(\underline{x}^{(k)}\right) \\ \underline{x}^{(k+1)} &= \underline{x}^{(k)} + \tau^{(k)}\underline{s}^{(k)} \end{aligned}$$

Damping parameter:

$$\tau^{(k)} \in [0, 1].$$

Homotopy Method

General Approach

FH Bielefeld
University of
Applied Sciences

Nonlinear problem

Solve non-linear equation system

$$\underline{F}(\underline{x}^*) = \underline{0}$$

with given start vector $\underline{x}_0 \in \mathbb{R}^n$.

Possible homotopy functions

- Fixpoint-Homotopy:

$$\underline{H}(\underline{x}, \lambda) = \lambda \underline{F}(\underline{x}) + (1-\lambda)(\underline{x} - \underline{x}_0) = \underline{0}$$

- Newton-Homotopy:

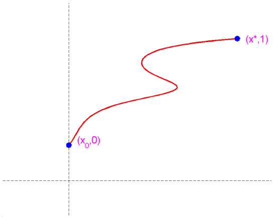
$$\underline{H}(\underline{x}, \lambda) = \underline{F}(\underline{x}) - (1-\lambda)\underline{F}(\underline{x}_0) = \underline{0}$$

Simple example

$$f(x) = 2x - 4 + sin(2\pi x),$$

$$x_0 = 0.5, \qquad x^* = 2.$$

Homotopy Path (Fixpoint)



FH Bielefeld
University of
Applied Sciences

Nonlinear problem

Solve non-linear equation system

$$\underline{F}(\underline{x}^*) = \underline{0}$$

with given start vector $\underline{x}_0 \in \mathbb{R}^n$.

Possible homotopy functions

- Fixpoint-Homotopy:

$$\underline{H}(\underline{x}, \lambda) = \lambda \underline{F}(\underline{x}) + (1-\lambda)(\underline{x} - \underline{x}_0) = \underline{0}$$

- Newton-Homotopy:

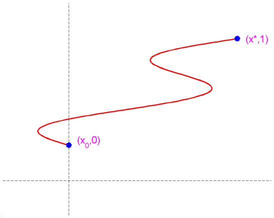
$$\underline{H}(\underline{x}, \lambda) = \underline{F}(\underline{x}) - (1-\lambda)\underline{F}(\underline{x}_0) = \underline{0}$$

Simple example

$$f(x) = 2x - 4 + sin(2\pi x),$$

$$x_0 = 0.5, \quad x^* = 2.$$

Homotopy Path (Newton)



Homotopy Method

General Approach

FH Bielefeld
University of
Applied Sciences

Homotopy-iteration

Start with $(x_0, \underbrace{\lambda_0}_{=0})$ and $\underline{H}(\underline{x}_0, \lambda_0) = \underline{0}$.

Determine $(\underline{x}_{i+1}, \lambda_{i+1})$ with $\underline{H}(\underline{x}_{i+1}, \lambda_{i+1}) = \underline{0}$.

Stop, when $\lambda_m = 1$ yields:

$$\Rightarrow \underline{H}(\underline{x}_m, \underbrace{\lambda_m}_{=1}) = \underline{F}(\underline{x}_m) = \underline{0} \Rightarrow \underline{x}^* = \underline{x}_m.$$

Procedure:
Perform predictor-corrector steps

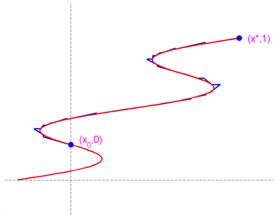
$\Rightarrow$ path $(\underline{x}(s), \lambda(s))$, s arc length.

Simple example

$$f(x) = 2x - 4 + sin(2\pi x),$$

$$x_0 = 0.5, \quad x^* = 2.$$

Homotopy Path (Iteration)



FH Bielefeld
University of
Applied Sciences

Homotopy-iteration

Start with $(x_0, \underbrace{\lambda_0}_{=0})$ and $\underline{H}(\underline{x}_0, \lambda_0) = \underline{0}$.

Determine $(\underline{x}_{i+1}, \lambda_{i+1})$ with $\underline{H}(\underline{x}_{i+1}, \lambda_{i+1}) = \underline{0}$.

Stop, when $\lambda_m = 1$ yields.

$$\Rightarrow \underline{H}(\underline{x}_m, \underbrace{\lambda_m}_{=1}) = \underline{F}(\underline{x}_m) = \underline{0} \Rightarrow \underline{x}^* = \underline{x}_m.$$

Procedure:
Perform predictor-corrector steps

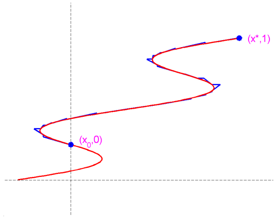
$$\Rightarrow \text{ path } (\underline{x}(s), \lambda(s)), s \text{ arc length.}$$

Simple example

$$f(x) = 2x - 4 + sin(2\pi x),$$

$$x_0 = 0.5, \quad x^* = 2.$$

Homotopy Path (Iteration)



Homotopy Method

General Approach

Homotopy-iteration

Start with $(x_0, \underbrace{\lambda_0}_{=0})$ and $\underline{H}(\underline{x}_0, \lambda_0) = \underline{0}$.

Determine $(\underline{x}_{i+1}, \lambda_{i+1})$ with $\underline{H}(\underline{x}_{i+1}, \lambda_{i+1}) = \underline{0}$.

Stop, when $\lambda_m = 1$ yields.

$$\Rightarrow \underline{H}(\underline{x}_m, \underbrace{\lambda_m}_{=1}) = \underline{F}(\underline{x}_m) = \underline{0} \Rightarrow \underline{x}^* = \underline{x}_m.$$

Procedure:
Perform predictor-corrector steps

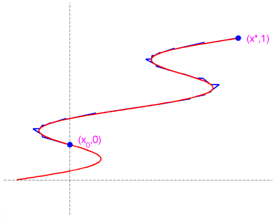
$\Rightarrow$ path $(\underline{x}(s), \lambda(s))$, s arc length.

Simple example

$$f(x) = 2x - 4 + sin(2\pi x),$$

$$x_0 = 0.5, \quad x^* = 2.$$

Homotopy Path (Iteration)



Homotopy-iteration

Start with $(x_0, \underbrace{\lambda_0}_{=0})$ and $\underline{H}(\underline{x}_0, \lambda_0) = \underline{0}$.

Determine $(\underline{x}_{i+1}, \lambda_{i+1})$ with $\underline{H}(\underline{x}_{i+1}, \lambda_{i+1}) = \underline{0}$.

Stop, when $\lambda_m = 1$ yields.

$$\Rightarrow \underline{H}(\underline{x}_m, \underbrace{\lambda_m}_{=1}) = \underline{F}(\underline{x}_m) = \underline{0} \Rightarrow \underline{x}^* = \underline{x}_m.$$

Procedure:
Perform predictor-corrector steps

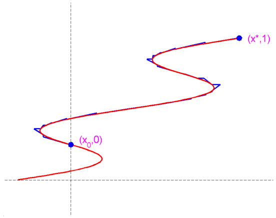
$\Rightarrow$ path $(\underline{x}(s), \lambda(s))$, s arc length.

Simple example

$$f(x) = 2x - 4 + sin(2\pi x),$$

$$x_0 = 0.5, \quad x^* = 2.$$

Homotopy Path (Iteration)



Predictor step

$$\underline{H}(\underline{x}(s), \lambda(s)) = \underline{0}$$

$$\Rightarrow \frac{\partial \underline{H}}{\partial \underline{x}} \cdot \underline{x}'(s) + \frac{\partial \underline{H}}{\partial \lambda} \cdot \lambda'(s) = \underline{0}.$$

Solve linear system

$$\underline{J}_{\underline{H}}(\underline{x}_i, \lambda_i) \cdot \underline{v}_i = \underline{0},$$

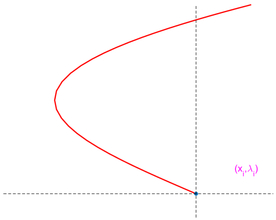
$\underline{J}_{\underline{H}} \in \mathbb{R}^{(n,n+1)}$, Jacobian matrix of $\underline{H}(\underline{x}, \lambda)$.

Perform predictor step

$$\begin{pmatrix} \underline{x}^{\#}_{i+1} \\ \lambda^{\#}_{i+1} \end{pmatrix} = \begin{pmatrix} \underline{x}_i \\ \lambda_i \end{pmatrix} + \tau_i \cdot \underline{v}_i$$

τ_i step size, $\underline{v}_i$ normalized direction.

Homotopy Path Calculation



Corrector step

Fix one coordinate, run Newton iteration steps with start values $(\underline{x}^{\#}_{i+1}, \lambda^{\#}_{i+1})$ until

$$\underline{H}(\underline{x}_{i+1}, \lambda_{i+1}) \approx \underline{0}.$$

Predictor step

$$\underline{H}(\underline{x}(s), \lambda(s)) = \underline{0}$$

$$\Rightarrow \frac{\partial \underline{H}}{\partial \underline{x}} \cdot \underline{x}'(s) + \frac{\partial \underline{H}}{\partial \lambda} \cdot \lambda'(s) = \underline{0}.$$

Solve linear system

$$J_{\underline{H}}(\underline{x}_i, \lambda_i) \cdot \underline{v}_i = \underline{0},$$

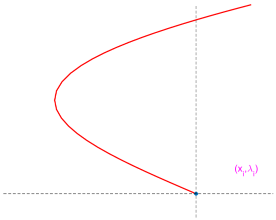
$J_{\underline{H}} \in \mathbb{R}^{(n,n+1)}$, Jacobian matrix of $\underline{H}(\underline{x}, \lambda)$.

Perform predictor step

$$\begin{pmatrix} \underline{x}_{i+1}^{\#} \\ \lambda_{i+1}^{\#} \end{pmatrix} = \begin{pmatrix} \underline{x}_i \\ \lambda_i \end{pmatrix} + \tau_i \cdot \underline{v}_i;$$

τ_i step size, $\underline{v}_i$ normalized direction.

Homotopy Path Calculation



Corrector step

Fix one coordinate, run Newton iteration steps with start values $(\underline{x}_{i+1}^{\#}, \lambda_{i+1}^{\#})$ until

$$\underline{H}(\underline{x}_{i+1}, \lambda_{i+1}) \approx \underline{0}.$$

Predictor step

$$\underline{H}(\underline{x}(s), \lambda(s)) = \underline{0}$$

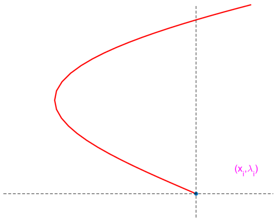
$$\Rightarrow \frac{\partial \underline{H}}{\partial \underline{x}} \cdot \underline{x}'(s) + \frac{\partial \underline{H}}{\partial \lambda} \cdot \lambda'(s) = \underline{0}.$$

Solve linear system

$$J_{\underline{H}}(\underline{x}_i, \lambda_i) \cdot \underline{v}_i = \underline{0},$$

$J_{\underline{H}} \in \mathbb{R}^{(n,n+1)}$, Jacobian matrix of $\underline{H}(\underline{x}, \lambda)$.

Homotopy Path Calculation



Predictor step

$$\underline{H}(\underline{x}(s), \lambda(s)) = \underline{0}$$

$$\Rightarrow \frac{\partial \underline{H}}{\partial \underline{x}} \cdot \underline{x}'(s) + \frac{\partial \underline{H}}{\partial \lambda} \cdot \lambda'(s) = \underline{0}.$$

Solve linear system

$$J_{\underline{H}}(\underline{x}_i, \lambda_i) \cdot \underline{v}_i = \underline{0},$$

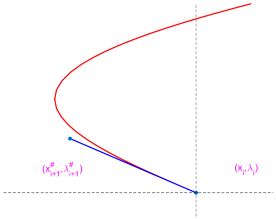
$J_{\underline{H}} \in \mathbb{R}^{(n,n+1)}$, Jacobian matrix of $\underline{H}(\underline{x}, \lambda)$.

Perform predictor step

$$\begin{pmatrix} \underline{x}_{i+1}^{\#} \\ \lambda_{i+1}^{\#} \end{pmatrix} = \begin{pmatrix} \underline{x}_i \\ \lambda_i \end{pmatrix} + \tau_i \cdot \underline{v}_i,$$

τ_i step size, $\underline{v}_i$ normalized direction.

Homotopy Path Calculation



Corrector step

Fix one coordinate, run Newton iteration steps with start values $(\underline{x}_{i+1}^{\#}, \lambda_{i+1}^{\#})$ until

$$\underline{H}(\underline{x}_{i+1}, \lambda_{i+1}) \approx \underline{0}.$$

Homotopy Method

Calculating Homotopy Path $(\underline{x}(s), \lambda(s))$

Predictor step

$$\underline{H}(\underline{x}(s), \lambda(s)) = \underline{0}$$

$$\Rightarrow \frac{\partial \underline{H}}{\partial \underline{x}} \cdot \underline{x}'(s) + \frac{\partial \underline{H}}{\partial \lambda} \cdot \lambda'(s) = \underline{0}.$$

Solve linear system

$$J_{\underline{H}}(\underline{x}_i, \lambda_i) \cdot \underline{v}_i = \underline{0},$$

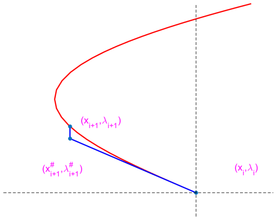
$J_{\underline{H}} \in \mathbb{R}^{(n,n+1)}$, Jacobian matrix of $\underline{H}(\underline{x}, \lambda)$.

Perform predictor step

$$\begin{pmatrix} \underline{x}^{\#}_{i+1} \\ \lambda^{\#}_{i+1} \end{pmatrix} = \begin{pmatrix} \underline{x}_i \\ \lambda_i \end{pmatrix} + \tau_i \cdot \underline{v}_i,$$

τ_i step size, $\underline{v}_i$ normalized direction.

Homotopy Path Calculation



Corrector step

Fix one coordinate, run Newton iteration steps with start values $(\underline{x}^{\#}_{i+1}, \lambda^{\#}_{i+1})$ until

$$\underline{H}(\underline{x}_{i+1}, \lambda_{i+1}) \approx \underline{0}.$$

Homotopy Method

Calculating Homotopy Path $(\underline{x}(s), \lambda(s))$

Predictor step

$$\underline{H}(\underline{x}(s), \lambda(s)) = \underline{0}$$

$$\Rightarrow \frac{\partial \underline{H}}{\partial \underline{x}} \cdot \underline{x}'(s) + \frac{\partial \underline{H}}{\partial \lambda} \cdot \lambda'(s) = \underline{0}.$$

Solve linear system

$$J_{\underline{H}}(\underline{x}_i, \lambda_i) \cdot \underline{v}_i = \underline{0},$$

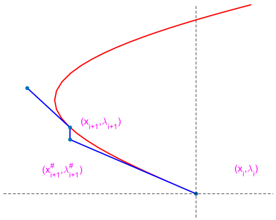
$J_{\underline{H}} \in \mathbb{R}^{(n,n+1)}$, Jacobian matrix of $\underline{H}(\underline{x}, \lambda)$.

Perform predictor step

$$\begin{pmatrix} \underline{x}^{\#}_{i+1} \\ \lambda^{\#}_{i+1} \end{pmatrix} = \begin{pmatrix} \underline{x}_i \\ \lambda_i \end{pmatrix} + \tau_i \cdot \underline{v}_i,$$

τ_i step size, $\underline{v}_i$ normalized direction.

Homotopy Path Calculation



Corrector step

Fix one coordinate, run Newton iteration steps with start values $(\underline{x}^{\#}_{i+1}, \lambda^{\#}_{i+1})$ until

$$\underline{H}(\underline{x}_{i+1}, \lambda_{i+1}) \approx \underline{0}.$$

FH Bielefeld
University of
Applied Sciences

Predictor step

$$\underline{H}(\underline{x}(s), \lambda(s)) = \underline{0}$$

$$\Rightarrow \frac{\partial \underline{H}}{\partial \underline{x}} \cdot \underline{x}'(s) + \frac{\partial \underline{H}}{\partial \lambda} \cdot \lambda'(s) = \underline{0}.$$

Solve linear system

$$J_{\underline{H}}(\underline{x}_i, \lambda_i) \cdot \underline{v}_i = \underline{0},$$

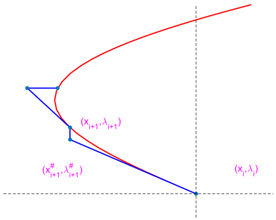
$J_{\underline{H}} \in \mathbb{R}^{(n,n+1)}$, Jacobian matrix of $\underline{H}(\underline{x}, \lambda)$.

Perform predictor step

$$\begin{pmatrix} \underline{x}^{\#}_{i+1} \\ \lambda^{\#}_{i+1} \end{pmatrix} = \begin{pmatrix} \underline{x}_i \\ \lambda_i \end{pmatrix} + \tau_i \cdot \underline{v}_i,$$

τ_i step size, $\underline{v}_i$ normalized direction.

Homotopy Path Calculation



Corrector step

Fix one coordinate, run Newton iteration steps with start values $(\underline{x}^{\#}_{i+1}, \lambda^{\#}_{i+1})$ until

$$\underline{H}(\underline{x}_{i+1}, \lambda_{i+1}) \approx \underline{0}.$$

Homotopy Method

Calculating Homotopy Path $\quad (\underline{x}(s), \lambda(s))$

Predictor step

$$\underline{H}(\underline{x}(s), \lambda(s)) = \underline{0}$$

$$\Rightarrow \frac{\partial \underline{H}}{\partial \underline{x}} \cdot \underline{x}'(s) + \frac{\partial \underline{H}}{\partial \lambda} \cdot \lambda'(s) = \underline{0}.$$

Solve linear system

$$J_{\underline{H}}(\underline{x}_i, \lambda_i) \cdot \underline{v}_i = \underline{0},$$

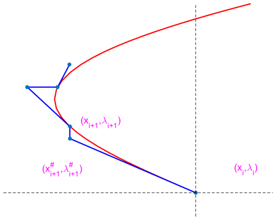
$J_{\underline{H}} \in \mathbb{R}^{(n,n+1)}$, Jacobian matrix of $\underline{H}(\underline{x}, \lambda)$.

Perform predictor step

$$\begin{pmatrix} \underline{x}_{i+1}^{\#} \\ \lambda_{i+1}^{\#} \end{pmatrix} = \begin{pmatrix} \underline{x}_i \\ \lambda_i \end{pmatrix} + \tau_i \cdot \underline{v}_i,$$

τ_i step size, $\underline{v}_i$ normalized direction.

Homotopy Path Calculation



Corrector step

Fix one coordinate, run Newton iteration steps with start values $(\underline{x}_{i+1}^{\#}, \lambda_{i+1}^{\#})$ until

$$\underline{H}(\underline{x}_{i+1}, \lambda_{i+1}) \approx \underline{0}.$$

Predictor step

$$\underline{H}(\underline{x}(s), \lambda(s)) = \underline{0}$$

$$\Rightarrow \frac{\partial \underline{H}}{\partial \underline{x}} \cdot \underline{x}'(s) + \frac{\partial \underline{H}}{\partial \lambda} \cdot \lambda'(s) = \underline{0}.$$

Solve linear system

$$J_{\underline{H}}(\underline{x}_i, \lambda_i) \cdot \underline{v}_i = \underline{0},$$

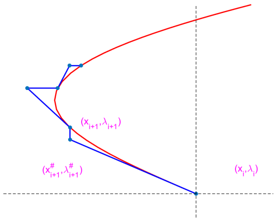
$J_{\underline{H}} \in \mathbb{R}^{(n,n+1)}$, Jacobian matrix of $\underline{H}(\underline{x}, \lambda)$.

Perform predictor step

$$\begin{pmatrix} \underline{x}^{\#}_{i+1} \\ \lambda^{\#}_{i+1} \end{pmatrix} = \begin{pmatrix} \underline{x}_i \\ \lambda_i \end{pmatrix} + \tau_i \cdot \underline{v}_i,$$

τ_i step size, $\underline{v}_i$ normalized direction.

Homotopy Path Calculation



Corrector step

Fix one coordinate, run Newton iteration steps with start values $(\underline{x}^{\#}_{i+1}, \lambda^{\#}_{i+1})$ until

$$\underline{H}(\underline{x}_{i+1}, \lambda_{i+1}) \approx \underline{0}.$$

Predictor step

$$\underline{H}(\underline{x}(s), \lambda(s)) = \underline{0}$$

$$\Rightarrow \frac{\partial \underline{H}}{\partial \underline{x}} \cdot \underline{x}'(s) + \frac{\partial \underline{H}}{\partial \lambda} \cdot \lambda'(s) = \underline{0}.$$

Solve linear system

$$J_{\underline{H}}(\underline{x}_i, \lambda_i) \cdot \underline{v}_i = \underline{0},$$

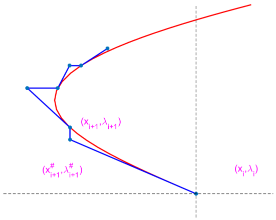
$J_{\underline{H}} \in \mathbb{R}^{(n,n+1)}$, Jacobian matrix of $\underline{H}(\underline{x}, \lambda)$.

Perform predictor step

$$\begin{pmatrix} \underline{x}^{\#}_{i+1} \\ \lambda^{\#}_{i+1} \end{pmatrix} = \begin{pmatrix} \underline{x}_i \\ \lambda_i \end{pmatrix} + \tau_i \cdot \underline{v}_i,$$

τ_i step size, $\underline{v}_i$ normalized direction.

Homotopy Path Calculation



Corrector step

Fix one coordinate, run Newton iteration steps with start values $(\underline{x}^{\#}_{i+1}, \lambda^{\#}_{i+1})$ until

$$\underline{H}(\underline{x}_{i+1}, \lambda_{i+1}) \approx \underline{0}.$$

FH Bielefeld University of Applied Sciences

Predictor step

$$\underline{H}(\underline{x}(s), \lambda(s)) = \underline{0}$$

$$\Rightarrow \frac{\partial \underline{H}}{\partial \underline{x}} \cdot \underline{x}'(s) + \frac{\partial \underline{H}}{\partial \lambda} \cdot \lambda'(s) = \underline{0}.$$

Solve linear system

$$J_{\underline{H}}(\underline{x}_i, \lambda_i) \cdot \underline{v}_i = \underline{0},$$

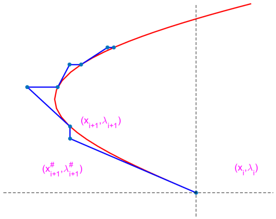
$J_{\underline{H}} \in \mathbb{R}^{(n,n+1)}$, Jacobian matrix of $\underline{H}(\underline{x}, \lambda)$.

Perform predictor step

$$\begin{pmatrix} \underline{x}_{i+1}^{\#} \\ \lambda_{i+1}^{\#} \end{pmatrix} = \begin{pmatrix} \underline{x}_i \\ \lambda_i \end{pmatrix} + \tau_i \cdot \underline{v}_i,$$

τ_i step size, $\underline{v}_i$ normalized direction.

Homotopy Path Calculation



Corrector step

Fix one coordinate, run Newton iteration steps with start values $(\underline{x}_{i+1}^{\#}, \lambda_{i+1}^{\#})$ until

$$\underline{H}(\underline{x}_{i+1}, \lambda_{i+1}) \approx \underline{0}.$$

Homotopy Method

Calculating Homotopy Path $(\underline{x}(s), \lambda(s))$

Predictor step

$$\underline{H}(\underline{x}(s), \lambda(s)) = \underline{0}$$

$$\Rightarrow \frac{\partial \underline{H}}{\partial \underline{x}} \cdot \underline{x}'(s) + \frac{\partial \underline{H}}{\partial \lambda} \cdot \lambda'(s) = \underline{0}.$$

Solve linear system

$$J_{\underline{H}}(\underline{x}_i, \lambda_i) \cdot \underline{v}_i = \underline{0},$$

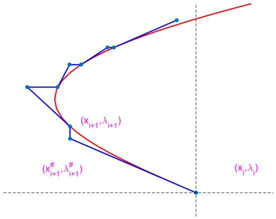
$J_{\underline{H}} \in \mathbb{R}^{(n,n+1)}$, Jacobian matrix of $\underline{H}(\underline{x}, \lambda)$.

Perform predictor step

$$\begin{pmatrix} \underline{x}^{\#}_{i+1} \\ \lambda^{\#}_{i+1} \end{pmatrix} = \begin{pmatrix} \underline{x}_i \\ \lambda_i \end{pmatrix} + \tau_i \cdot \underline{v}_i,$$

τ_i step size, $\underline{v}_i$ normalized direction.

Homotopy Path Calculation



Corrector step

Fix one coordinate, run Newton iteration steps with start values $(\underline{x}^{\#}_{i+1}, \lambda^{\#}_{i+1})$ until

$$\underline{H}(\underline{x}_{i+1}, \lambda_{i+1}) \approx \underline{0}.$$

Homotopy Method

Calculating Homotopy Path $(\underline{x}(s), \lambda(s))$

Predictor step

$$\underline{H}(\underline{x}(s), \lambda(s)) = \underline{0}$$

$$\Rightarrow \frac{\partial \underline{H}}{\partial \underline{x}} \cdot \underline{x}'(s) + \frac{\partial \underline{H}}{\partial \lambda} \cdot \lambda'(s) = \underline{0}.$$

Solve linear system

$$J_{\underline{H}}(\underline{x}_i, \lambda_i) \cdot \underline{v}_i = \underline{0},$$

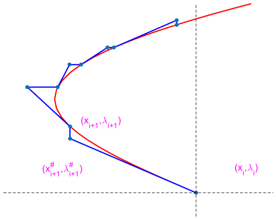
$J_{\underline{H}} \in \mathbb{R}^{(n,n+1)}$, Jacobian matrix of $\underline{H}(\underline{x}, \lambda)$.

Perform predictor step

$$\begin{pmatrix} \underline{x}_{i+1}^{\#} \\ \lambda_{i+1}^{\#} \end{pmatrix} = \begin{pmatrix} \underline{x}_i \\ \lambda_i \end{pmatrix} + \tau_i \cdot \underline{v}_i,$$

τ_i step size, $\underline{v}_i$ normalized direction.

Homotopy Path Calculation



Corrector step

Fix one coordinate, run Newton iteration steps with start values $(\underline{x}_{i+1}^{\#}, \lambda_{i+1}^{\#})$ until

$$\underline{H}(\underline{x}_{i+1}, \lambda_{i+1}) \approx \underline{0}.$$

Predictor step

$$\underline{H}(\underline{x}(s), \lambda(s)) = \underline{0}$$

$$\Rightarrow \frac{\partial \underline{H}}{\partial \underline{x}} \cdot \underline{x}'(s) + \frac{\partial \underline{H}}{\partial \lambda} \cdot \lambda'(s) = \underline{0}.$$

Solve linear system

$$J_{\underline{H}}(\underline{x}_i, \lambda_i) \cdot \underline{v}_i = \underline{0},$$

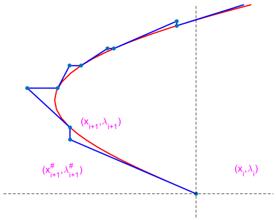
$J_{\underline{H}} \in \mathbb{R}^{(n,n+1)}$, Jacobian matrix of $\underline{H}(\underline{x}, \lambda)$.

Perform predictor step

$$\begin{pmatrix} \underline{x}^{\#}_{i+1} \\ \lambda^{\#}_{i+1} \end{pmatrix} = \begin{pmatrix} \underline{x}_i \\ \lambda_i \end{pmatrix} + \tau_i \cdot \underline{v}_i,$$

τ_i step size, $\underline{v}_i$ normalized direction.

Homotopy Path Calculation



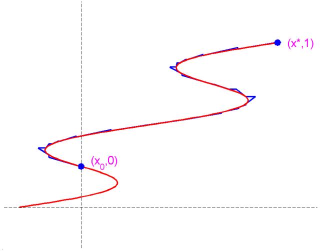
Corrector step

Fix one coordinate, run Newton iteration steps with start values $(\underline{x}^{\#}_{i+1}, \lambda^{\#}_{i+1})$ until

$$\underline{H}(\underline{x}_{i+1}, \lambda_{i+1}) \approx \underline{0}.$$

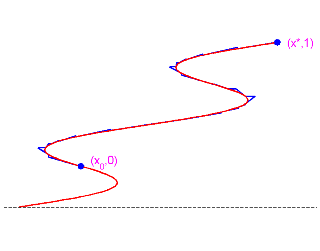
Problems in current implementation (To-Do-List):

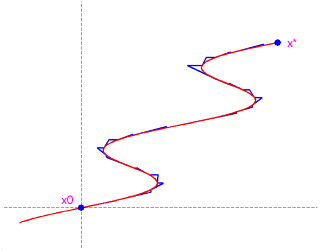
- Determination of starting direction
  - For Newton-Homotopy try both directions
  - For Fixpoint-Homotopy only positive direction
- Improve scaling of iteration variables and equations
- Utilize Homotopy method for initialization
- Provide better starting vector for iteration



Problems in current implementation (To-Do-List):

- Determination of starting direction
  - For Newton-Homotopy try both directions
  - For Fixpoint-Homotopy only positive direction
- Improve scaling of iteration variables and equations
- Utilize Homotopy method for initialization
- Provide better starting vector for iteration



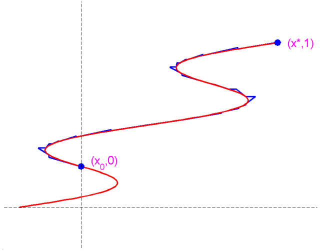


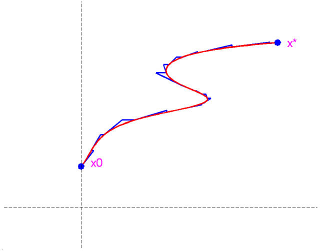
Homotopy Method

Status and Outlook

Problems in current implementation (To-Do-List):

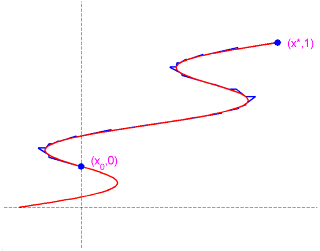
- Determination of starting direction
  - For Newton-Homotopy try both directions
  - For Fixpoint-Homotopy only positive direction
- Improve scaling of iteration variables and equations
- Utilize Homotopy method for initialization
- Provide better starting vector for iteration

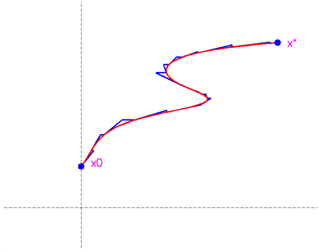




Problems in current implementation (To-Do-List):

- Determination of starting direction
  - For Newton-Homotopy try both directions
  - For Fixpoint-Homotopy only positive direction
- Improve scaling of iteration variables and equations
  - Problems with differentiated variables
- Utilize Homotopy method for initialization
- Provide better starting vector for iteration



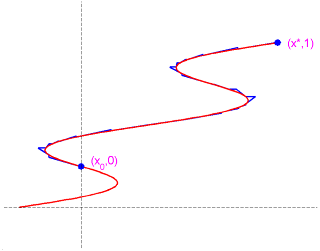


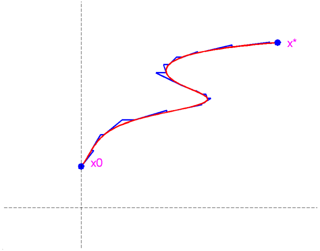
Homotopy Method

Status and Outlook

Problems in current implementation (To-Do-List):

- Determination of starting direction
  - For Newton-Homotopy try both directions
  - For Fixpoint-Homotopy only positive direction
- Improve scaling of iteration variables and equations
  - Problems with differentiated variables
- Utilize Homotopy method for initialization
- Provide better starting vector for iteration



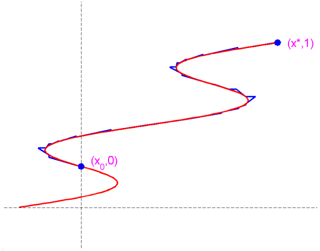


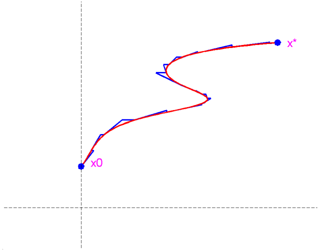
Homotopy Method

Status and Outlook

Problems in current implementation (To-Do-List):

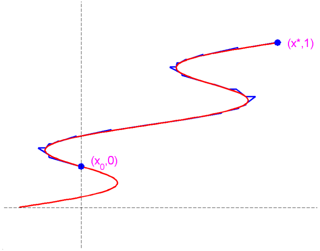
- Determination of starting direction
  - For Newton-Homotopy try both directions
  - For Fixpoint-Homotopy only positive direction
- Improve scaling of iteration variables and equations
  - Problems with differentiated variables
- Utilize Homotopy method for initialization
  - Proper symbolic preparation of the operator homotopy(f(x),g(x))=0 necessary
- Provide better starting vector for iteration

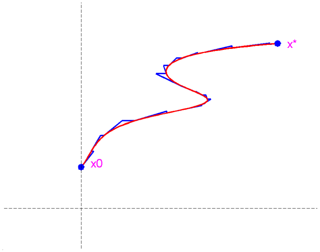




Problems in current implementation (To-Do-List):

- Determination of starting direction
  - For Newton-Homotopy try both directions
  - For Fixpoint-Homotopy only positive direction
- Improve scaling of iteration variables and equations
  - Problems with differentiated variables
- Utilize Homotopy method for initialization
  - Proper symbolic preparation of the operator `homotopy(f(x),g(x))=0` necessary.
- Provide better starting vector for iteration





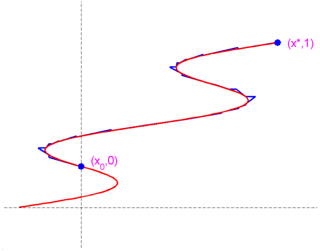
FH Bielefeld
University of
Applied Sciences

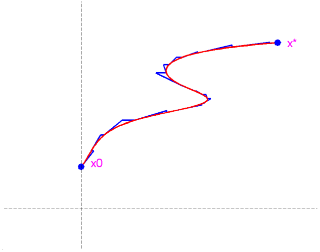
Problems in current implementation (To-Do-List):

- Determination of starting direction
  - For Newton-Homotopy try both directions
  - For Fixpoint-Homotopy only positive direction
- Improve scaling of iteration variables and equations
  - Problems with differentiated variables
- Utilize Homotopy method for initialization
  - Proper symbolic preparation of the operator `homotopy(f(x),g(x))=0` necessary.
- Provide better starting vector for iteration
  - Context-dependent extrapolation:

$$J_{fODE} = \left(\frac{fODE(\underline{x} + h\underline{e_i}) - fODE(\underline{x})}{h} \right)_{i=1...n}$$

  - OR: Generate symbolic Jacobians for all relevant cases



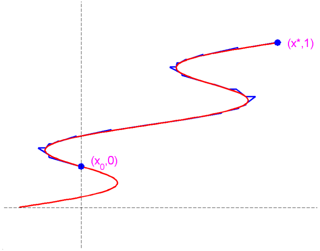


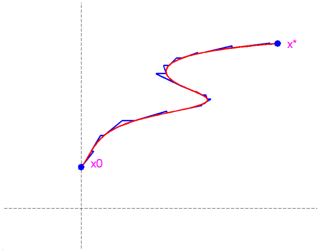
Problems in current implementation (To-Do-List):

- Determination of starting direction
  - For Newton-Homotopy try both directions
  - For Fixpoint-Homotopy only positive direction
- Improve scaling of iteration variables and equations
  - Problems with differentiated variables
- Utilize Homotopy method for initialization
  - Proper symbolic preparation of the operator `homotopy(f(x),g(x))=0` necessary.
- Provide better starting vector for iteration
  - Context-dependent extrapolation:

$$J_{fODE} = \left(\frac{fODE(\underline{x} + h\,\underline{e_i}) - fODE(\underline{x})}{h} \right)_{i=1\ldots n}$$

  - OR: Generate symbolic Jacobians for all relevant cases



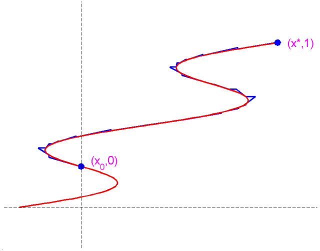


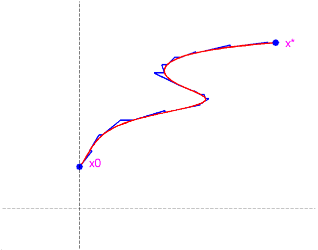
Problems in current implementation (To-Do-List):

- Determination of starting direction
  - For Newton-Homotopy try both directions
  - For Fixpoint-Homotopy only positive direction
- Improve scaling of iteration variables and equations
  - Problems with differentiated variables
- Utilize Homotopy method for initialization
  - Proper symbolic preparation of the operator `homotopy(f(x),g(x))=0` necessary.
- Provide better starting vector for iteration
  - Context-dependent extrapolation:

$$J_{fODE} = \left(\frac{fODE(\underline{x} + h\,\underline{e_i}) - fODE(\underline{x})}{h} \right)_{i=1\ldots n}$$

  - OR: Generate symbolic Jacobians for all relevant cases





Symbolic inversion of non-linear functions in OpenModelica

- Already available for most of known functions:
  - quadratic functions: $ax^2 + bx + c$,
  - monomial functions: x^n,
  - $\sin(x)$,
  - $\cos(x)$,
  - $\tan(x)$,
  - $\log(x)$,
  - $\exp(x)$,
  - $\tanh(x)$,
  - $\cosh(x)$,
  - ...
- Complex compositions are possible:

Symbolic inversion of non-linear functions in OpenModelica

- Already available for most of known functions:
  - quadratic functions: $ax^2 + bx + c$,
  - monomial functions: x^n,
  - $\sin(x)$,
  - $\cos(x)$,
  - $\tan(x)$,
  - $\log(x)$,
  - $\exp(x)$,
  - $\tanh(x)$,
  - $\cosh(x)$,
  - ...
- Complex compositions are possible:

New Features in Module: ExpressionSolve

Methods for Solving Non-Linear Single Equations

Symbolic inversion of non-linear functions in OpenModelica

- Already available for most of known functions:
  - quadratic functions: $ax^2 + bx + c$,
  - monomial functions: x^n,
  - $\sin(x)$,
  - $\cos(x)$,
  - $\tan(x)$,
  - $\log(x)$,
  - $\exp(x)$,
  - $\tanh(x)$,
  - $\cosh(x)$,
  - $\ldots$
- Complex compositions are possible:

New Features in Module: ExpressionSolve

Methods for Solving Non-Linear Single Equations

Symbolic inversion of non-linear functions in OpenModelica

- Already available for most of known functions:
  - quadratic functions: $ax^2 + bx + c$,
  - monomial functions: x^n,
  - $\sin(x)$,
  - $\cos(x)$,
  - $\tan(x)$,
  - $\log(x)$,
  - $\exp(x)$,
  - $\tanh(x)$,
  - $\cosh(x)$,
  - $\dots$
- Complex compositions are possible:

Symbolic inversion of non-linear functions in OpenModelica

- Already available for most of known functions:
  - quadratic functions: $ax^2 + bx + c$,
  - monomial functions: x^n,
  - $\sin(x)$,
  - $\cos(x)$,
  - $\tan(x)$,
  - $\log(x)$,
  - $\exp(x)$,
  - $\tanh(x)$,
  - $\cosh(x)$,
  - ...
- Complex compositions are possible:

New Features in Module: ExpressionSolve

Methods for Solving Non-Linear Single Equations

Symbolic inversion of non-linear functions in OpenModelica

- Already available for most of known functions:
  - quadratic functions: $ax^2 + bx + c$,
  - monomial functions: x^n,
  - $\sin(x)$,
  - $\cos(x)$,
  - $\tan(x)$,
  - $\log(x)$,
  - $\exp(x)$,
  - $\tanh(x)$,
  - $\cosh(x)$,
  - ...
- Complex compositions are possible:

New Features in Module: ExpressionSolve

Methods for Solving Non-Linear Single Equations

Symbolic inversion of non-linear functions in OpenModelica

- Already available for most of known functions:
  - quadratic functions: $ax^2 + bx + c$,
  - monomial functions: x^n,
  - $\sin(x)$,
  - $\cos(x)$,
  - $\tan(x)$,
  - $\log(x)$,
  - $\exp(x)$,
  - $\tanh(x)$,
  - $\cosh(x)$,
  - $\dots$
- Complex compositions are possible:

New Features in Module: ExpressionSolve

Methods for Solving Non-Linear Single Equations

Symbolic inversion of non-linear functions in OpenModelica

- Already available for most of known functions:
  - quadratic functions: $ax^2 + bx + c$,
  - monomial functions: x^n,
  - $\sin(x)$,
  - $\cos(x)$,
  - $\tan(x)$,
  - $\log(x)$,
  - $\exp(x)$,
  - $\tanh(x)$,
  - $\cosh(x)$,
  - ...
- Complex compositions are possible:

Symbolic inversion of non-linear functions in OpenModelica

- Already available for most of known functions:
  - quadratic functions: $ax^2 + bx + c$,
  - monomial functions: x^n,
  - $\sin(x)$,
  - $\cos(x)$,
  - $\tan(x)$,
  - $\log(x)$,
  - $\exp(x)$,
  - $\tanh(x)$,
  - $\cosh(x)$,
  - $\dots$
- Complex compositions are possible:

New Features in Module: ExpressionSolve

Methods for Solving Non-Linear Single Equations

FH Bielefeld
University of
Applied Sciences

Symbolic inversion of non-linear functions in OpenModelica

- Already available for most of known functions:
  - quadratic functions: $ax^2 + bx + c$,
  - monomial functions: x^n,
  - $\sin(x)$,
  - $\cos(x)$,
  - $\tan(x)$,
  - $\log(x)$,
  - $\exp(x)$,
  - $\tanh(x)$,
  - $\cosh(x)$,
  - ...
- Complex compositions are possible:

New Features in Module: ExpressionSolve

Methods for Solving Non-Linear Single Equations

Symbolic inversion of non-linear functions in OpenModelica

- Already available for most of known functions:
  - quadratic functions: $ax^2 + bx + c$,
  - monomial functions: x^n,
  - $\sin(x)$,
  - $\cos(x)$,
  - $\tan(x)$,
  - $\log(x)$,
  - $\exp(x)$,
  - $\tanh(x)$,
  - $\cosh(x)$,
  - $\ldots$
- Complex compositions are possible:

Symbolic inversion of non-linear functions in OpenModelica

- Already available for most of known functions:
  - quadratic functions: $ax^2 + bx + c$,
  - monomial functions: x^n,
  - $\sin(x)$,
  - $\cos(x)$,
  - $\tan(x)$,
  - $\log(x)$,
  - $\exp(x)$,
  - $\tanh(x)$,
  - $\cosh(x)$,
  - $\ldots$
- Complex compositions are possible:
  - $$a(t) \cdot f(x)^{n(t)} + b(t) \cdot f(x)^{\frac{n(t)}{2}} + c(t) = 0$$
  - $\ldots$

FH Bielefeld
University of
Applied Sciences

Symbolic inversion of non-linear functions in OpenModelica

- Already available for most of known functions:
  - quadratic functions: $ax^2 + bx + c$,
  - monomial functions: x^n,
  - $\sin(x)$,
  - $\cos(x)$,
  - $\tan(x)$,
  - $\log(x)$,
  - $\exp(x)$,
  - $\tanh(x)$,
  - $\cosh(x)$,
  - $\ldots$
- Complex compositions are possible:
  - $$a(t) \cdot f(x)^{y(t)} + b(t) \cdot f(x)^{\frac{y(t)}{2}} + c(t) = 0$$

Symbolic inversion of non-linear functions in OpenModelica

- Already available for most of known functions:
  - quadratic functions: $ax^2 + bx + c$,
  - monomial functions: x^n,
  - $\sin(x)$,
  - $\cos(x)$,
  - $\tan(x)$,
  - $\log(x)$,
  - $\exp(x)$,
  - $\tanh(x)$,
  - $\cosh(x)$,
  - ...
- Complex compositions are possible:
  - $$a(t) \cdot f(x)^{y(t)} + b(t) \cdot f(x)^{\frac{y(t)}{2}} + c(t) = 0$$
  - ...

New Features in Module: ExpressionSolve

Methods for Solving Non-Linear Single Equations

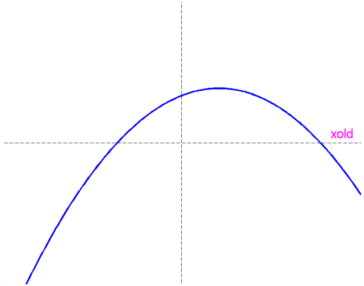
FH Bielefeld
University of
Applied Sciences

Quadratic equation

$$f(x) = a \cdot x^2 + b \cdot x + c = 0$$

Solution

$$x = \min_{x^*} \left\{ \left| x^* - x_{\text{old}} \right| \;\middle|\; x^* \in \{x_1, x_2\} \right\}$$

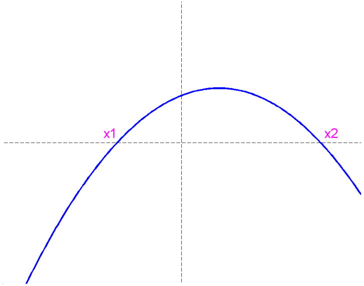


Quadratic equation

$$f(x) = a \cdot x^2 + b \cdot x + c = 0$$

Well-known solution formula

$$x_{1,2} = \begin{cases} \dfrac{-\,b \pm \sqrt{b^2 - 4 \cdot a \cdot c}}{2 \cdot a} & \text{if } a \neq 0 \\[2ex] \dfrac{-\,c}{b} & \text{if } a = 0 \end{cases}$$



Solution

$$x = \min_{x^*} \left\{ \left| x^* - x_{\text{old}} \right| \,\middle|\, x^* \in \{x_1, x_2\} \right\}$$

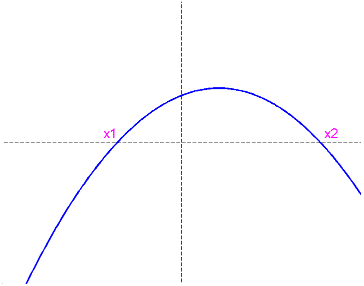
Quadratic equation

$$f(x) = a \cdot x^2 + b \cdot x + c = 0$$

Numerically stable solution formula (Vieta)

$$x_1 = -\left(\frac{b + \operatorname{sign}(b) \cdot \sqrt{b^2 - 4 \cdot a \cdot c}}{2 \cdot a}\right)$$

$$x_2 = \frac{c}{a \cdot x_1}$$



Solution

$$x = \min_{x^*} \left\{ \left| x^* - x_{\text{old}} \right| \;\middle|\; x^* \in \{x_1, x_2\} \right\}$$

FH Bielefeld
University of
Applied Sciences

Quadratic equation

$$f(x) = a \cdot x^2 + b \cdot x + c = 0$$

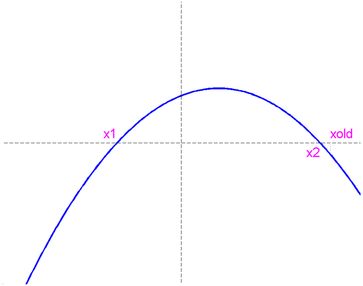
Numerically stable solution formula (Vieta)

$$x_1 = -\left(\frac{b + \operatorname{sign}(b) \cdot \sqrt{b^2 - 4 \cdot a \cdot c}}{2 \cdot a}\right)$$

$$x_2 = \frac{c}{a \cdot x_1}$$

Solution

$$x = \min_{x^*} \left\{ \left| x^* - x_{\text{old}} \right| \;\middle|\; x^* \in \{x_1, x_2\} \right\}$$



Generalization using substitution

$$a(t) \cdot f(x)^{y(t)} + b(t) \cdot f(x)^{\frac{y(t)}{2}} + c(t) = 0$$

Example

$$\sin(t) \cdot \log(x)^{2 \cdot n} + \cos(t) \cdot \log(x)^{n} + \exp(t) = 0$$

substitution:

$$\sin(t) \cdot z^2 + \cos(t) \cdot z + \exp(t) = 0$$

where $z = \log(x)^n = \mathrm{solveQE}(.)$.

$$\rightarrow x = \exp\left(\mathrm{sign}(z_{old}) \cdot \left|\mathrm{solveQE}(.)^{\frac{1}{n}}\right|\right)$$

New Features in Module: ExpressionSolve

Methods for Solving Non-Linear Single Equations

Generalization using substitution

$$a(t) \cdot f(x)^{y(t)} + b(t) \cdot f(x)^{\frac{y(t)}{2}} + c(t) = 0$$

Example

$$\sin(t) \cdot \log(x)^{2 \cdot n} + \cos(t) \cdot \log(x)^{n} + \exp(t) = 0$$

substitution:

$$\sin(t) \cdot z^2 + \cos(t) \cdot z + \exp(t) = 0$$

where $z = \log(x)^n = \text{solveQE}(.)$.

$$\rightarrow x = \exp\left(\text{sign}(z_{0/1}) \cdot \left|\text{solveQE}(.)^{\frac{1}{n}}\right|\right)$$

New Features in Module: ExpressionSolve

Methods for Solving Non-Linear Single Equations

FH Bielefeld
University of
Applied Sciences

Generalization using substitution

$$a(t) \cdot f(x)^{y(t)} + b(t) \cdot f(x)^{\frac{y(t)}{2}} + c(t) = 0$$

Example

$$\sin(t) \cdot \log(x)^{2 \cdot n} + \cos(t) \cdot \log(x)^{n} + \exp(t) = 0$$

substitution:

$$\sin(t) \cdot z^2 + \cos(t) \cdot z + \exp(t) = 0$$

where $z = \log(x)^n = \text{solveQE}(.)$.

$$\rightarrow x = \exp\left(\text{sign}(z_{old}) \cdot \left|\text{solveQE}(.)^{\frac{1}{n}}\right|\right)$$

Generalization using substitution

$$a(t) \cdot f(x)^{y(t)} + b(t) \cdot f(x)^{\frac{y(t)}{2}} + c(t) = 0$$

Example

$$\sin(t) \cdot \log(x)^{2 \cdot n} + \cos(t) \cdot \log(x)^{n} + \exp(t) = 0$$

substitution:

$$\sin(t) \cdot z^2 + \cos(t) \cdot z + \exp(t) = 0$$

where $z = \log(x)^n = \mathrm{solveQE}(.)$.

$$\rightarrow x = \exp\left(\mathrm{sign}(z_{old}) \cdot \left|\mathrm{solveQE}(.)^{\frac{1}{n}}\right|\right)$$

New Features in Module: ExpressionSolve

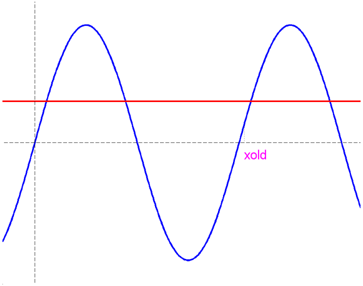
Methods for Solving Non-Linear Single Equations

Inversion of sine function

$$\sin(x) = y$$

Solution

$$x = \min_{x^*}\left\{ \left| x^* - x_{\text{old}} \right| \;\middle|\; x^* \in \{x_1, x_2\} \right\}$$



New Features in Module: ExpressionSolve

Methods for Solving Non-Linear Single Equations

Solution process

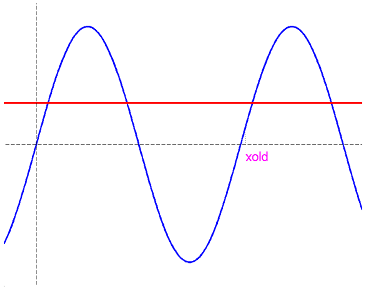
$$\sin(x) = y$$

$$x = \operatorname{asin}(y) + 2 \cdot k \cdot \pi, \qquad k \in \mathbb{Z}$$

$$x = -\operatorname{asin}(y) + (2 \cdot k + 1) \cdot \pi, \qquad k \in \mathbb{Z}$$

Solution

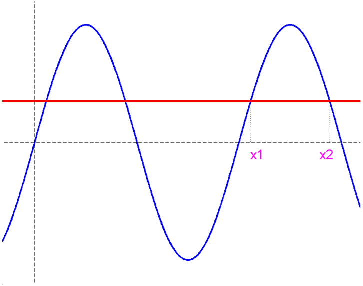
$$x = \min_{x^*} \left\{ \left| x^* - x_{\text{old}} \right| \,\middle|\, x^* \in \{x_1, x_2\} \right\}$$



Solution process

$$
\begin{aligned}
\sin(x) &= y \\
x_1 &= \operatorname{asin}(y) + 2 \cdot k_1 \cdot \pi, & k_1 \in \mathbb{Z} \\
x_2 &= -\operatorname{asin}(y) + (2 \cdot k_2 + 1) \cdot \pi, & k_2 \in \mathbb{Z}
\end{aligned}
$$

$$
\begin{aligned}
k_1 &= \operatorname{round}\left(\frac{x_{\text{old}} - \operatorname{asin}(y)}{2 \cdot \pi} \right) \\
k_2 &= \operatorname{round}\left(\frac{x_{\text{old}} + \operatorname{asin}(y)}{2 \cdot \pi} - \frac{1}{2} \right)
\end{aligned}
$$



Solution

$$
x = \min_{x^*} \left\{ \left| x^* - x_{\text{old}} \right| \,\middle|\, x^* \in \{x_1, x_2\} \right\}
$$

Solution process

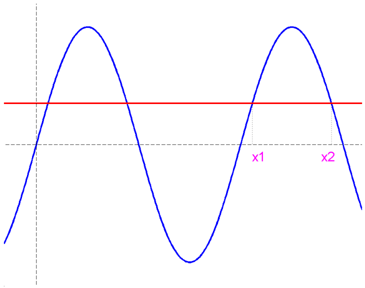
$$\sin(x) = y$$

$$x_1 = \operatorname{asin}(y) + 2 \cdot k_1 \cdot \pi, \qquad k_1 \in \mathbb{Z}$$

$$x_2 = -\operatorname{asin}(y) + (2 \cdot k_2 + 1) \cdot \pi, \qquad k_2 \in \mathbb{Z}$$

$$k_1 = \text{round}\left(\frac{x_{\text{old}} - \operatorname{asin}(y)}{2 \cdot \pi}\right)$$

$$k_2 = \text{round}\left(\frac{x_{\text{old}} + \operatorname{asin}(y)}{2 \cdot \pi} - \frac{1}{2}\right)$$



Solution

$$x = \min_{x^*}\left\{\left|x^* - x_{\text{old}}\right| \,\middle|\, x^* \in \{x_1, x_2\}\right\}$$

Strong component

$$
\begin{aligned}
f_1(x_3, x_4): &\quad 5x_3 + x_4 = 0 \\
f_2(x_1, x_2): &\quad x_1 + x_2 + \mathit{time} = 0 \\
f_3(x_1, x_4): &\quad \sin(x_1) - x_4 = 0 \\
f_4(x_2, x_3): &\quad 2x_2 + x_3 = 0
\end{aligned}
$$

Strong component

$$\begin{aligned} f_1(x_3, x_4): &\quad 5x_3 + x_4 = 0 \\ f_2(x_1, x_2): &\quad x_1 + x_2 + \mathit{time} = 0 \\ f_3(x_1, x_4): &\quad \sin(x_1) - x_4 = 0 \\ f_4(x_2, x_3): &\quad 2x_2 + x_3 = 0 \end{aligned}$$

- Tearing idea:
  - Select a set of tearing variables and treat them as known variables in the following
  - Transform the system to a sequentially evaluable one

Teared component

$$
\begin{aligned}
f_1(x_3, x_4): &\quad 5x_3 + x_4 = 0 \\
f_2(\boldsymbol{x_1}, x_2): &\quad \boldsymbol{x_1} + x_2 + time = 0 \\
f_3(\boldsymbol{x_1}, x_4): &\quad \sin(\boldsymbol{x_1}) - x_4 = 0 \\
f_4(x_2, x_3): &\quad 2x_2 + x_3 = 0
\end{aligned}
$$

- Tearing idea:
  - Select a set of tearing variables and treat them as known variables in the following
  - Transform the system to a sequentially evaluable one

Causalised component

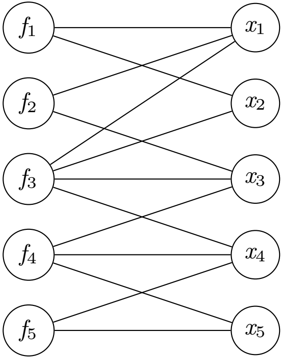
$$
\begin{aligned}
f_2 &: \; x_2 := -\boldsymbol{x_1} - time \\
f_4 &: \; x_3 := -2x_2 \\
f_1 &: \; x_4 := -5x_3 \\
f_3 &: \; \sin(\boldsymbol{x_1}) - x_4 \overset{!}{=} 0 \qquad (res)
\end{aligned}
$$

- Tearing idea:
  - Select a set of tearing variables and treat them as known variables in the following
  - Transform the system to a sequentially evaluable one

FH Bielefeld
University of
Applied Sciences

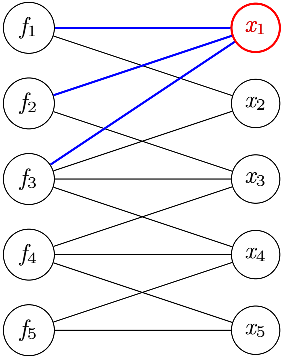
Example

$$
\begin{aligned}
f_1 &: \quad x_1 + x_2 \cdot \mathit{time} = 0 \\
f_2 &: \quad x_1 - x_3 \cdot \cos(\mathit{time}) = 0 \\
f_3 &: \quad x_1 + x_2 + x_3 + 2x_4 + 4 = 0 \\
f_4 &: \quad x_3 + x_4 + 2x_5 + 2 = 0 \\
f_5 &: \quad x_4 - x_5 \cdot \mathit{time} - 2 \cdot \mathit{time} = 0
\end{aligned}
$$



Example

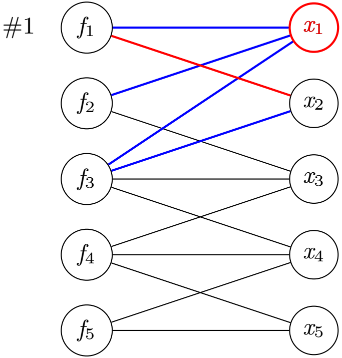
$$
\begin{aligned}
f_1 &: \quad x_1 + x_2 \cdot \mathit{time} = 0 \\
f_2 &: \quad x_1 - x_3 \cdot \cos(\mathit{time}) = 0 \\
f_3 &: \quad x_1 + x_2 + x_3 + 2x_4 + 4 = 0 \\
f_4 &: \quad x_3 + x_4 + 2x_5 + 2 = 0 \\
f_5 &: \quad x_4 - x_5 \cdot \mathit{time} - 2 \cdot \mathit{time} = 0
\end{aligned}
$$



FH Bielefeld
University of
Applied Sciences

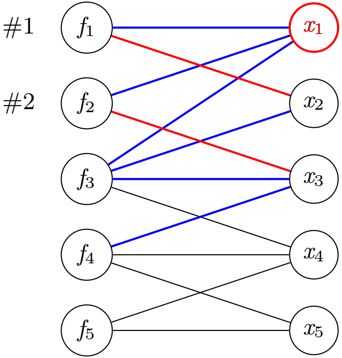
Example

$$
\begin{aligned}
f_1 &: \quad x_1 + x_2 \cdot time = 0 \\
f_2 &: \quad x_1 - x_3 \cdot \cos(time) = 0 \\
f_3 &: \quad x_1 + x_2 + x_3 + 2x_4 + 4 = 0 \\
f_4 &: \quad x_3 + x_4 + 2x_5 + 2 = 0 \\
f_5 &: \quad x_4 - x_5 \cdot time - 2 \cdot time = 0
\end{aligned}
$$



Example

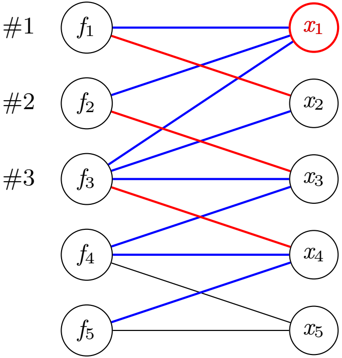
$$
\begin{aligned}
f_1 :& \quad x_1 + x_2 \cdot \mathit{time} = 0\\
f_2 :& \quad x_1 - x_3 \cdot \cos(\mathit{time}) = 0\\
f_3 :& \quad x_1 + x_2 + x_3 + 2x_4 + 4 = 0\\
f_4 :& \quad x_3 + x_4 + 2x_5 + 2 = 0\\
f_5 :& \quad x_4 - x_5 \cdot \mathit{time} - 2 \cdot \mathit{time} = 0
\end{aligned}
$$



FH Bielefeld
University of
Applied Sciences

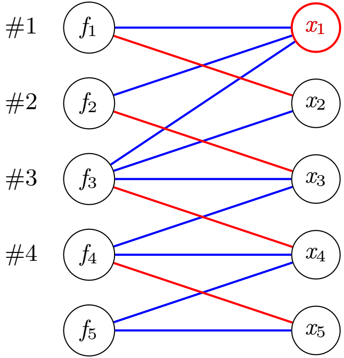
Example

$$
\begin{aligned}
f_1 &: \quad x_1 + x_2 \cdot time = 0 \\
f_2 &: \quad x_1 - x_3 \cdot \cos(time) = 0 \\
f_3 &: \quad x_1 + x_2 + x_3 + 2x_4 + 4 = 0 \\
f_4 &: \quad x_3 + x_4 + 2x_5 + 2 = 0 \\
f_5 &: \quad x_4 - x_5 \cdot time - 2 \cdot time = 0
\end{aligned}
$$



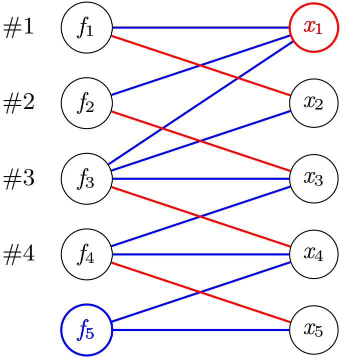
Example

$$
\begin{aligned}
f_1 :&\quad x_1 + x_2 \cdot \mathit{time} = 0 \\
f_2 :&\quad x_1 - x_3 \cdot \cos(\mathit{time}) = 0 \\
f_3 :&\quad x_1 + x_2 + x_3 + 2x_4 + 4 = 0 \\
f_4 :&\quad x_3 + x_4 + 2x_5 + 2 = 0 \\
f_5 :&\quad x_4 - x_5 \cdot \mathit{time} - 2 \cdot \mathit{time} = 0
\end{aligned}
$$



Example

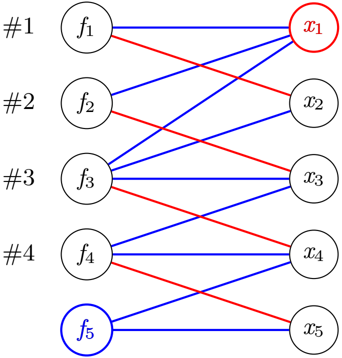
$$
\begin{aligned}
f_1 :&\quad x_1 + x_2 \cdot \mathit{time} = 0\\
f_2 :&\quad x_1 - x_3 \cdot \cos(\mathit{time}) = 0\\
f_3 :&\quad x_1 + x_2 + x_3 + 2x_4 + 4 = 0\\
f_4 :&\quad x_3 + x_4 + 2x_5 + 2 = 0\\
f_5 :&\quad x_4 - x_5 \cdot \mathit{time} - 2 \cdot \mathit{time} = 0
\end{aligned}
$$



FH Bielefeld
University of
Applied Sciences

Example

$$
\begin{aligned}
f_1 &: \quad \boldsymbol{x_1} + x_2 \cdot \mathit{time} = 0 \\
f_2 &: \quad \boldsymbol{x_1} - x_3 \cdot \cos(\mathit{time}) = 0 \\
f_3 &: \quad \boldsymbol{x_1} + x_2 + x_3 + 2x_4 + 4 = 0 \\
f_4 &: \quad x_3 + x_4 + 2x_5 + 2 = 0 \\
\boldsymbol{f_5} &: \quad x_4 - x_5 \cdot \mathit{time} - 2 \cdot \mathit{time} = 0
\end{aligned}
$$



Causalised system

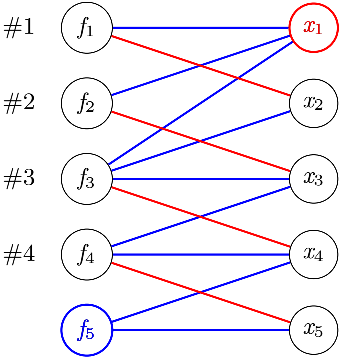
$$f_1 : \; x_2 := \frac{-\boldsymbol{x_1}}{time}$$

$$f_2 : \; x_3 := \frac{\boldsymbol{x_1}}{\cos(time)}$$

$$f_3 : \; x_4 := \frac{-\boldsymbol{x_1} - x_2 - x_3 - 4}{2}$$

$$f_4 : \; x_5 := \frac{-x_3 - x_4 - 2}{2}$$

$$\boldsymbol{f_5} : \; x_4 - x_5 \cdot time - 2 \cdot time \overset{!}{=} 0 \qquad (res)$$



Causalised system

$$f_1 : \; x_2 := \frac{-\boldsymbol{x_1}}{time}$$

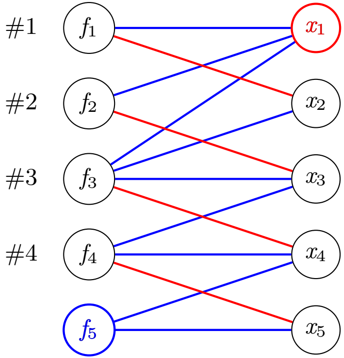
$$f_2 : \; x_3 := \frac{\boldsymbol{x_1}}{\cos(time)}$$

$$f_3 : \; x_4 := \frac{-\boldsymbol{x_1} - x_2 - x_3 - 4}{2}$$

$$f_4 : \; x_5 := \frac{-x_3 - x_4 - 2}{2}$$

$$\boldsymbol{f_5} : \; x_4 - x_5 \cdot time - 2 \cdot time \stackrel{!}{=} 0 \qquad (res)$$

$\rightarrow$ Division by zero $\Rightarrow$ Simulation failure

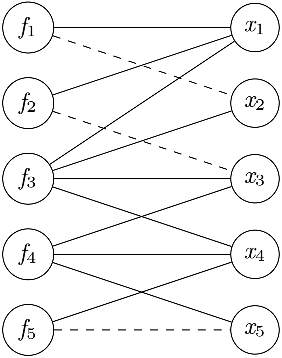


FH Bielefeld
University of
Applied Sciences

Example

$$
\begin{aligned}
f_1 &: \quad x_1 + x_2 \cdot \mathit{time} = 0 \\
f_2 &: \quad x_1 - x_3 \cdot \cos(\mathit{time}) = 0 \\
f_3 &: \quad x_1 + x_2 + x_3 + 2x_4 + 4 = 0 \\
f_4 &: \quad x_3 + x_4 + 2x_5 + 2 = 0 \\
f_5 &: \quad x_4 - x_5 \cdot \mathit{time} - 2 \cdot \mathit{time} = 0
\end{aligned}
$$

With constraints for assignments:

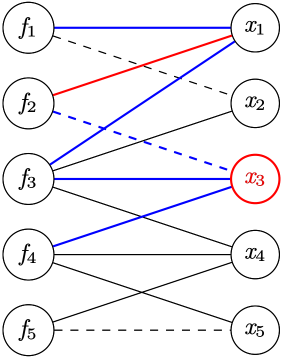


Example

$$
\begin{aligned}
f_1 &: \quad x_1 + x_2 \cdot \mathit{time} = 0 \\
f_2 &: \quad x_1 - x_3 \cdot \cos(\mathit{time}) = 0 \\
f_3 &: \quad x_1 + x_2 + x_3 + 2x_4 + 4 = 0 \\
f_4 &: \quad x_3 + x_4 + 2x_5 + 2 = 0 \\
f_5 &: \quad x_4 - x_5 \cdot \mathit{time} - 2 \cdot \mathit{time} = 0
\end{aligned}
$$

With constraints for assignments:

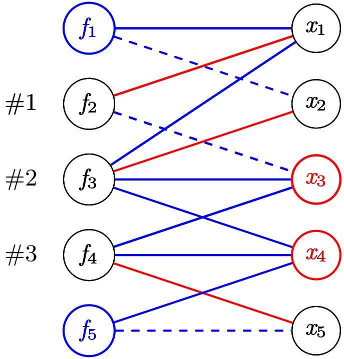
$\#1$



Example

$$
\begin{aligned}
f_1 &: \quad x_1 + x_2 \cdot time = 0 \\
f_2 &: \quad x_1 - x_3 \cdot \cos(time) = 0 \\
f_3 &: \quad x_1 + x_2 + x_3 + 2x_4 + 4 = 0 \\
f_4 &: \quad x_3 + x_4 + 2x_5 + 2 = 0 \\
f_5 &: \quad x_4 - x_5 \cdot time - 2 \cdot time = 0
\end{aligned}
$$

With constraints for assignments:

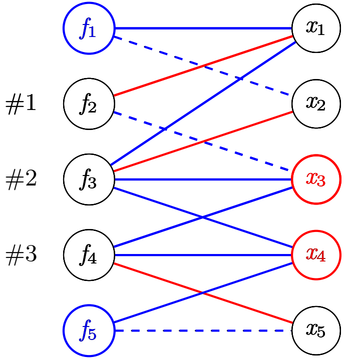


FH Bielefeld
University of
Applied Sciences

Example

$$
\begin{aligned}
\boldsymbol{f_1} &: \quad x_1 + x_2 \cdot time = 0 \\
f_2 &: \quad x_1 - \boldsymbol{x_3} \cdot \cos(time) = 0 \\
f_3 &: \quad x_1 + x_2 + \boldsymbol{x_3} + 2\boldsymbol{x_4} + 4 = 0 \\
f_4 &: \quad \boldsymbol{x_3} + \boldsymbol{x_4} + 2x_5 + 2 = 0 \\
\boldsymbol{f_5} &: \quad \boldsymbol{x_4} - x_5 \cdot time - 2 \cdot time = 0
\end{aligned}
$$

With constraints for assignments:



Status and Plans with Respect to Tearing

Consideration of Solvability

Example

$$f_2: \quad x_1 = \boldsymbol{x_3} \cdot \cos(time)$$

$$f_3: \quad x_2 = -x_1 - \boldsymbol{x_3} - 2\boldsymbol{x_4} - 4$$

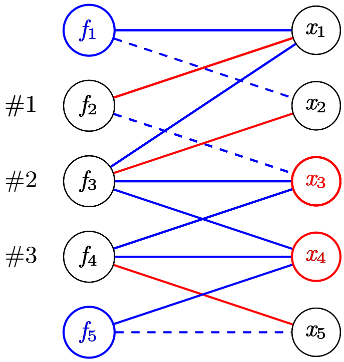
$$f_4: \quad x_5 = -\frac{\boldsymbol{x_3}}{2} - \frac{\boldsymbol{x_4}}{2} - 1$$

$$\boldsymbol{f_1}: \quad x_1 + x_2 \cdot time \stackrel{!}{=} 0 \qquad (res1)$$

$$\boldsymbol{f_5}: \quad \boldsymbol{x_4} - x_5 \cdot time - 2 \cdot time \stackrel{!}{=} 0 \qquad (res2)$$

- In general: Bigger number of tearing variables with due regard to the solvability
- In this case: x_5 would have causalised the whole system
  - $\rightarrow$ Heuristic does not find the smallest possible tearing set necessarily

With constraints for assignments:



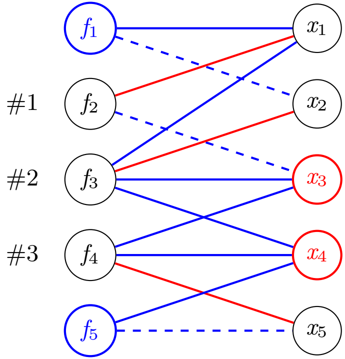
Status and Plans with Respect to Tearing

Consideration of Solvability

Problems in current implementation (To-Do-List):

- Improve performance for systems with dimensions greater than 200
  - More sophistical implementation (e.g., array hash-tables)
  - Investigate parallelization of algorithm
- Advanced solvability check
- Introduce dynamic tearing techniques

With constraints for assignments:



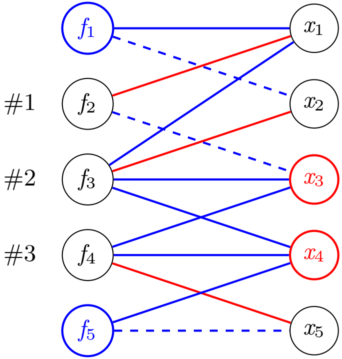
Status and Plans with Respect to Tearing

Consideration of Solvability

Problems in current implementation (To-Do-List):

- Improve performance for systems with dimensions greater than 200
  - More sophistical implementation (e.g. array hash-tables)
  - Investigate parallelization of algorithm
- Advanced solvability check
- Introduce dynamic tearing techniques

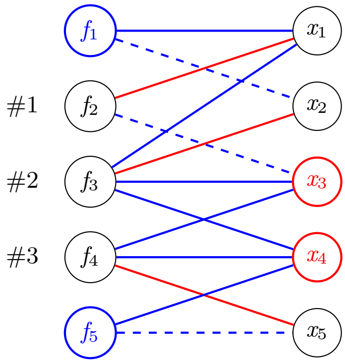
With constraints for assignments:



Problems in current implementation (To-Do-List):

- Improve performance for systems with dimensions greater than 200
  - More sophistical implementation (e.g. array hash-tables)
  - Investigate parallelization of algorithm
- Advanced solvability check
- Introduce dynamic tearing techniques

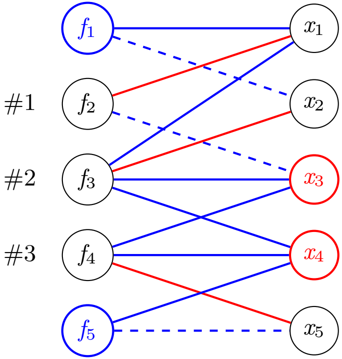
With constraints for assignments:



Problems in current implementation (To-Do-List):

- Improve performance for systems with dimensions greater than 200
  - More sophistical implementation (e.g. array hash-tables)
  - Investigate parallelization of algorithm
- Advanced solvability check
  - Make use of expressionSolve module
- Introduce dynamic tearing techniques

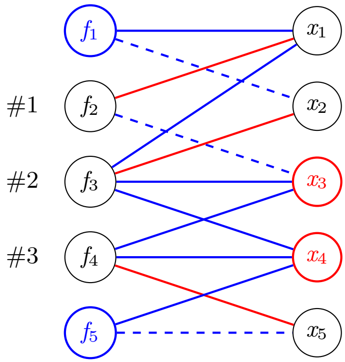
With constraints for assignments:



Problems in current implementation (To-Do-List):

- Improve performance for systems with dimensions greater than 200
  - More sophistical implementation (e.g. array hash-tables)
  - Investigate parallelization of algorithm
- Advanced solvability check
  - Make use of expressionSolve module
- Introduce dynamic tearing techniques

With constraints for assignments:



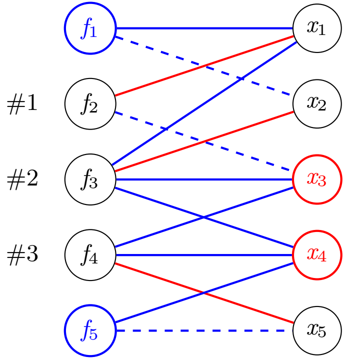
Status and Plans with Respect to Tearing

Consideration of Solvability

Problems in current implementation (To-Do-List):

- Improve performance for systems with dimensions greater than 200
  - More sophistical implementation (e.g. array hash-tables)
  - Investigate parallelization of algorithm
- Advanced solvability check
  - Make use of expressionSolve module
- Introduce dynamic tearing techniques
  - E.g. change tearing set during simulation
  - Check on division by zero during simulation
  - Investigate proper and robust switching criteria

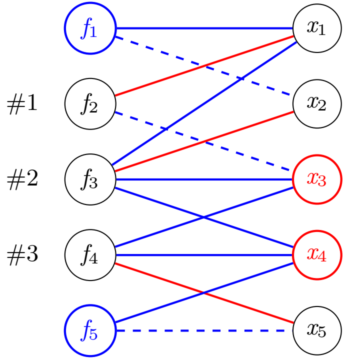
With constraints for assignments:



Problems in current implementation (To-Do-List):

- Improve performance for systems with dimensions greater than 200
  - More sophistical implementation (e.g. array hash-tables)
  - Investigate parallelization of algorithm
- Advanced solvability check
  - Make use of expressionSolve module
- Introduce dynamic tearing techniques
  - E.g. change tearing set during simulation
  - Check on division by zero during simulation
  - Investigate proper and robust switching criteria

With constraints for assignments:



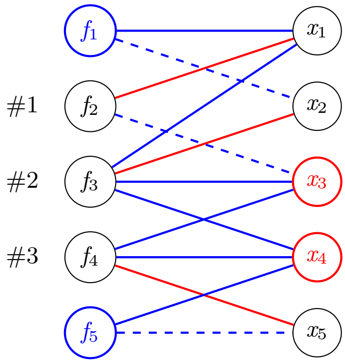
Status and Plans with Respect to Tearing

Consideration of Solvability

Problems in current implementation (To-Do-List):

- Improve performance for systems with dimensions greater than 200
  - More sophistical implementation (e.g. array hash-tables)
  - Investigate parallelization of algorithm
- Advanced solvability check
  - Make use of expressionSolve module
- Introduce dynamic tearing techniques
  - E.g. change tearing set during simulation
  - Check on division by zero during simulation
  - Investigate proper and robust switching criteria

With constraints for assignments:



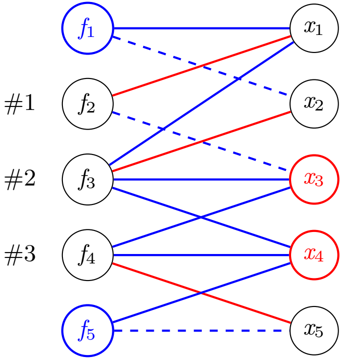
Status and Plans with Respect to Tearing

Consideration of Solvability

Problems in current implementation (To-Do-List):

- Improve performance for systems with dimensions greater than 200
  - More sophistical implementation (e.g. array hash-tables)
  - Investigate parallelization of algorithm
- Advanced solvability check
  - Make use of expressionSolve module
- Introduce dynamic tearing techniques
  - E.g. change tearing set during simulation
  - Check on division by zero during simulation
  - Investigate proper and robust switching criteria

With constraints for assignments:



First investigations with a prototype CSE module

- Preliminary functionality in OpenModelica 1.9.2beta
  - Already efficient implementation using hash-tables,
  - Option `-cseBinary`: collects all binary common subexpression,
  - Option `-cseCall`: collects all multiple function calls,
  - Option `-cseEachCall`: extracts all function calls,
  - Works only for models with real subexpressions,
  - Handles complex function calls involving return types like `tuples`, `arrays`, `etc.`,
  - Enormous performance increase detected (e.g. `Modelica.Fluid`)
- Expected effects on non-linear equation systems

Effects of Common Subexpression Elimination

Current Status and Plans

First investigations with a prototype CSE module

- Preliminary functionality in OpenModelica 1.9.2beta
  - Already efficient implementation using hash-tables,
  - Option `-cseBinary`: collects all binary common subexpression,
  - Option `-cseCall`: collects all multiple function calls,
  - Option `-cseEachCall`: extracts all function calls,
  - Works only for models with real subexpressions,
  - Handles complex function calls involving return types like `tuples`, `arrays`, `etc.`,
  - Enormous performance increase detected (e.g. `Modelica.Fluid`)
- Expected effects on non-linear equation systems

Effects of Common Subexpression Elimination

Current Status and Plans

First investigations with a prototype CSE module

- Preliminary functionality in OpenModelica 1.9.2beta
  - Already efficient implementation using hash-tables,
  - Option `-cseBinary`: collects all binary common subexpression,
  - Option `-cseCall`: collects all multiple function calls,
  - Option `-cseEachCall`: extracts all function calls,
  - Works only for models with real subexpressions,
  - Handles complex function calls involving return types like `tuples`, `arrays`, `etc.`,
  - Enormous performance increase detected (e.g. `Modelica.Fluid`)
- Expected effects on non-linear equation systems

Effects of Common Subexpression Elimination

Current Status and Plans

First investigations with a prototype CSE module

- Preliminary functionality in OpenModelica 1.9.2beta
  - Already efficient implementation using hash-tables,
  - Option `-cseBinary`: collects all binary common subexpression,
  - Option `-cseCall`: collects all multiple function calls,
  - Option `-cseEachCall`: extracts all function calls,
  - Works only for models with real subexpressions,
  - Handles complex function calls involving return types like `tuples`, `arrays`, `etc.`,
  - Enormous performance increase detected (e.g. `Modelica.Fluid`)
- Expected effects on non-linear equation systems

First investigations with a prototype CSE module

- Preliminary functionality in OpenModelica 1.9.2beta
  - Already efficient implementation using hash-tables,
  - Option `-cseBinary`: collects all binary common subexpression,
  - Option `-cseCall`: collects all multiple function calls,
  - Option `-cseEachCall`: extracts all function calls,
  - Works only for models with real subexpressions,
  - Handles complex function calls involving return types like `tuples`, `arrays`, `etc.`,
  - Enormous performance increase detected (e.g. `Modelica.Fluid`)
- Expected effects on non-linear equation systems

Effects of Common Subexpression Elimination

Current Status and Plans

First investigations with a prototype CSE module

- Preliminary functionality in OpenModelica 1.9.2beta
  - Already efficient implementation using hash-tables,
  - Option `-cseBinary`: collects all binary common subexpression,
  - Option `-cseCall`: collects all multiple function calls,
  - Option `-cseEachCall`: extracts all function calls,
  - Works only for models with real subexpressions,
  - Handles complex function calls involving return types like `tuples`, `arrays`, `etc.`,
  - Enormous performance increase detected (e.g. `Modelica.Fluid`)
- Expected effects on non-linear equation systems

Effects of Common Subexpression Elimination

FH Bielefeld
University of
Applied Sciences

First investigations with a prototype CSE module

- Preliminary functionality in OpenModelica 1.9.2beta
  - Already efficient implementation using hash-tables,
  - Option `-cseBinary`: collects all binary common subexpression,
  - Option `-cseCall`: collects all multiple function calls,
  - Option `-cseEachCall`: extracts all function calls,
  - Works only for models with real subexpressions,
  - Handles complex function calls involving return types like `tuples`, `arrays, etc.`,
  - Enormous performance increase detected (e.g. `Modelica.Fluid`)
- Expected effects on non-linear equation systems

Effects of Common Subexpression Elimination

First investigations with a prototype CSE module

- Preliminary functionality in OpenModelica 1.9.2beta
  - Already efficient implementation using hash-tables,
  - Option `-cseBinary`: collects all binary common subexpression,
  - Option `-cseCall`: collects all multiple function calls,
  - Option `-cseEachCall`: extracts all function calls,
  - Works only for models with real subexpressions,
  - Handles complex function calls involving return types like `tuples`, `arrays`, `etc.`,
  - Enormous performance increase detected (e.g. `Modelica.Fluid`)
- Expected effects on non-linear equation systems

Effects of Common Subexpression Elimination

Current Status and Plans

First investigations with a prototype CSE module

- Preliminary functionality in OpenModelica 1.9.2beta
  - Already efficient implementation using hash-tables,
  - Option `-cseBinary`: collects all binary common subexpression,
  - Option `-cseCall`: collects all multiple function calls,
  - Option `-cseEachCall`: extracts all function calls,
  - Works only for models with real subexpressions,
  - Handles complex function calls involving return types like `tuples`, `arrays`, `etc.`,
  - Enormous performance increase detected (e.g. `Modelica.Fluid`)
- Expected effects on non-linear equation systems
  - Structural change of algebraic loops
  - Less computing effort due to code motion

Effects of Common Subexpression Elimination

Current Status and Plans

First investigations with a prototype CSE module

- Preliminary functionality in OpenModelica 1.9.2beta
  - Already efficient implementation using hash-tables,
  - Option `-cseBinary`: collects all binary common subexpression,
  - Option `-cseCall`: collects all multiple function calls,
  - Option `-cseEachCall`: extracts all function calls,
  - Works only for models with real subexpressions,
  - Handles complex function calls involving return types like `tuples`, `arrays`, `etc.`,
  - Enormous performance increase detected (e.g. `Modelica.Fluid`)
- Expected effects on non-linear equation systems
  - Structural change of algebraic loops
  - Less computing effort due to code motion

First investigations with a prototype CSE module

- Preliminary functionality in OpenModelica 1.9.2beta
  - Already efficient implementation using hash-tables,
  - Option `-cseBinary`: collects all binary common subexpression,
  - Option `-cseCall`: collects all multiple function calls,
  - Option `-cseEachCall`: extracts all function calls,
  - Works only for models with real subexpressions,
  - Handles complex function calls involving return types like `tuples`, `arrays`, `etc.`,
  - Enormous performance increase detected (e.g. `Modelica.Fluid`)
- Expected effects on non-linear equation systems
  - Structural change of algebraic loops
  - Less computing effort due to code motion

Original equations

$$
\begin{aligned}
-4 + v_1 \cdot f(v_1) + v_2 &= source \\
2v_1 \cdot f(v_1) + v_2 + v_4 - v3 &= source \\
3v_1 \cdot f(v_1) - 7v_2 - 2v_3 + 3v_4 &= 0 \\
v_1 \cdot f(v_1) + v_2 - v_3 - v_4 &= 0
\end{aligned}
$$

Effects of Common Subexpression Elimination

Structural Changes of Strongly Connected Components

Original equations

$$-4 + v_1 \cdot f(v_1) + v_2 = source$$
$$2v_1 \cdot f(v_1) + v_2 + v_4 - v3 = source$$
$$3v_1 \cdot f(v_1) - 7v_2 - 2v_3 + 3v_4 = 0$$
$$v_1 \cdot f(v_1) + v_2 - v_3 - v_4 = 0$$

Equations after tearing

SCC1	Non-linear tearing variables v1, v4
	v2 := source + 4 - v1 * f(v1)
	v3 := v2 - v4 + v1 * f(v1)
	res1 := 2.0 * v1 * f(v1) + v2 + v4 - v3 - source
	res2 := 3.0 * v1 * f(v1) + -7.0 * v2 + -2.0 * v3 + 3.0 * v4

Effects of Common Subexpression Elimination

Structural Changes of Strongly Connected Components

Original equations

$$-4 + v_1 \cdot f(v_1) + v_2 = source$$
$$2v_1 \cdot f(v_1) + v_2 + v_4 - v3 = source$$
$$3v_1 \cdot f(v_1) - 7v_2 - 2v_3 + 3v_4 = 0$$
$$v_1 \cdot f(v_1) + v_2 - v_3 - v_4 = 0$$

Equations after tearing

SCC1	Non-linear tearing variables v1, v4
	v2 := source + 4 - v1 * f(v1)
	v3 := v2 - v4 + v1 * f(v1)
	res1 := 2.0 * v1 * f(v1) + v2 + v4 - v3 - source
	res2 := 3.0 * v1 * f(v1) + -7.0 * v2 + -2.0 * v3 + 3.0 * v4

Equations after common subexpression elimination and tearing

cse1 = f(v1)   cse2 = v1 * cse1

SCC1	Linear tearing variables v2, v4
	cse2 = source + 4 - v2
	v3 = cse2 + v2 - v4
	res1 := 2.0 * cse2 + v2 + v4 - v3 - source
	res2 := 3.0 * cse2 + -7.0 * v2 + -2.0 * v3 + 3.0 * v4
SCC2	Non-linear tearing variable v1
	cse1 := f(v1)
	res1 := cse2 - v1 * cse1

Effects of Common Subexpression Elimination

Performance Improvements Due to Code Motion

Original equations

$$w = f_3(x, n)$$
$$f_1(x, n) \cdot y + 1.1 \cdot f_2(x, n) \cdot \sinh(z) = 2$$
$$f_4(x, n) \cdot \sinh(y) + 1.1 \cdot f_4(x, n) \cdot z = \sinh(z)$$
$$\mathrm{der}(x) = y \cdot z$$

Effects of Common Subexpression Elimination

Performance Improvements Due to Code Motion

Original equations

$$w = f_3(x, n)$$
$$f_1(x, n) \cdot y + 1.1 \cdot f_2(x, n) \cdot \sinh(z) = 2$$
$$f_4(x, n) \cdot \sinh(y) + 1.1 \cdot f_4(x, n) \cdot z = \sinh(z)$$
$$\mathrm{der}(x) = y \cdot z$$

Equations after tearing

SCC1*	Non-linear iteration variables y, z
	res1 := f1(x, n) * y + 1.1 * f2(x, n) * sinh(z) - 2.0
	res2 := f4(x, n) * sinh(y) + 1.1 * f4(x, n) * z - sinh(z)
SCC2*	der(x) := y * z
SCC3*	w := f3(x, n)

Effects of Common Subexpression Elimination

Performance Improvements Due to Code Motion

Original equations

$$w = f_3(x, n)$$
$$f_1(x, n) \cdot y + 1.1 \cdot f_2(x, n) \cdot \sinh(z) = 2$$
$$f_4(x, n) \cdot \sinh(y) + 1.1 \cdot f_4(x, n) \cdot z = \sinh(z)$$
$$\mathrm{der}(x) = y \cdot z$$

Equations after tearing

SCC1*	Non-linear iteration variables y, z res1 := f1(x, n) * y + 1.1 * f2(x, n) * sinh(z) - 2.0 res2 := f4(x, n) * sinh(y) + 1.1 * f4(x, n) * z - sinh(z)
SCC2*	der(x) := y * z
SCC3*	w := f3(x, n)

Equations after code motion and tearing

SCC1#	cse1 = f1(x, n)
SCC2#	cse2 = f2(x, n)
SCC3#	cse4 = f4(x, n)
SCC4#	Non-linear iteration variables y, z cse5 :=sinh(y) cse6 :=sinh(z) res1 := cse1 * y + 1.1 * cse2 * cse6 - 2.0 res2 := cse4 * cse5 + 1.1 * cse4 * z - cse6
SCC5#	der(x) = y * z
SCC6#	cse3 = f3(x, n)
SCC7#	w = cse3

Outlook

Success consists of going from failure to failure without loss of enthusiasm.

*Winston Churchill *1874 †1965*