Initialisation of Models with Dynamic State-Selection

Karim Abdelhak, Bernhard Bachmann

University of Applied Sciences Bielefeld
Bielefeld, Germany

1. Introduction

Task

Problem

If a system gets modelled with to many degrees of freedom it results into a *singular* system of equations and can not be simulated.

Method of Resolution

Some equations, so called *constraint equations*, will be differentiated with pantelides algorithm to resolve the singularity.

Task

Problem

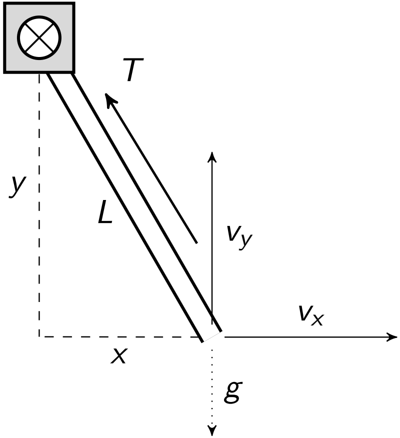
If a system gets modelled with to many degrees of freedom it results into a *singular* system of equations and can not be simulated.

Method of Resolution

Some equations, so called *constraint equations*, will be differentiated with pantelides algorithm to resolve the singularity.

Example: Planar Pendulum

Layout



Constants:

- L: Length of the Pendulum
- g: Gravitational Acceleration

Variables:

- T: Tension
- x: Horizontal Position
- y: Vertical Position
- v_x: Horizontal Velocity
- v_y: Vertical Velocity

Figure: Schematic Representation of the Planar Pendulum.

Example: Planar Pendulum

System Differential-Algebraic Equations

$$
\begin{aligned}
\dot{x} &= v_x && (f_1)\\
\dot{y} &= v_y && (f_2)\\
\dot{v}_x &= Tx && (f_3)\\
\dot{v}_y &= Ty - g && (f_4)\\
0 &= x^2 + y^2 - L^2 && (f_5)
\end{aligned}
$$

2. Index Reduction

StateSets

Definition

StateSet

Sets of n constraint equations containing a maximum of $n-1$ unknowns get detected during pantelides algorithm. These sets of equations each form a *StateSet*.

Abbreviation	Explanation
I	Unique Index
E	Constraint Eqations
C	Candidates for the State Selection
J	Jacobian-Matrix $\delta E/\delta C$

Figure: Additional *StateSet*-Information

Example: Planar Pendulum

StateSets

Constraint Equations:

$$0 = x^2 + y^2 - L^2 \quad (f_5)$$

$$0 = 2xv_x + 2yv_y \quad (f_5')$$

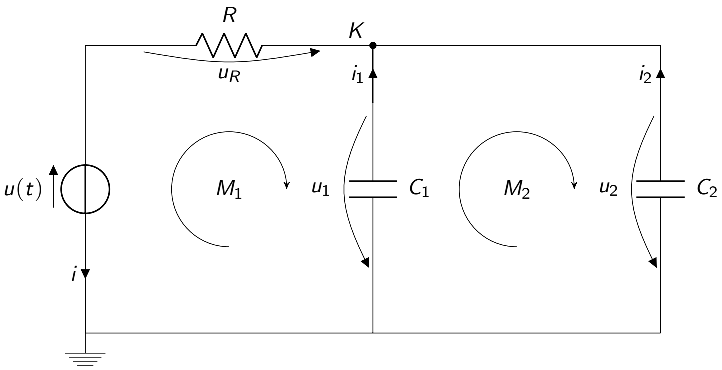
StateSets:

	$Set1$	$Set2$
I	1	2
E	(f_5)	(f_5')
C	(x, y)	(v_x, v_y)
J	$[2x, 2y]$	$[2x, 2y]$

3. Initialisation

Overconstraint

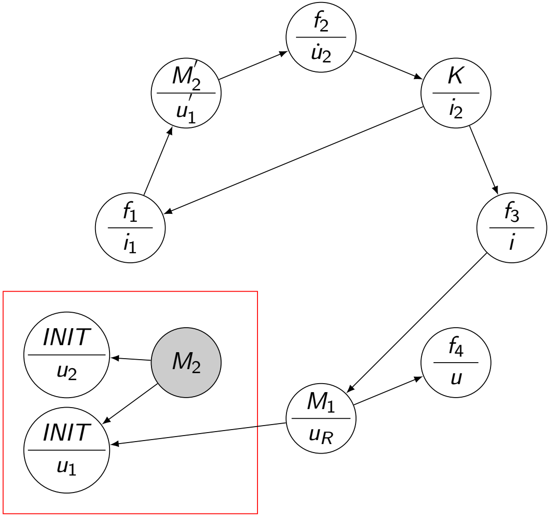
Example: Simple Circuit



$$
\begin{aligned}
0 &= C_1 \cdot \dot{u}_1 - i_1 && (f_1)\\
0 &= C_2 \cdot \dot{u}_2 - i_2 && (f_2)\\
0 &= R \cdot i - u_R && (f_3)\\
0 &= \sin(t) - u && (f_4)\\
0 &= i - i_1 - i_2 && (K)\\
0 &= u + u_R + u_1 && (M_1)\\
0 &= u_2 - u_1 && (M_2)
\end{aligned}
$$

Overconstraint

Example: Simple Circuit



Overconstraint

$$
\begin{aligned}
0 &= C_1 \cdot \dot{u}_1 - i_1 && (f_1)\\
0 &= C_2 \cdot \dot{u}_2 - i_2 && (f_2)\\
0 &= R \cdot i - u_R && (f_3)\\
0 &= \sin(t) - u && (f_4)\\
0 &= i - i_1 - i_2 && (K)\\
0 &= u + u_R + u_1 && (M_1)\\
0 &= u_2 - u_1 && (M_2)\\
0 &= \dot{u}_2 - \dot{u}_1 && (M_2')
\end{aligned}
$$

Underconstraint

1. Matching the system, but preferring all non-state variables. (uses matching of simulation system as base)
2. All states which are not matched get fixed and therefore initial equations are generated.
3. If there are unmatched equations, the system was modelled with redundant initial equations. Continue with processing the newly formed overdetermination.

StateSet-Dependencies

System of *DAEs* Containing at Least One *StateSet*

$$0 = f(x, y, z)$$
$$0 = f'(x, y, z, \dot{x}, \dot{y})$$
$$0 = g(x, y, z, \dot{x}, \dot{y})$$

	Set
E	(f)
C	(x, y)
J	$\delta f / \delta(x, y)$

- **f** Constraint equations
- **g** All other equations of the system
- **x** States contained in the constraint equations, will become *StateCandidates*
- **y** Algebraic variables contained in the constraint equations, will become *StateCandidates*
- **z** All other variables contained in the system, will **not** become *StateCandidates*

StateSet-Dependencies

System of *DAEs* Containing at Least One *StateSet*

$$0 = f(x, y, z)$$
$$0 = f'(x, y, z, \dot{x}, \dot{y})$$
$$0 = g(x, y, z, \dot{x}, \dot{y})$$

	Set
E	(f)
C	(x, y)
J	$\delta f / \delta(x, y)$

Crossdependent	*Set.J* depends on $z \wedge z$ is connected to another *StateSet*
Selfdependent	*Set.J* depends on x or y
Independent	Neither *cross*- nor *selfdependent*

Example: Planar Pendulum

Dependencies

	$Set1$	$Set2$
I	1	2
E	(f_5)	(f_5')
C	(x, y)	(v_x, v_y)
J	$[2x, 2y]$	$[2x, 2y]$

Dependencies

- Set1 Selfdependent: $Set1.J$ contains x and y
- Set2 Crossdependent: $Set2.J$ contains x and y

Problems with Dynamic State Selection

Problem

By the time of initialisation it is unclear which candidates are actual states. Therefore it is unclear, which candidates need to be initialised.

Method

Each *StateSet* will be processed individually. All constraint equations and initial equations for each *StateSet* will be solved for all candidates during the initialisation.

Criteria

1. The square submatrix of the *jacobian matrix* containing all constraint equations derived by all candidates, which are not initialised, must be non-singular.
2. The algorithm for full pivoting the jacobian matrix to find this submatrix must not be inside an algebraic loop.

Problems with Dynamic State Selection

Problem

By the time of initialisation it is unclear which candidates are actual states. Therefore it is unclear, which candidates need to be initialised.

Method

Each *StateSet* will be processed individually. All constraint equations and initial equations for each *StateSet* will be solved for all candidates during the initialisation.

Criteria

1. The square submatrix of the *jacobian matrix* containing all constraint equations derived by all candidates, which are not initialised, must be non-singular.
2. The algorithm for full pivoting the jacobian matrix to find this submatrix must not be inside an algebraic loop.

Problems with Dynamic State Selection

Problem

By the time of initialisation it is unclear which candidates are actual states. Therefore it is unclear, which candidates need to be initialised.

Method

Each *StateSet* will be processed individually. All constraint equations and initial equations for each *StateSet* will be solved for all candidates during the initialisation.

Criteria

1. The square submatrix of the *jacobian matrix* containing all constraint equations derived by all candidates, which are not initialised, must be non-singular.
2. The algorithm for full pivoting the jacobian matrix to find this submatrix must not be inside an algebraic loop.

Splitting of the Remaining Equations

Lemma

An algebraic loop during the process of initialisation containing constraint equations of a StateSet can exclusively contain other constraint equations of the same StateSet.

Splitting of the Remaining Equations

Lemma

An algebraic loop during the process of initialisation containing constraint equations of a StateSet can exclusively contain other constraint equations of the same StateSet.

Implication of this Lemma

The remaining equations g and variables z can each be split up into two sets. Those which can be solved exclusively *before* (g_B, z_B) and those which can be solved exclusively *after* (g_A, z_A) the candidates of given *StateSet*.

Splitting of the Remaining Equations

Implication of this Lemma

The remaining equations g and variables z can each be split up into two sets. Those which can be solved exclusively *before* (g_B, z_B) and those which can be solved exclusively *after* (g_A, z_A) the candidates of given *StateSet*.

$$
\begin{aligned}
0 &= f(x, y, z_B) \\
0 &= f^{'}(x, y, z_B, z_A\dot{x}, \dot{y}) \\
0 &= g_A(x, y, z_B, z_A, \dot{x}, \dot{y}) \\
0 &= g_B(z_B)
\end{aligned}
$$

Cases for each *StateSet*

Number of actual states to be initialised

$$|\chi| = |x| + |y| - |f|$$

Well Defined (Case I)	$\|\chi\| = \|Set.Fix\| \wedge$ remaining subset of $Set.J$ is non-singular
Underconstraint (Case II)	$\|\chi\| > \|Set.Fix\|$
Overconstraint (Case III)	$\|\chi\| < \|Set.Fix\|$

Underconstraint (Case II)

Additional Equations

Initial Equations

$$Set.V[j] = \sum_{i=1}^{|Set.C|} Set.A[j,i] \cdot Set.C_{i,Start}, \quad \forall j = |Set.Fix| \ldots |\chi|$$

$$Set.init(x, y, Set.F)$$

$$Set.pivot(x, y, z_U, Set.A, Set.F)$$

$$HEU(Set.F)$$

Underconstraint (Case II)

Additional Equations

Initial Equations

$$\sum_{i=1}^{|Set.C|} Set.F[j,i] \cdot C_i = \sum_{i=1}^{|Set.C|} Set.F[j,i] \cdot C_{i,Start}, \quad \forall j = |Set.Fix| \ldots |\chi| \qquad (Set.init)$$

$$Set.init(x, y, Set.F)$$

$$Set.pivot(x, y, z_U, Set.A, Set.F)$$

$$HEU(Set.F)$$

Underconstraint (Case II)

Additional Equations

Initial Equations

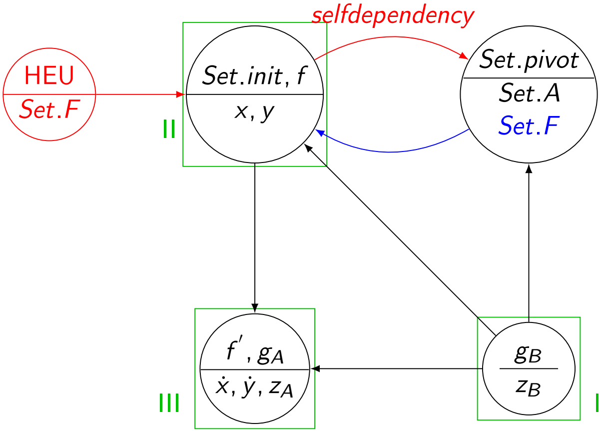
$$\sum_{i=1}^{|Set.C|} Set.F[j,i] \cdot C_i = \sum_{i=1}^{|Set.C|} Set.F[j,i] \cdot C_{i,Start}, \quad \forall j = |Set.Fix| \ldots |\chi| \qquad (Set.init)$$

$$Set.init(x, y, Set.F)$$

$$Set.pivot(x, y, z_U, Set.A, Set.F)$$

$$HEU(Set.F)$$

Digraph for the Process of Initialisation



Digraph for the Process of Initialisation

Example: Planar Pendulum, Set1

	$Set1$
I	1
E	(f_5)
C	(x, y)
J	$[2x, 2y]$

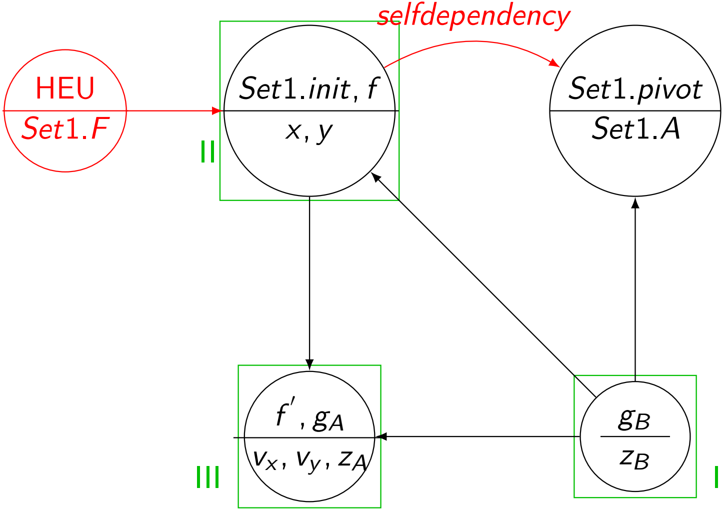
$$g_B, z_B = \emptyset$$

$$f = (f_5)$$

$$f' = (f_5')$$

$$g_A = (f_1, f_2, f_3, f_4, f_5'', set2.init, set2.pivot)$$

$$z_A = (Set2.A, Set2.F, T, \dot{x}, \dot{y}, \dot{v}_x, \dot{v}_y)$$

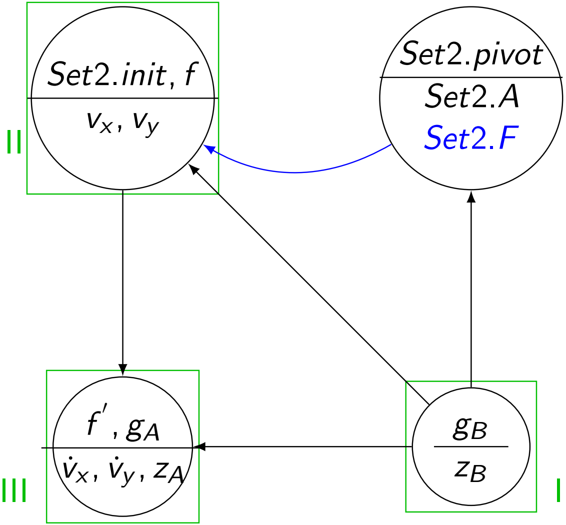


Digraph for the Process of Initialisation

Example: Planar Pendulum, Set2

	$Set2$
I	2
E	(f_5')
C	(v_x, v_y)
J	$[2x, 2y]$

$$g_B = (Set1.pivot, Set1.init, f_5)$$
$$z_B = (x, y, Set1.A, Set1.F)$$
$$f = (f_5')$$
$$f' = (f_5'')$$
$$g_A = (f_1, f_2, f_3, f_4)$$
$$z_A = (T, \dot{x}, \dot{y})$$



Heuristic

Example

$$0 = 2xv_x + 2yv_y \qquad (f_5')$$

If x resolves to be zero, this equation can not be solved for v_x, likewise for y and v_y.

Method

1. For every candidate count the number of elements containing other candidates appearing in the corresponding row of the jacobian matrix.
2. Chose the candidates with the highest count until enough states are fixed.

Heuristic

Example

$$0 = 2xv_x + 2yv_y \qquad (f_5')$$

If x resolves to be zero, this equation can not be solved for v_x, likewise for y and v_y.

Method

1. For every candidate count the number of elements containing other candidates appearing in the corresponding row of the jacobian matrix.
2. Chose the candidates with the highest count until enough states are fixed.

Overconstraint (Case III)

Method

For each overconstraint dynamic *StateSet* do:

1. Symbolical consistency check (like *static* index reduction)
2. Check if the jacobian matrix is singular (underconstraint)
3. Merge all initial and constraint equation nodes
4. Merge all candidate nodes
5. Match the equation and candidate node

Continue checking the full system as if it was reduced by *static* index reduction.

Summary

Already Implemented

Local Branch: The Heuristic to find the most likely candidates to be initialised

Future Implementation

Better handling of non-selfdependent *StateSets* as described

Further Ideas

1. Improve the handling of parameter-dependent *StateSets*
2. Use *StateSets*-Dependencies during simulation

Summary

Already Implemented

Local Branch: The Heuristic to find the most likely candidates to be initialised

Future Implementation

Better handling of non-selfdependent *StateSets* as described

Further Ideas

1. Improve the handling of parameter-dependent *StateSets*
2. Use *StateSets*-Dependencies during simulation

Thank you for your attention!