

# Differential-Algebraic Equations (DAEs)

L. T. Biegler  
Chemical Engineering Department  
Carnegie Mellon University  
Pittsburgh, PA 15213  
biegler@cmu.edu  
<http://dynopt.cheme.cmu.edu>

May 11, 2000

Introduction

Simple Examples of DAE Systems

Definition of Index

Problems of High Index Systems

Accuracy, Stability, Consistent Initial Conditions

Properties of DAE Solvers

Reformulation of DAE Systems

Process Examples

References and Software

# Introduction

Start with ODE system:  $y' (=dy/dt) = f(y, t), y(0) = y_0$

Here we expect an evolution of  $y$  in time and there are a number of methods that ensure an accurate and stable evolution.

Formulating pure ODE problems in engineering often requires the combination of

- conservation laws (mass and energy balance),
- constitutive equations (equations of state, pressure drops, heat transfer...)
- design constraints (desired operations...)

Implementation of these is often easier and much more efficient by keeping these relations separate. This leads to a set of differential and algebraic equations (DAEs):

$$F(y, y', t) = 0 \text{ with } y(0) = y_0 \quad \text{Fully Implicit}$$

Classes of these problems include:

$$Ay' + f(y,t) = 0 \text{ with } y(0) = y_0 \quad \text{Linear Implicit}$$

$$\begin{aligned} x' &= f(x, z, t) \\ g(x, z, t) &= 0 \end{aligned} \quad \text{Semi-explicit}$$

where:

$x$  - differential variables

$z$  - algebraic variables,  $y^T = [x^T \ z^T]$ ,

For consistency, we consider the semi-explicit form only.

DAEs are solved using extensions of ODE solvers.

Two approaches:

### 1. Nested Approach

- given  $x_n$ , solve  $g(x_n, z_n) = 0 \implies z_n(x_n)$
- using ODE method,  
evolve  $x_{n+1} = \Phi(x_n, z_n(x_n), t_n)$

This is the most common approach:

- requires  $z = z(x)$  (implicit function)
- required if only an explicit method is available (e.g., explicit Euler or Runge-Kutta)
- can be expensive due to inner iterations

### 2. Simultaneous Approach

Solve  $x' = f(x, z, t)$ ,  $g(x, z, t) = 0$  simultaneously using an implicit solver to evolve both  $x$  and  $z$  in time.

- requires an implicit solver
- much more efficient
- provides for more flexible problem specification (!)

How is this done?

Consider a BDF solver. For a semi-explicit system, we can write:

$$\begin{aligned} \mathbf{x}_{n+1} &= h \beta_{-1} f(\mathbf{x}_{n+1}, \mathbf{z}_{n+1}, t_n) + \sum_{j=0, k} \alpha_j \mathbf{x}_{n-j} \\ g(\mathbf{x}_{n+1}, \mathbf{z}_{n+1}, t_{n+1}) &= 0 \end{aligned}$$

and we can solve this system for  $\mathbf{x}_{n+1}, \mathbf{z}_{n+1}$  using Newton's method. At iteration  $l$ :

$$\begin{bmatrix} \mathbf{I} - h\beta_{-1} \frac{\partial f}{\partial \mathbf{x}} & -h\beta_{-1} \frac{\partial f}{\partial \mathbf{z}} \\ \frac{\partial g}{\partial \mathbf{x}} & \frac{\partial g}{\partial \mathbf{z}} \end{bmatrix} \begin{bmatrix} \Delta \mathbf{x}_{n+1} \\ \Delta \mathbf{z}_{n+1} \end{bmatrix} = - \begin{bmatrix} \mathbf{x}_{n+1}^l - \sum_{j=0}^k \alpha_j \mathbf{x}_{n-j}^l - h\beta_{-1} f(\mathbf{x}_{n+1}^l, \mathbf{z}_{n+1}^l, t_{n+1}) \\ g(\mathbf{x}_{n+1}^l, \mathbf{z}_{n+1}^l, t_{n+1}) \end{bmatrix}$$

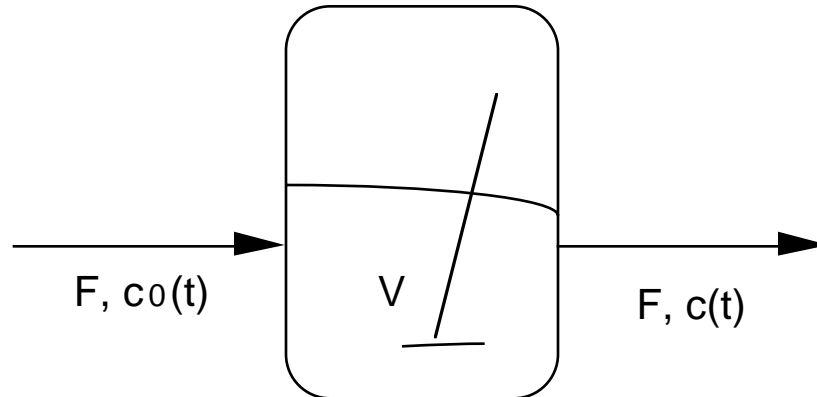
and note that the Jacobian matrix is nonsingular at  $h = 0$  as long as  $\frac{\partial g}{\partial \mathbf{z}}$  is nonsingular (a necessary condition for the implicit function  $\mathbf{z}(\mathbf{x})$ )

Thus if  $\frac{\partial g}{\partial \mathbf{z}}$  is nonsingular, both the nested and simultaneous approaches should work.

What if  $\frac{\partial g}{\partial \mathbf{z}}$  is singular?

## Simple Examples of DAEs

Example: Mixing Tank



$$c' = (c_0(t) - c(t))/\tau \quad \text{where } \tau = V/F$$

Consider two cases:

i)  $c_0(t) = \gamma(t)$  specified

$$c' = (\gamma(t) - c(t))/\tau \quad c(0) \text{ specified}$$

Simple ODE system or DAE system with  $c_0$  as algebraic variable.

$$c' = (c_0(t) - c(t))/\tau$$

$$c_0(t) - \gamma(t) = 0$$

ii)  $c(t) = \gamma(t)$

$$c' = (c_0(t) - c(t))/\tau$$

$$c(t) - \gamma(t) = 0$$

Solution is:

$$c_0(t) = \gamma(t) + \gamma(t)' \tau$$

and also:

$$c(0) = \gamma(0)$$

$$c_0(0) = \gamma(0) + \gamma(0)' \tau$$

Note:

- in case ii,  $\frac{\partial g}{\partial z}$  is singular
- $c(0)$  is given by  $\gamma(0)$ , there is no freedom to specify it.

If we didn't know the solution and specified it, this would fail:

Apply implicit Euler, for first step,  $t_1$ :

$$c(t_1) - c(0) - h (c_0(t_1) - c(t_1))/\tau = 0$$
$$c(t_1) - \gamma(t_1) = 0$$

or, after rearrangement:

$$c_0(t_1) = (\gamma(t_1) - c(0)) \tau/h + \gamma(t_1)$$

as  $h \rightarrow 0$  and  $\gamma(0) \neq c(0)$ , then  $c_0(t_1)$  blows up because the initial conditions are inconsistent.

What is the difference between these two cases?

Some additional simple examples:

Consider the (linear implicit) DAE system:

$$E y' = A y + \gamma(t) \text{ with consistent initial conditions}$$

and apply implicit Euler:

$$E(y_{n+1} - y_n)/h = A y_{n+1} + \gamma(t_{n+1})$$

and rearrangement gives:

$$y_{n+1} = (E - A h)^{-1} [E y_n + h \gamma(t_{n+1})]$$

Now the true solution,  $y(t_n)$ , satisfies:

$$E[(y(t_{n+1}) - y(t_n))/h + h y''(\xi)/2] = A y(t_{n+1}) + \gamma(t_{n+1})$$

and defining  $e_n = y(t_n) - y_n$ , we have:

$$\begin{aligned} e_{n+1} &= (E - A h)^{-1} [E e_n - h^2 y''(\xi)/2] \\ e_0 &= 0, \text{ known initial conditions} \end{aligned}$$

Consider three cases:

a) Set  $E = I$  (pure ODE system)

$$\text{for first step: } e_1 = (I - A h)^{-1} [-h^2 y''(\xi)/2] = O(h^2)$$

corresponds to local error for implicit Euler method

b) Singular algebraic equations

$$\text{Set } A = I, E = \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \text{ so that } \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} y' = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix} y + \begin{bmatrix} g_1 \\ g_2 \end{bmatrix}$$

$$\text{and } y_2 = -g_2(t), y_1 = -g_2(t)' - g_1(t)$$

After rearrangement:

$$e_1 = h^2/2 \begin{bmatrix} 0 & -1/h \\ 0 & 0 \end{bmatrix} \begin{bmatrix} 0 & 1 \\ 0 & 0 \end{bmatrix} \begin{bmatrix} y_1'' \\ y_2'' \end{bmatrix} = \begin{bmatrix} h g_2''/2 \\ 0 \end{bmatrix} = O(h)$$

c) More singular algebraic equations (Upper Hessenberg)

$$\text{Set } A = I, E = \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \text{ so that } \begin{bmatrix} 0 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} y' = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} y + \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix}$$

$$\text{and } y_3 = g(t), y_2 = g'(t), y_1 = g''(t)$$

After rearrangement:

$$e_1 = (E-hA)^{-1} E h^2 y''/2 = \begin{bmatrix} (g'' + h g''')/2 \\ h g''/2 \\ 0 \end{bmatrix} = O(1)$$

Notes:

- choosing a larger upper Hessenberg system could lead to errors of  $O(1/h)$
- how can we recognize problematic DAEs
- what causes them in engineering problems?

## Classification of DAEs

Petzold (1982) defined the difficulty of solving a DAE system in terms of nilpotency (degree of singularity) of *matrix pencils*  $(A - \lambda E)$ .

For semi-explicit systems, this corresponds to the definition of an index

Definition: For the semi-explicit DAE system:

$$\begin{aligned}x' &= f(x, z, t) \\g(x, z, t) &= 0\end{aligned}$$

the index is the minimum number of times  $g(x, z, t)$  must be differentiated with respect to time in order to yield a pure ODE system:

$$\begin{aligned}x' &= f(x, z, t) \\z' &= s(x, z, t)\end{aligned}$$

Example 1:

$$y_2' = y_1 + \gamma_1(t) \qquad 0 = y_2 + \gamma_2(t)$$

Differentiate  $g(x, z)$  and substitute ODE:

$$y_2' = -\gamma_2'(t) = y_1 + \gamma_1(t) \implies y_1 = -\gamma_2'(t) - \gamma_1(t)$$

Differentiate again to get pure ODE system:

$$y_1' = -\gamma_2''(t) - \gamma_1'(t) \implies \text{Index } 2$$

Example 2:

$$y_2' = y_1 \quad y_3' = y_2 \quad 0 = y_3 - \gamma(t)$$

Differentiate  $g(x, z)$  and substitute ODE:

$$y_3' = \gamma'(t) = y_2 \implies y_2 = \gamma'(t)$$

Differentiate again:

$$y_2' = \gamma''(t) \implies \gamma''(t) = y_1$$

Differentiate yet again to get pure ODE system:

$$y_1' = -\gamma'''(t) \implies \text{Index 3}$$

Example 3: Mixing tank

$$c' = (c_0(t) - c(t))/\tau$$

$$c(t) - \gamma(t) = 0$$

Differentiate  $g(x, z)$  and substitute ODE:

$$c' = \gamma'(t) = (c_0(t) - c(t))/\tau$$

$$c_0(t) = (\tau \gamma'(t) + c(t))$$

Differentiate again:

$$c_0'(t) = (\tau \gamma''(t) + \gamma'(t)) \implies \text{Index 2}$$

Note that increasing index restricts the selection of initial conditions!

## Numerical Problems with High Index Systems

1. Order Reduction - which methods are affected?
2. Stability Properties?
3. Linear Algebra for BDF and IRK Jacobian.
4. Inconsistent Initial Conditions

To understand each of these points better, let's consider properties of these methods.

### Order Reduction

Brenan et al. (1989) discuss a number of high index cases along with their impact on a number of different methods. A short (and loosely stated) summary follows:

- Implicit Euler methods loses order of accuracy with each increase in index
- For BDF methods,

For index 2, BDF is convergent and globally accurate to  $O(h^k)$  but these require tight Newton solutions accurate to  $O(h^{k+1})$ .

For upper Hessenberg index 3 with fixed  $h$ , BDF is convergent and globally accurate to  $O(h^k)$  but these require tight Newton solutions accurate to  $O(h^{k+2})$ .

- For other LMS methods, generally same criteria as BDF for semi-explicit DAEs
- For Runge-Kutta methods,
 

more complicated conditions are needed. For index  $p$ , order of R-K methods is given by:

$$k = \min(k_d, k_{a,i} - p + 2)$$

where  $k_d$  is the order of the ODE solver,  $k_{a,i}$  is the order determined by error tests (nontrivial for index  $i \leq p$ ). This also requires tight convergence of the Newton solver.

### Stability

- BDF and other LMS methods are stable for index 2 DAE systems and upper Hessenberg index 3 (with some exceptions).
- Runge-Kutta methods defined by the Butcher block:

$$\begin{array}{c|c} \mathbf{c} & \mathbf{A} \\ \hline & \mathbf{b} \end{array}$$

can be made stable (with some exceptions) for index 2 systems provided that:

$$r = |1 - \mathbf{b}^T \mathbf{A} \mathbf{c}| < 1$$

L-stable systems (where  $r = 0$ ) are preferred. Tight Newton tolerance is required for nonlinear systems.

## Linear Algebra for BDF and IRK Jacobian

For high index systems, the matrix:

$$\begin{bmatrix} I - h\beta_1 \frac{\partial f}{\partial x} & -h\beta_1 \frac{\partial f}{\partial z} \\ \frac{\partial g}{\partial x} & \frac{\partial g}{\partial z} \end{bmatrix}$$

(and related IRK matrices) become singular as  $h \rightarrow 0$ . Many DAE solvers use a singular value decomposition option to decompose these matrices.

### Inconsistent Initial Conditions

- All of the above properties hold only if the initial conditions for  $x(0)$  and  $z(0)$  are consistent. Otherwise, the DAE solver is likely to fail, especially for small initial  $h$ .
- Because this is the key issue for solving DAE systems, it is important that the user supply consistent initial conditions.
- For index one systems:  
 $\frac{\partial g}{\partial z}$  is nonsingular, one can specify  $x(0)$  freely and solve for  $z(0)$  from  $g(x(0), z(0), 0) = 0$ .
- For higher index systems:  
 $\frac{\partial g}{\partial z}$  is singular,  $x(0)$  cannot be specified freely and we need to analyze and reformulate the DAE to get consistent initial conditions.

## Reformulation and Solution of DAEs

Finding consistent initial conditions and solving the DAE system are closely related problems. An abstract way of looking at this problem is to consider:

$$\begin{aligned}x' &= f(x, z, t) \implies F(y, y', t) = 0 \\g(x, z, t) &= 0\end{aligned}$$

Differentiating  $p$  times, where  $p$  is the index leads to:

$$\begin{aligned}F(y, y', t) &= 0 \\F'(y, y', y'', t) &= 0 \\F''(y, y', y'', y''', t) &= 0 \\&\text{-----} \\F^p(y, y', y'', y''', \dots, y^{p+1}, t) &= 0\end{aligned}$$

These are known as the derivative array equations. This is an underdetermined system of equations.

- Solving this system for  $t = 0$  yields consistent initial conditions
- Solving this system for each time step leads to DAE solution.
- Solution to the derivative array equations contains a unique vector for  $(x, x', z)$ .

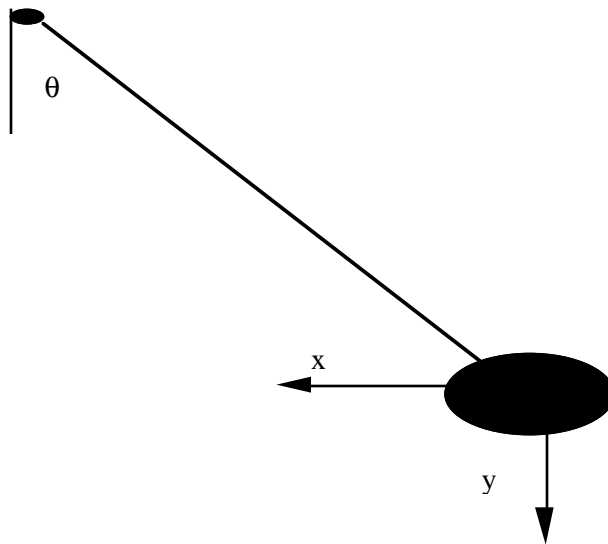
These equations can also be reformulated (through repeated substitution of  $y'$ ) to get:

(1) $y' = \phi(y, t)$	Pure ODE system
(2) $h(y, t) = 0$	State Invariants

Most reformulation methods work with the derivative array equations or the reformulated pure ODE system and invariants. We can classify these methods as:

- A. Direct differentiation to form  $y' = \phi(y, t)$  in (1)
- B. Discover (2) through differentiation and substitution of DAE system
- C. Form Augmented System of DAE
- D. Solve derivative array equations directly.
  - Structural Approach
  - Successive Linear Programming Approach
  - Gauss-Newton-Marquardt methods with SVD.

## Example - Pendulum problem



$$\begin{aligned}x' &= u \\y' &= v \\u' &= -Tx \\v' &= g - Ty \\x^2 + y^2 &= 1\end{aligned}$$

1) Find index

$$\begin{aligned}\text{i)} \quad x^2 + y^2 &= 1 \\2xx' + 2yy' &= 0 \\2xu + 2yv &= 0\end{aligned}$$

$$\begin{aligned}\text{ii)} \quad 2(xu' + x'u) + 2(yv' + vy') &= 0 \\2(u^2 + v^2) + 2(-Tx^2 + gy - Ty^2) &= 0 \\2(u^2 + v^2) - 2T(x^2 + y^2) + 2gy &= 0\end{aligned}$$

$$\begin{aligned}\text{iii)} \quad 4(uu' + vv') - 4T(xx' + yy') + 2gy' &= 2T'(x^2 + y^2) \\4(-Tux - Tyv + gv) - 4T(xu + yv) + 2gv &= 2T'\end{aligned}$$

==> (Index 3)

So the pure ODE's (1) are:

$$\begin{aligned}x' &= u \\y' &= v \\u' &= -Tx \\v' &= g - Ty \\T' &= 4T(xu + yv) + 3gv\end{aligned}$$

with invariants (2) :

$$\begin{aligned}x^2 + y^2 &= 1 \\2xu + 2yv &= 0 \\2(u^2 + v^2) - 2T(x^2 + y^2) + 2gy &= 0\end{aligned}$$

Consider first approach:

1) need to find  $x(0)$ ,  $y(0)$ ,  $u(0)$ ,  $v(0)$ ,  $T(0)$  from (2) in order to solve (1). This is an over-determined system and it is not clear how to get this.

2) Because of  $p=3$  differentiations (1) is the same if we had

$$x^2 + y^2 = c_1 + c_2t + c_3t^2$$

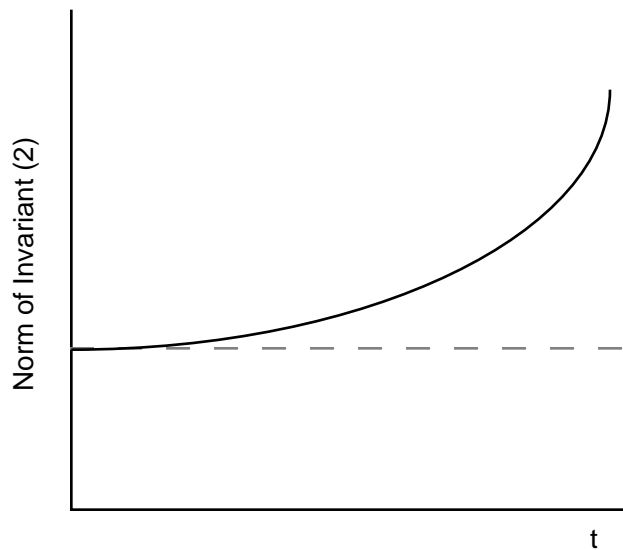
(or in general if  $g(y,z) = \sum_{i=0, p-1} c_i t^i$ )

Note:

Even if we solve  $(x(0)^2 + y(0)^2 = 1)$  we will have a drift problem

$$x(t)^2 + y(t)^2 = 1 + \varepsilon_1 + \varepsilon_2 t + \varepsilon_3 t^2$$

due to roundoff error  $\implies$  algebraic equations will not be satisfied.



Second Approach - retain all algebraic equations, add invariants through differentiation, drop ODE's (Bachmann et al., 1990)

### General idea

- 1) Check if DAE system is index 1. If yes, stop.
- 2) Else, identify Algebraic Equations (AEs) that solve a subset of Algebraic Variables (AVs).
- 3) Differentiate remaining Algebraic Equations with respect to time. This leads to  $y'$  terms in the invariant Algebraic Equation.
- 4) Substitute right hand side of ODE for  $y'$  into invariant Algebraic Equation. Go to 1.

## Pendulum Example Revisited:

$$\begin{aligned}x' &= u \\y' &= v \\u' &= -Tx \\v' &= g - Ty \\x^2 + y^2 &= 1\end{aligned}$$

Differentiate AE

$$2x x' + 2y y' = 0 \quad (\text{solves for } x')$$

Replace  $x' = u$  with  $2xu + 2yv = 0$

$$\begin{aligned}y' &= v \\u' &= -Tx \\v' &= g - Ty \\x^2 + y^2 &= 1 && (\text{now } x \text{ is an AV}) \\2xu + 2yv &= 0 && (\text{need to solve for } T)\end{aligned}$$

differentiate AE

$$\begin{aligned}2(x'u + xu') + 2(y'v + yv') &= 0 && (\text{solves for } u') \\2(u^2 + xu') + 2(v^2 + yv') &= 0 \\ \implies 2(u^2 - Tx^2) + 2(v^2 + gy - Ty^2) &= 0 && (*)\end{aligned}$$

Replace  $u' = -Tx$  by (\*)

$$\begin{aligned}y' &= v \\v' &= g - Ty \\x^2 + y^2 &= 1 \\2xu + 2yv &= 0 \\(u^2 + v^2) - T(x^2 + y^2) + gy &= 0\end{aligned}$$

This system is index 1.

Note:

- The IC's here are obvious since we can specify  $y(0)$ ,  $v(0)$  and solve for  $x(0)$ ,  $u(0)$ ,  $T(0)$ ,  $y'(0)$ ,  $v'(0)$  (index 1).
- Reformulation is not unique. We could have chosen  $y'$ ,  $v'$  instead of  $x'$ ,  $u'$ .
- Note the relations  $x' - u$ ,  $u' + Tx$  are not observed directly. They are implicitly satisfied in the invariants but not at same level as  $y$  and  $v$ . (not sure what the accuracy is for  $x$ ,  $u$ ).

## Algebra can get messy...

Matson & Soderlind (1993) simplified the algebra by introducing dummy variables.

$$\begin{array}{l} x' = xp \\ y' = yp \\ u' = up \\ \hline v' = vp \\ xp = u \\ yp = v \\ up = -Tx \\ vp = g - Ty \\ x^2 + y^2 = 1 \end{array}$$

This leads to

$$2(x xp + y yp) = 0 \implies \text{replace } x' = xp \\ \{2(xu + yv) = 0\}$$

$$2(x up + xp u) + 2(yp v + y vp) = 0 \\ \implies \text{replace } u' = up \\ \{2(-Tx^2 + u^2) + 2(v^2 + yg - Ty^2) = 0\}$$

System becomes:

$$\begin{array}{l} y' = yp \\ v' = vp \\ \hline xp = u \\ yp = v \\ up = -Tx \\ vp = g - Ty \\ x^2 + y^2 = 1 \\ x(xu + yv) = 0 \\ (u^2 + v^2 + yg) = T(x^2 + y^2) \end{array}$$

A similar structural decomposition based on incidence matrices and graph partitioning was developed by Pantelides (1988) and used in SPEEDUP and gProms.

### Third approach

Start with full ODE plus invariants

$$y' = \phi(y, t) \quad (1)$$

$$h(y) = 0 \quad (2)$$

These invariants (2) must characterize the solution manifold and  $\frac{\partial h^T}{\partial y}$  must be full rank. These must be obtained from some other procedure.

Form augmented system

$$y' = \phi(y, t) + \frac{\partial h}{\partial y} \lambda$$

$$h(y) = 0$$

where  $\lambda$  are Lagrange multipliers that keep the systems on the  $h(y) = 0$  manifold. Here we need

To see this, differentiate invariant along  $h(y,t) = 0$  to get:

$$\frac{\partial h^T}{\partial y} y' + \frac{\partial h}{\partial t} = 0 \text{ which implies } \frac{\partial h^T}{\partial y} \phi(y, t) + \frac{\partial h}{\partial t} = 0.$$

Multiplying augmented system gives:

$$\frac{\partial h^T}{\partial y} (y' - \phi(y, t)) = \frac{\partial h^T}{\partial y} \frac{\partial h}{\partial y} \lambda = 0$$

and since  $\frac{\partial h^T}{\partial y} \frac{\partial h}{\partial y}$  is nonsingular (full rank assumption) we have  $\lambda = 0$ .

Augmented system is index 2 and needs to be solved using BDF solver.

## Pendulum Example Yet Again:

Pure ODE system:  $y' = \phi(y, t)$

$$\begin{aligned}x' &= u \\y' &= v \\u' &= -Tx \\v' &= g - Ty \\T' &= 4T(xu + yv) + 3gv\end{aligned}$$

Invariants:  $h(y) = 0$

$$\begin{aligned}x^2 + y^2 &= 1 \\2xu + 2yv &= 0 \\(u^2 + v^2) - T(x^2 + y^2) + gy &= 0\end{aligned}$$

Augmented (index 2) system:  $y' = \phi(y, t) + \frac{\partial h}{\partial y} \lambda$

$$\begin{aligned}x' &= u + 2x \lambda_1 + 2u \lambda_1 - 2 T x \lambda_3 \\y' &= v + 2y \lambda_1 + 2v \lambda_1 - 2 T y \lambda_3 \\u' &= -Tx + 2x \lambda_1 + 2 u \lambda_3 \\v' &= g - Ty + 2y \lambda_1 + 2 v \lambda_3 \\T' &= 4T(xu + yv) + 3gv - 2(x^2 + y^2) \lambda_3\end{aligned}$$

$$\begin{aligned}x^2 + y^2 &= 1 \\2xu + 2yv &= 0 \\(u^2 + v^2) - T(x^2 + y^2) + gy &= 0\end{aligned}$$

Initialize with  $\lambda(0) = 0$

## Fourth Approach

Solve derivative array equations directly.

### Structural Approach (Chung and Westerberg)

- Develop derivative array equations

$$F(y, y', t) = 0$$

$$F'(y, y', y'', t) = 0$$

$$F''(y, y', y'', y''', t) = 0$$

-----

$$F^p(y, y', y'', y''', \dots, y^{p+1}, t) = 0$$

- Analyze Incidence Matrix of complete system
- Choose pivoting variables among  $(x, x', y)$  from among all equations.
- Solve with this set at  $t=0$  to find  $y(0)$ .

### Successive Linear Programming Approach (Gopal and Biegler)

- Define derivative array equations:  
 $G(y, y', \dots, y^{p+1}, t) = 0$
- Solve  $\text{Min } \|G(y, y', \dots, y^{p+1}, 0)\|_1$  using SLP

### Gauss-Newton-Maquardt methods with SVD. (Campbell and Moore)

- Define derivative array equations:  
 $G(y, y', \dots, y^{p+1}, t) = 0$
- Solve  $\text{Min } \|G(y, y', \dots, y^{p+1}, 0)\|_2$  using SVD

Note:

First and second approaches emphasize solution of pure ODEs (1) or invariants (2), respectively.

Third and fourth approaches attempt to get 'balanced' solutions of (1) and (2).

### Process Examples

Where do high index DAE systems arise?

- converting simulation problems to control problems
- making 'reasonable' simplifications in process model

In these cases we need to:

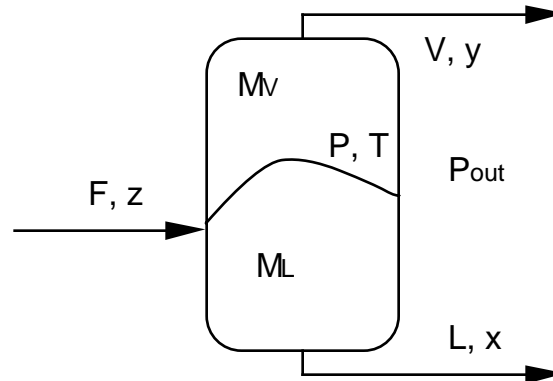
Analyze DAE to get index, and possibly identify invariant equations (2).

Obtain consistent initial conditions for  $x(0)$  and  $z(0)$ .

To do this we can use either the

- Second Approach  
useful for analytic reformulation of DAE
- Fourth Approach  
requires implementation of algorithmic solver for derivative array equations (not implemented in commercial tools)

## Flash examples



Specify temperature and tank volume

$$M_i' = F z_i - V y_i - L x_i$$

$$M_T' = F - V - L$$

$$M_i = M_L x_i + M_V y_i$$

$$M_T = M_L + M_V$$

$$y_i = K_i x_i$$

$$f^{K_i}(T, P, x) - K_i = 0$$

$$V = M_L / \rho_L + M_V / \rho_V$$

$$f^L(T, P, x) - \rho_L = 0$$

$$f^V(T, P, y) - \rho_V = 0$$

$$\sum_i x_i - \sum_i y_i = 0$$

$$L - g^L(T, P - P_{out}, x) = 0$$

$$V - g^V(T, P - P_{out}, y) = 0$$

Specified: T, F, z, V, P<sub>out</sub>

N + 1 Differential Equations and Variables: M<sub>i</sub>, M<sub>T</sub>

3N + 7 Algebraic Equations & Variables: y, x, K, P, V,

L, ρ<sub>L</sub>, ρ<sub>V</sub>, M<sub>L</sub>, M<sub>V</sub>

We can find an output set for all of the algebraic and differential variables ==> **Index 1**

Now assume that  $M_V$  is negligible. Then we can specify both temperature and pressure (instead of volume) and this leads to the following equations:

$$\begin{aligned} M_i' &= F z_i - V y_i - L x_i \\ M_L' &= F - V - L \\ M_i &= M_L x_i \\ y_i &= K_i x_i \\ f^{K_i}(T, P, x) - K_i &= 0 \\ \sum_i x_i - \sum_i y_i &= 0 \\ L - g^L(M_L) &= 0 \end{aligned}$$

Specified: T, P, F, z

N + 1 Differential Equations and Variables:  $M_i, M_L$

3N + 2 Algebraic Equations & Variables: y, x, K, V, L

We cannot find an output set for all of the algebraic and differential variables. What about V?

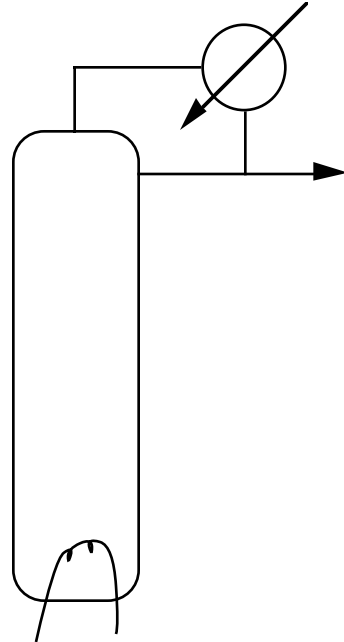
$$\sum_i (K_i - 1)x_i = 0$$

Differentiate once:

$$\begin{aligned} \sum_i \left( \frac{\partial K_i}{\partial x_i} x_i + K_i - 1 \right) x_i' &= 0 \text{ and substitute} \\ x_i' &= (M_i/M_L)' = (M_i' M_L - M_i M_L') / M_L^2 \\ &= ((Fz_i - Lx_i - Vy_i)M_L - M_i (F - L - V)) / M_L^2 \end{aligned}$$

**==> Index 2**

## Batch Distillation Example



Negligible Vapor Holdup  
Pressure and pressure drop specified  
Stages numbered from top  
NC Components

### Condenser (2 + NC)

$$\begin{aligned}M_c' &= V_1 - L_0 - D \\M_c x_{D,i}' &= V_1 (y_{1,i} - x_{D,i}) \\M_c h_{D,i}' &= V_1 (H_1 - h_D) - Q_c\end{aligned}$$

### Trays, $j = 1, NT, NT$ (2 + NC) Equations

$$\begin{aligned}M_j' &= V_{j+1} + L_{j-1} - (V_j + L_j) \\M_j x_{j,i}' &= V_{j+1}(y_{j+1,i} - x_{j,i}) + L_{j-1,i}(x_{j-1,i} - x_{j,i}) - V_j(y_{j,i} - x_{j,i}) \\M_j h_j' &= V_{j+1} (H_{j+1} - h_j) + L_{j-1} (h_{j-1} - h_j) - V_j (H_j - h_j)\end{aligned}$$

### Condenser (2 + NC)

$$\begin{aligned}M_r' &= L_{NT} - V_r \\M_r x_{r,i}' &= L_{NT} (x_{NT,i} - x_{r,i}) - V_r (y_{r,i} - x_{r,i}) \\M_c h_D' &= L_{NT} (h_{NT} - h_r) - V_r (H_r - h_r)\end{aligned}$$

## Algebraic Equations

$$\begin{aligned}
 y_{j,i} &= K_i(T_j, P_j, x_j) x_{j,i} \\
 \sum_i y_{j,i} &= 1 \quad \sum_i y_{r,i} = 1 \\
 L_i - g^L(M_i) &= 0 \\
 D+L_0 - g^L(M_c) &= 0 \\
 L_j - g^L(M_i) &= 0 \\
 h_j - f^L(T_j, P_j, x_j) &= 0 \\
 H_j - f^V(T_j, P_j, y_j) &= 0 \\
 h_r - f^L(T_r, P_r, x_r) &= 0 \\
 H_r - f^V(T_r, P_r, y_r) &= 0 \\
 h_D - f^L(T_D, P_D, x_D) &= 0
 \end{aligned}$$

Specified:  $L_0, Q_c, Q_r, P$

4 + 2 NC + 2 NT + NT NC Differential Equations and  
Variables:  $M_c, x_D, h_D, M_j, x_j, h_j, M_r, x_r, h_r$

6 + NC + 4 NT + NT NC Algebraic Equations and  
Variables:  $L_j, D, h_D, V_j, y_j, H_j, V_r, y_r, H_r$

However,  $V_j$ , and  $V_r$  cannot be calculated from:

$$\sum_i y_{j,i} = 0 \quad \sum_i y_{r,i} = 0.$$

This is an index 2 problem and we need to differentiate these equations:

$$\begin{aligned}
 (\sum_i y_{j,i})' &= (\sum_i K_{j,i} x_{j,i})' \\
 &= \sum_i (K_{j,i} x'_{j,i} + x_{j,i} \frac{\partial K_{j,i}}{\partial x_{j,i}} x'_{j,i} + x_{j,i} \frac{\partial K_{j,i}}{\partial T_j} T_j')
 \end{aligned}$$

$$T_j' = -\sum_i (K_{j,i} x'_{j,i} + x_{j,i} \frac{\partial K_{j,i}}{\partial x_{j,i}} x'_{j,i}) / (\sum_i x_{j,i} \frac{\partial K_{j,i}}{\partial T_j})$$

Now we have:

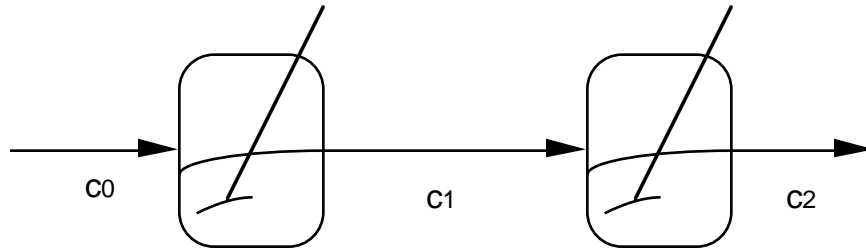
$$\begin{aligned}h_j' &= \frac{\partial h_j^T}{\partial x_j} x_j' + \frac{\partial h_j}{\partial T_j} T_j' \\ &= (V_{j+1} (H_{j+1} - h_j) + L_{j-1} (h_{j-1} - h_j) - V_j (H_j - h_j)) / M_j\end{aligned}$$

Substitute right hand sides from  $T_j'$  and  $x_j'$  from above and this leads to a new algebraic equation that contains  $V_j$

Equation for  $V_r$  follows similarly and the resulting system is **Index 1**.

## Dynamic CSTR Examples

Consider two CSTRs in series



$$c_1' = (c_0 - c_1)/\tau + R(c_1)$$

$$c_2' = (c_1 - c_2)/\tau + R(c_2)$$

and specify the output  $c_2(t) = K$ .

The problem is not index 1

differentiating once:

$$c_2'(t) = 0 = (c_1 - c_2)/\tau + R(c_2)$$

This replaces the ODE for  $c_2'$ :

$$c_1' = (c_0 - c_1)/\tau + R(c_1)$$

$$(c_1 - c_2)/\tau + R(c_2) = 0$$

$$c_2(t) = K.$$

differentiating twice:

$$(c_1' - c_2')/\tau + \frac{\partial R}{\partial c_2} c_2' = 0 \text{ or}$$

$$c_1' = c_2' - \tau \frac{\partial R}{\partial c_2} c_2' = (1 - \tau \frac{\partial R}{\partial c_2}) [(c_1 - c_2)/\tau + R(c_2)]$$

Replace ODE for  $c_1$  with an algebraic equation:

$$(c_0 - c_1)/\tau + R(c_1) = (1 - \tau \frac{\partial R}{\partial c_2}) [(c_1 - c_2)/\tau + R(c_2)]$$

Differentiating a third time gives an ODE in  $c_0$

The final system is transformed from index 3 to an (algebraic) index 1 system.

$$(c_0 - c_1)/\tau + R(c_1) = (1 - \tau \frac{\partial R}{\partial c_2}) [(c_1 - c_2)/\tau + R(c_2)]$$
$$(c_1 - c_2)/\tau + R(c_2) = 0$$
$$c_2(t) = K.$$

There is no freedom in specification of initial conditions

Notes:

- Generalizing to  $N$  CSTRs with an output specification leads to an index  $N+1$  system and reformulation leads to a purely algebraic system
- Replacing the output specification with a feedback controller:  $c_2(t) - K = f(c_0(t))$  leads to an index 1 system.

## References

Reference books on DAEs include:

Ascher, U. M. and L. R. Petzold, Computer Methods For Ordinary Differential Equations And Differential-Algebraic Equations, SIAM, 1999

Brenan, K., S. Campbell and L. Petzold, Numerical Solution of Initial Value Problems in Differential Algebraic Equations, North Holland, 1989, republished by SIAM, 1996

There are several papers on reformulation that are specific to process engineering including:

Bachmann, R., Bruell, L., T. Mrziglod and U. Pallaske, "On methods of reducing the index of differential-algebraic equations," *Comp. Chem. Engr.*, 14, p. 1271 (1990)

Campbell, S. and E. Moore, "Progress on a general method for nonlinear, higher index DAEs II," *Circuits Systems Signal Process.*, 13, p. 123 (1994)

Chung, Y. and A. W. Westerberg, "A proposed numerical algorithm for solving nonlinear index problems," *I & EC Research*, 29, p. 1234 (1990)

Gopal, V. and L. T. Biegler, "An Optimization Approach for Consistent Initialization and Reinitialization after Discontinuities of Differential Algebraic Equations," *SIAM J. Sci. Comput.*, 20, 9, p. 447 (1998)

Mattson, S. and G. Soderlind, "Index reduction in differential algebraic systems using dummy derivatives," *SIAM J. Sci. Comput.*, 14, p. 677 (1993)

Pantelides, C. C., "The consistent initialization of differential-algebraic equations," *SIAM J. Sci. Comput.*, 7, p. 720 (1988)

# Software

## Free Net-based Codes

<http://www.netlib.org>

**file**    **coldae.f**

by     Ascher and Spiteri  
for    semi-explicit differential-algebraic equations with index at most 2.  
alg    collocation, proj on constraint manifold (modification of colnew)  
prec   double

**file**    **ddasrt.f**

by     Petzold  
for    stiff differential-algebraic system solver with root stopping  
alg    backward differentiation formulae  
prec   double  
rel    good  
age    stable  
gams   I1a2

**file**    **ddassl.f**

by     Petzold  
for    stiff differential-algebraic system solver  
alg    backward differentiation formulae  
prec   double  
rel    good  
age    stable  
gams   I1a2

**file**    **mebdfdae**

by     Cash <j.cash@ic.ac.uk>  
for    stiff ODE and DAE initial-value problems  
alg    extended backward differentiation formulae  
prec   double  
lang   Fortran  
gams   I1a2

**file** **dgelda.tar.gz**  
for general linear differential algebraic equations  
by P. Kunkel, V. Mehrmann, W. Rath and J. Weickert  
# The package includes a computation of all the local invariants of the  
, system, a regularization procedure and an index reduction scheme  
and  
, it can be combined with every solution method for standard index 1  
, systems. Nonuniqueness and inconsistencies are treated in a least  
, square sense. In our package we have implemented backward  
, differentiation formulas (BDF) methods and Runge-Kutta schemes.  
prec double  
lang fortran  
gams 11a2

<http://www-users.cs.umn.edu/~petzold/DASPK/>

DASPK Package - 1995 Revision, P. N. Brown, A. C. Hindmarsh, L. R. Petzold

DASPK is a solver [1] for systems of differential-algebraic equations. It includes options for both direct and iterative (Krylov) methods for the solution of the linear systems arising at each (implicit) time step.

The 1995 revision to DASPK includes a completely new procedure [2] for calculating consistent initial conditions for a large class of problems (which includes semi-explicit index-1 systems). This procedure includes options for inequality constraints on selected components. The package also includes a new option to omit the algebraic components from the local error control.

Along with the solver itself, the DASPK package includes four example programs and a set of general-purpose preconditioner files. These are described in more detail below. The package includes separate single and double precision versions of all source files.

<http://www.zib.de/nowak/limex4.html>

LIMEX is an extrapolation integrator for the solution of linearly-implicit differential-algebraic systems of the form

$$B(t,y) * y'(t) = f(t,y)$$

with B a (n,n)-matrix of rank less or equal n.

Overview of current versions:

4.1A1 Non-sparse dense or banded Jacobians.

Direct solvers: LAPACK routines. Based on BLAS and LAPACK routines.

4.1A2 Non-sparse dense or banded Jacobians.

Direct solvers: NAG routines. Based on NAG routines (Mark 16).

4.1B1 Sparse Jacobians. Direct solvers: MA28 routines from the Harwell library, iterative solvers: GMRES and BICGSTAB with a variable ILU preconditioner. Based on BLAS and LAPACK routines.

4.1B2 Sparse Jacobians. Direct solvers: NAG routines, iterative solvers: GMRES and BICGSTAB with a variable ILU preconditioner. Based on NAG routines (Mark 16).

Version 4.1B1 and 4.1B2 are available on request.

<ftp://ftp.unige.ch/pub/doc/math/stiff/radau5.f>

RADAU5 implicit Runge-Kutta method of order 5 (Radau IIA) for problems of the form  $My'=f(x,y)$  with possibly singular matrix M; with dense output (collocation solution). Concerning the linear algebra routines the user has the choice to link the program either with DC\_DECSOL and DECSOL or with DC\_LAPACK and LAPACK and LAPACKC

## **NaG Library**

//D02NVF// is a setup routine which must be called by the user, prior to an integrator in the D02M-D02N subchapter, if Backward Differentiation Formulae (BDF) are to be used.

## **Other Codes**

DASOLV - LSODI BDF code with sparse Jacobians, (Imperial College, C. Pantelides)

DSL48s - DASSL BDF code with sparse Jacobians using MA48 from Harwell library, (P. Barton, MIT)

SDASAC - DASSL BDF code with sparse Jacobians using MA28 from Harwell library, (M. Mangold, MPI, Magdeburg)