Symbolic Elimination Techniques for Dynamic Optimization

Fredrik Magnusson

Department of Automatic Control
Faculty of Engineering
Lund University, Sweden

April 1, 2016

Outline

Outline

1. Overview
2. Tearing
3. Results

Me

- Started February 2012
- Working on JModelica.org: Open-source framework for large-scale dynamic optimization
- Looking for ways to make dynamic optimization algorithms more
  - efficient
  - reliable
  - accessible

Dynamic optimization

- Optimization problems with differential equations as constraints
- Applications include
  - optimal control (open or closed loop)
  - parameter estimation or optimization
  - state estimation (moving horizon estimation)
  - experiment design

JModelica.org

Optimal control

$$
\begin{array}{ll}
\text{minimize} & \phi(t_f, x(t_f)) + \int_0^{t_f} L(x(t), u(t))\,\mathrm{d}t, \\
\text{with respect to} & t_f, x, u, \\
\text{subject to} & \dot{x} = f(x(t), u(t)), \\
& x(0) = x_0, \\
& g_i(t, x(t), u(t)) \leq 0, \\
& \psi(x(t_f)) = 0, \\
& \forall t \in [0, t_f].
\end{array}
$$

Optimal control with DAE

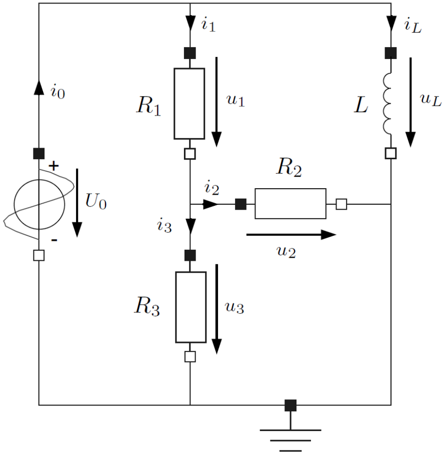
Differential-algebraic equation (DAE) instead of explicit ODE:

$$
\begin{aligned}
\text{minimize} \quad & \phi(t_f, x(t_f), y(t_f)) + \int_0^{t_f} L(x(t), y(t), u(t))\,\mathrm{d}t, \\
\text{with respect to} \quad & t_f, x, y, u, \\
\text{subject to} \quad & F(\dot{x}(t), x(t), y(t), u(t)) = 0, \\
& x(0) = x_0, \\
& g_i(t, x(t), y(t), u(t)) \leq 0, \\
& \psi(x(t_f), y(t_f)) = 0, \\
& \forall t \in [0, t_f].
\end{aligned}
$$

Simulation of DAEs

- DAE systems can be simulated with specialized DAE solvers
- Common to instead transform (reduce) the DAE to an ODE and apply ODE solvers
- These transformations have many benefits, but a few drawbacks

Example



DAE

$$U_0 = \sin(t)$$
$$u_1 = R_1 \cdot i_1$$
$$u_2 = R_2 \cdot i_2$$
$$u_3 = R_3 \cdot i_3$$
$$u_L = L \cdot \frac{\mathrm{d}}{\mathrm{d}t} i_L$$
$$U_0 = u_1 + u_3$$
$$u_L = u_1 + u_2$$
$$u_3 = u_2$$
$$i_0 = i_1 + i_L$$
$$i_1 = i_2 + i_3$$

ODE

$$\frac{\mathrm{d}}{\mathrm{d}t} i_L = \frac{\sin(t)}{L}$$

Transformation techniques

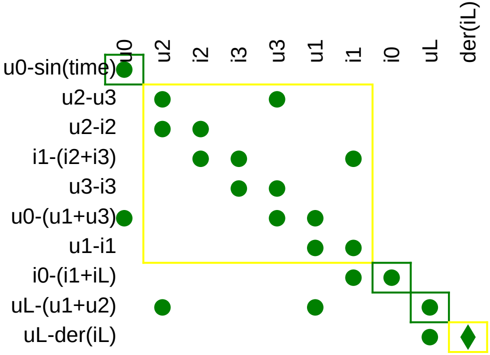
Going from DAE to ODE in general involves many steps:

1. Alias elimination	Get rid of equations and corresponding variables of the form $x \pm y = 0$
2. Index reduction	Perform index reduction until DAE is index 1 (dummy derivatives)
3. Matching	Match variables and equations (Hopcroft-Karp)
4. BLT	Transform the system to block-lower triangular (BLT) form with blocks of minimal size (Tarjan)
5. Tearing	Tear algebraic loops
6. Solve loops	Solve algebraic loops (Newton or LU)

Steps 1 and 5 are optional and done for performance.

BLT example

The block-lower triangular (BLT) transformation is central. Example:



- Allows state derivatives $\dot{x}$ and algebraic variables y to be solved for sequentially (in terms of state x and input u), resulting in ODE
- Non-scalar and/or nonlinear blocks require numerical treatment

Causalization for dynamic optimization

- DAE-constrained optimization traditionally done using full DAE
- Research idea: Utilize some of the transformation techniques for DAE simulation for optimization
- For simulation, goal is to get equivalent ODE
- My goal is instead to get the equivalent (reduced) DAE that is most suitable for numerical optimization

Elimination techniques

1. Alias elimination	Get rid of equations and corresponding variables of the form $x \pm y = 0$
2. Index reduction	Perform index reduction until DAE is index 1 (dummy derivatives)
3. Matching	Match variables and equations (Hopcroft-Karp)
4. BLT	Transform the system to block-lower triangular (BLT) form with blocks of minimal size (Tarjan)
5. Tearing	Tear algebraic loops
~~6. Solve loops~~	Solve algebraic loops (Newton or LU)
7. Sparsity preservation	Undo some parts of steps 4 and 5 to preserve sparsity

Outline

1. Overview
2. Tearing
3. Results

Linear tearing

- Used to improve efficiency when solving $\bar{A}x = b$ when $\bar{A}$ is sparse but without significant structure
- Permute $\bar{A}$ to get

$$\begin{bmatrix} A & B \\ C & D \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} b_1 \\ b_2 \end{bmatrix}$$

  such that A is easy to invert and D is small

- Solution is then

$$x_2 = \left(D - CA^{-1}B\right)^{-1} \left(b_2 - CA^{-1}b_1\right)$$
$$x_1 = A^{-1} \left(b_1 - B\left(D - CA^{-1}B\right)^{-1} \left(b_2 - CA^{-1}b_1\right)\right)$$

- $D - CA^{-1}B$ is the Schur complement of block A

Linear tearing in BLT

- In the BLT form, we use linear tearing for linear, non-scalar blocks
- Make A lower triangular (invert by forward substitution)
- Terminology:

  Causalized variables $x_1 \in \mathbb{R}^{n_c}$

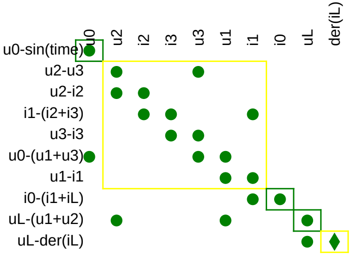
  Tearing variables $x_2 \in \mathbb{R}^{n_t}$

  Causalized equations $Ax_1 + Bx_2 = b_1 \in \mathbb{R}^{n_c}$

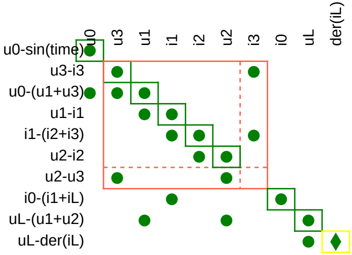
  Tearing residuals $Cx_1 + Dx_2 = b_2 \in \mathbb{R}^{n_t}$
- Inverting $\bar{A}$ has cost $\mathcal{O}((n_c + n_t)^3)$
- Using Schur complement when A is lower triangular, cost is instead $\mathcal{O}(n_c^2 n_t + n_c n_t^2 + n_t^3)$
- Got rid of $\mathcal{O}(n_c^3) \implies$ we want few tearing variables

Tearing example

BLT without tearing



BLT with tearing



Selecting tearing variables and residuals

- Finding a minimal set of tearing variables and residuals such that A is lower triangular is NP-hard
- However, some choices of tearing variables/residuals will cause ill-conditioned Schur complement!
- So even if it were tractable to minimize n_t, would often be bad
- Selection either by heuristic algorithms, or manually by experts
- The automatic tearing in Dymola is a trade secret and one of the major reasons of its success

Nonlinear tearing

- For nonlinear systems $F(x) = 0$, the main idea is the same
- Tear to create partition

$$
\begin{array}{cc}
 & \begin{array}{cc} \quad\quad\quad x_1 \quad\quad\quad & \quad\quad x_2 \quad\quad \end{array} \\
\begin{array}{c} F_1 \\ \\ \\ \\ F_2 \end{array} &
\left[\begin{array}{cccc|ccc}
1 & 0 & \cdots & 0 & * & \cdots & * \\
* & 1 & \cdots & 0 & * & \cdots & * \\
\vdots & \vdots & \ddots & 0 & \vdots & \ddots & \vdots \\
* & * & * & 1 & * & \cdots & * \\
\hline
* & * & \cdots & * & * & \cdots & * \\
\vdots & \vdots & \ddots & \vdots & \vdots & \ddots & \vdots \\
* & * & \cdots & * & * & \cdots & *
\end{array}\right],
\end{array}
$$

where F_1 is lower triangular and constant along diagonal

- F_1 nonlinear w.r.t. x_1, but easy to invert by forward substitution

Skipping algebraic loops

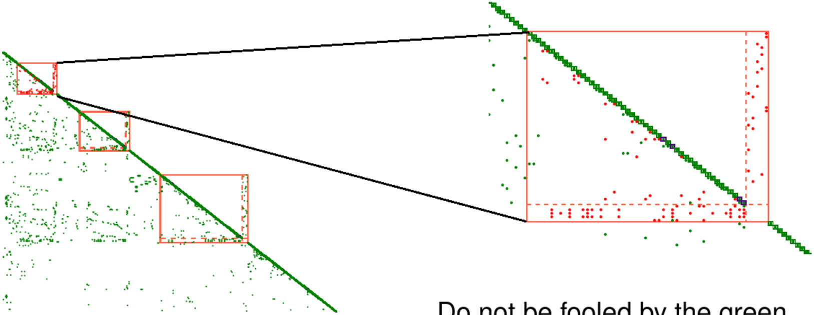
- When transforming all the way to an ODE, we need to numerically solve the tearing residuals
    - LU decomposition for linear blocks, Newton for nonlinear blocks
- The resulting ODE is thus not on closed form, and takes a long time to evaluate the right-hand side of
- Simply leave the tearing residuals, yielding a smaller DAE with cheap residuals
- Consequently no point in solving for the state derivatives $\Longrightarrow$ always choose all state derivatives as tearing variables

Sparsity preservation

- Resulting DAE is much smaller, but usually much denser
- Sometimes the resulting density is crippling for optimization
- Should thus also consider sparsity when tearing
- As far as I know, this is previously unexplored/unpublished territory for dynamic systems (even for simulation)
- Nice ideas from dynamic pivot selection in direct methods for sparse matrices (Markowitz criterion and local minimum fill-in)

A real example

A real BLT form where tearing is important



Do not be fooled by the green landscape! Nonlinearities in the state variables.

Outline

1. Overview
2. Tearing
3. Results

Old Results

Results from 1 year ago without tearing and sparsity preservation

Problem		n_x	n_y	Time [s]
ST-WF	Full	13	27	3.7
	Reduced	13	4	2.0
CCPP	Full	10	123	2.3
	Reduced	10	1	0.9
Dist. Col.	Full	125	1000	12
	Reduced	125	2	94

All of these problems lack algebraic loops $\Longrightarrow$ no use for tearing. But maybe sparsity preservation helps!

New results

Some new problems to demonstrate effects of tearing

Problem		n_x	n_y	Time [s]
Dist. Col.	Full	125	1000	12
	Reduced	125	~~2~~ 63	~~94~~ 4
Fourbar1	Full	2	452	37
	Reduced	2	46	2
HRSG	Full	18	75	8.4
	Reduced	18	14	3.6
Dbl. Pend.	Full	4	124	∞
	Reduced	4	7	3.0

Conclusion

Elimination techniques for dynamic optimization:

- Eliminate algebraic variables by identifying BLT structure and tear non-scalar blocks
- Keep some choice algebraic variables to preserve sparsity
- Employing these techniques reduces solution time by a factor between 2 and ∞
- I have yet to encounter a problem where a suitable combination of these ideas do more harm than good
- No other dynamic optimization software utilizes these techniques

The end

Thank you for listening!

The End