

Solving DAEs by Taylor Series: Part I

NED NEDIALKOV

Department of Computing and Software
McMaster University
nedialk@mcmaster.ca

and

JOHN PRYCE

Computer Information Systems Engineering Dept.
Cranfield University, RMCS Shrivenham, UK
pryce@rmcs.cranfield.ac.uk

Workshop on Taylor Models
17-20 December 2003, Miami, Florida

The Problem

Solve numerically the initial-value problem (IVP) for a system of differential-algebraic equations (DAEs) with

n equations f_i in

n dependent variables $x_j = x_j(t)$

of the form

$$f_i(t, \text{the } x_j \text{ and derivatives of them}) = 0.$$

- Derivatives of order > 1 are allowed.
- Fully implicit; e.g. f_i may contain expressions such as

$$(x_1' x_2'')^2, (\ln(t + x_1 x_2'))'', \text{ etc.}$$

Informally,

- the **index** of a DAE is the minimum number of differentiations needed to reduce the DAE to an explicit ordinary differential equation (ODE);
- the higher the index of a DAE, the more difficult it is to solve it.
An explicit ODE is an index-0 DAE.

Current methods are **restricted to index ≤ 3** DAEs, usually in the form

$$f(t, x, x') = 0.$$

Solver	Author(s)	Index
DASSL	Petzold	≤ 1
MEBDFDAE	Cash	≤ 3
PSIDE	De Swart, Lioen, & Van der Veen	≤ 3
RADAU/RADAU5	Hairer & Wanner	≤ 3

We do not impose restrictions on the index and the order of a DAE.

Approach:

- Pryce's structural analysis + Taylor series

Advantages:

- can solve DAEs directly in a compact form, without reducing to first-order, lower-index form
- sparsity can be exploited
- high-accuracy is readily attainable

Disadvantages:

- explicit method—not efficient on very stiff problems; some progress on Hermite-Obreschkoff methods
- complex machinery is involved

References

J. Pryce, *A Simple Structural Analysis Method for DAEs*, BIT, 41(2), pp. 364–394.

N. S. Nedialkov and J. D. Pryce, *Solving Differential-Algebraic Equations by Taylor Series (I): Computing Coefficients*, submitted to BIT.

URL: www.cas.mcmaster.ca/~nedialk/PAPERS

Acknowledgments

Ole Stauning (Denmark) provided invaluable help with his automatic differentiation (AD) packages FADBAD and TADIFF.

Andreas Wächter (IBM T. J. Watson) provided assistance with his optimization package IPOPT.

Outline

- I. Theoretical Background
- II. The Integration Process
- III. Software

I. Theoretical Background

“DAEs aren’t ODEs” (Linda Petzold)

As well as obvious algebraic constraints they typically have “hidden constraints”.

Our structural analysis method (related to that of C.C. Pantelides) automatically takes care of this.

Pryce's Structural Analysis

1. Form the $n \times n$ matrix $\Sigma = (\sigma_{ij})$ where

$$\sigma_{ij} = \begin{cases} \text{order of derivative to which} \\ x_j \text{ occurs in } f_i; \\ -\infty \text{ if it does not occur.} \end{cases}$$

2. Find a Highest Value Transversal (HVT):

n positions (i, j) with one entry in each row & column such that $\sum \sigma_{ij}$ is maximized.

3. Find the smallest $c_i, d_j \geq 0$ satisfying

$$d_j - c_i \geq \sigma_{ij} \quad \text{for all } i, j$$

with

$$d_j - c_i = \sigma_{ij} \quad \text{on the HVT.}$$

Example: Simple Pendulum

$$0 = f_1 = x'' + x\lambda$$

$$0 = f_2 = y'' + y\lambda - g$$

$$0 = f_3 = x^2 + y^2 - L^2,$$

where $g > 0$ and $L > 0$ are constants, and λ is a Lagrange multiplier.

	x	y	λ	c
f_1	2	$-\infty$	0	0
f_2	$-\infty$	2	0	0
f_3	0	0	$-\infty$	2
d	2	2	0	

Two HVTs: $(1,1)$, $(2,3)$, $(3,2)$ and $(1,3)$, $(2,2)$, $(3,1)$

4. Form the System Jacobian

$$\mathbf{J} = \frac{\partial \left(f_1^{(c_1)}, \dots, f_n^{(c_n)} \right)}{\partial \left(x_1^{(d_1)}, \dots, x_n^{(d_n)} \right)}.$$

Example:

$$0 = f_1 = x'' + x\lambda$$

$$0 = f_2 = y'' + y\lambda - g$$

$$0 = f_3 = x^2 + y^2 - L^2,$$

$$\mathbf{J} = \frac{\partial \left(f_1^{(0)}, f_2^{(0)}, f_3^{(2)} \right)}{\partial \left(x^{(2)}, y^{(2)}, \lambda^{(0)} \right)} = \begin{bmatrix} \frac{\partial f_1}{\partial x''} & 0 & \frac{\partial f_1}{\partial \lambda} \\ 0 & \frac{\partial f_2}{\partial y''} & \frac{\partial f_2}{\partial \lambda} \\ \frac{\partial f_3}{\partial x} & \frac{\partial f_3}{\partial y} & 0 \end{bmatrix} = \begin{bmatrix} 1 & 0 & x \\ 0 & 1 & y \\ 2x & 2y & 0 \end{bmatrix}$$

Here, $\partial f_3^{(2)} / \partial x^{(2)} = \partial f_3^{(0)} / \partial x^{(0)}$ and $\partial f_3^{(2)} / \partial y^{(2)} = \partial f_3^{(0)} / \partial y^{(0)}$
(from Griewank's Lemma).

Informally, a **consistent initial point** is the values of an appropriate set of the x_j and derivatives of them, at a time t , that specify a unique solution.

5. If there is a consistent point of the DAE at which the System Jacobian J is nonsingular, then
 - DAE is solvable in a neighborhood of this point;
 - method shows how to reduce DAE to an ODE system;
 - or alternatively **solve by Taylor series**.

Computing Taylor Coefficients

We want to compute the $x_{j,r}$ (up to some order) in the series expansion

$$x_j(t) = \sum_{r \geq 0} x_{j,r} (t - t_0)^r, \quad j = 1, \dots, n.$$

Substituting into f_i ,

$$f_i(t) = \sum_{r \geq 0} f_{i,r} (t - t_0)^r, \quad i = 1, \dots, n.$$

We want $f_{i,r} = 0$.

Each $f_{i,r}$ is an expression in a finite number of the $x_{j,r}$.

We compute these expressions using automatic differentiation.

Solution process for Taylor coefficients:

For each stage $k = -\max_j d_j, k_d + 1, \dots, p$

$$\begin{array}{l} \text{solve } \{ f_{i,k+c_i} = 0 : \text{ all } i \text{ s.t. } k + c_i \geq 0 \} \\ \text{for } \{ x_{j,k+d_j} : \text{ all } j \text{ s.t. } k + d_j \geq 0 \} \end{array}$$

Example:

		stage k						
		$\mathbf{c, d}$	-2	-1	0	1	\dots	p
$f_1 = x'' + x\lambda$	$\mathbf{0}$				$f_{1,0}$	$f_{1,1}$	\dots	$f_{1,p+2}$
$f_2 = y'' + y\lambda - g$	$\mathbf{0}$				$f_{2,0}$	$f_{2,1}$	\dots	$f_{2,p+2}$
$f_3 = x^2 + y^2 - L^2$	$\mathbf{2}$		$f_{3,0}$	$f_{3,1}$	$f_{3,2}$	$f_{3,3}$	\dots	$f_{3,p}$
x	$\mathbf{2}$		x_0	x_1	x_2	x_3	\dots	x_{p+2}
y	$\mathbf{2}$		y_0	y_1	y_2	y_3	\dots	y_{p+2}
λ	$\mathbf{0}$				λ_0	λ_1	\dots	λ_p

Given an initial guess for $(x, x', y, y') = (x_0, x_1, y_0, y_1)$, solve:

stage	equations
-2	$0 = f_{3,0} = x_0^2 + y_0^2 - L^2$
-1	$0 = f_{3,1} = 2x_0x_1 + 2y_0y_1$
0	$0 = f_{1,0} = 2x_2 + x_0\lambda_0$ $0 = f_{2,0} = 2y_2 + y_0\lambda_0 - g$ $0 = f_{3,2} = 2x_0x_2 + 2y_0y_2 + x_1^2 + y_1^2$
1	$0 = f_{1,1} = 3!x_3 + x_0\lambda_1 + x_1\lambda_0$ $0 = f_{2,1} = 3!y_3 + y_0\lambda_1 + y_1\lambda_0$ $0 = f_{3,3} = 2x_0x_3 + 2y_0y_3 + 2x_1x_2 + 2y_1y_2$
\vdots	\vdots

At stage $k = 0$, the Jacobian

$$\mathbf{J}_0 = \frac{\partial(f_{1,0}, f_{2,0}, f_{3,2})}{\partial(x_2, y_2, \lambda_0)} = \begin{bmatrix} 2 & 0 & x_0 \\ 0 & 2 & y_0 \\ 2x_0 & 2y_0 & 0 \end{bmatrix},$$

is a diagonally scaled version $D_1 \mathbf{J} D_2$ of the System Jacobian \mathbf{J} .

Stages $k = 1, 2, \dots$: the systems are linear with a diagonally scaled \mathbf{J} .

Stages $k < 0$ comprise the projection on the constraint manifold.

On each stage, with an initial guess \tilde{x} , we solve

$$\min_x \|\tilde{x} - x\|_2 \quad \text{subject to} \quad \{ f_{i,k+c_i} = 0 : k + c_i \geq 0 \}.$$

The systems are generally nonlinear and undetermined.

\tilde{x} and x are vectors indexed over

$$\{ (j, k + d_j) : k + d_j \geq 0 \}.$$

First step User gives values for some or all of

$$x_j^{(k+d_j)} \quad \text{for which} \quad k \leq 0 \quad \text{and} \quad k + d_j \geq 0$$

Next steps good approximations are readily obtained from the Taylor series.

II. The Integration Process

- A. Input: DAE, initial values, integration interval, tolerance(s).
- B. Initialization
 1. generate the Σ matrix
 2. solve a linear assignment problem
 3. find a consistent initial point
- C. If a consistent initial point is computed, then, on each integration step:
 1. generate Taylor coefficients
 2. compute a Taylor series solution
 3. if an error tolerance is not satisfied, reduce the stepsize and goto C2
 4. project the Taylor series solution
 5. if an error tolerance for the projection is not satisfied, half the stepsize and do C2 and C4.

III. Software

DAETC — High-Index DAE Solver

Designed as a set of C++ classes

Solver 5,332 lines of C/C++ code

20 examples 3,049 lines of C/C++ code

DAETC builds on

- LAP [R. Jonker and A. Volgenant] for solving linear assignment problems (C);
- FADBAD/TADIFF [C. Bendsten and O. Stauning] for generating Taylor coefficients and Jacobians of Taylor coefficients (C++);
- IPOPT [A. Wächter] for finding a consistent initial point and projecting on each integration step (FORTRAN 77);
- LAPACK for solving linear systems (FORTRAN 77).

Two-pendula Example

An artificial problem to show we can solve high-index DAEs

$$0 = x'' + x\lambda$$

$$0 = y'' + y\lambda - g$$

$$0 = x^2 + y^2 - L^2$$

$$0 = u'' + u\kappa$$

$$0 = v'' + v\kappa - g$$

$$0 = u^2 + v^2 - (L + c\lambda)^2$$

g, L, c are constant

$x(t), y(t), u(t), v(t)$ are the state variables

$\lambda(t), \kappa(t)$ are Lagrange multipliers

Index is 5.

The C++ Definition

```
#define sqr(Y) ((Y)*(Y))

template <typename T>
void TwoPendula( T *f, const T *Y, const T & t )
{
    double g, L, c;
    g = 1.0;
    L = 1.0;
    c = 0.1;

    // x = Y[0], y = Y[1], lambda = Y[2]
    // u = Y[3], v = Y[4], kappa = Y[5]

    f[0] = diff(Y[0],2) + Y[0]*Y[2];
    f[1] = diff(Y[1],2) + Y[1]*Y[2] - g;
    f[2] =  sqr(Y[0])    + sqr(Y[1]) - sqr(L);

    f[3] = diff(Y[3],2) + Y[3]*Y[5];
    f[4] = diff(Y[4],2) + Y[4]*Y[5] - g;
    f[5] =  sqr(Y[3])    + sqr(Y[4]) - sqr(L+c*Y[2]);
}
```

- This is a template function.
It is used to generate the Σ matrix and the computational graph for computing Taylor coefficients.
- `diff(x,i)` returns the i th derivative of x .
- The mathematical description is translated “one-to-one”.

Output

A dash denotes $-\infty$

Signature Matrix

	0	1	2	3	4	5	c_j
0	2*	-	0	-	-	-	2
1	-	2	0*	-	-	-	2
2	0	0*	-	-	-	-	4
3	-	-	-	2	-	0*	0
4	-	-	-	-	2*	0	0
5	-	-	0	0*	0	-	2
d_j	4	4	2	2	2	0	

Structural Index 5

Degrees of Freedom 4

Solution Scheme

Stage	Functions	Variables
-4	-	x(0, 0)
	-	x(1, 0)
	f(2, 0)	-
	-	-
	-	-
	-	-
-3	-	x(0, 1)
	-	x(1, 1)
	f(2, 1)	-
	-	-
	-	-
	-	-
-2	f(0, 0)	x(0, 2)
	f(1, 0)	x(1, 2)
	f(2, 2)	x(2, 0)
	-	x(3, 0)
	-	x(4, 0)
	f(5, 0)	-

-1	f(0, 1)	x(0, 3)
	f(1, 1)	x(1, 3)
	f(2, 3)	x(2, 1)
	-	x(3, 1)
	-	x(4, 1)
	f(5, 1)	-

0	f(0, 2)	x(0, 4)
	f(1, 2)	x(1, 4)
	f(2, 4)	x(2, 2)
	f(3, 0)	x(3, 2)
	f(4, 0)	x(4, 2)
	f(5, 2)	x(5, 0)

Solution at $t = 100$

$$\begin{aligned}
 x(0,0) &= -4.576627e-01 \\
 x(0,1) &= 1.482003e+00 \\
 x(0,2) &= 1.678422e+00 \\
 x(0,3) &= -4.387699e+00 \\
 x(0,4) &= -1.604256e+01
 \end{aligned}$$

$$x(1,0) = 8.891259e-01$$

$$x(1,1) = 7.628362e-01$$

$$x(1,2) = -2.260760e+00$$

$$x(1,3) = -4.832381e+00$$

$$x(1,4) = 1.082985e+01$$

$$x(2,0) = 3.667378e+00$$

$$x(2,1) = 2.288509e+00$$

$$x(2,2) = -6.782281e+00$$

$$x(3,0) = -1.275677e+00$$

$$x(3,1) = 2.528904e-01$$

$$x(3,2) = 2.121749e+00$$

$$x(4,0) = 4.905315e-01$$

$$x(4,1) = 1.295300e+00$$

$$x(4,2) = 1.841314e-01$$

$$x(5,0) = 1.663234e+00$$

```
CPU time (sec).....14.09
CPU time/step.....0.02
Steps.....819
  accepted.....819 (100.0%)
  rejected.....0 (0.0%)
  failed projections.....0 (0.0%)
```

Solution Plots

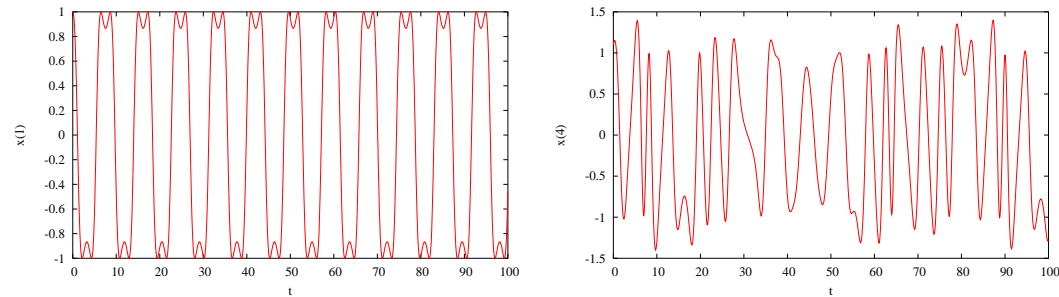


Figure 1: $x(1) = x$, $x(4) = u$ versus t

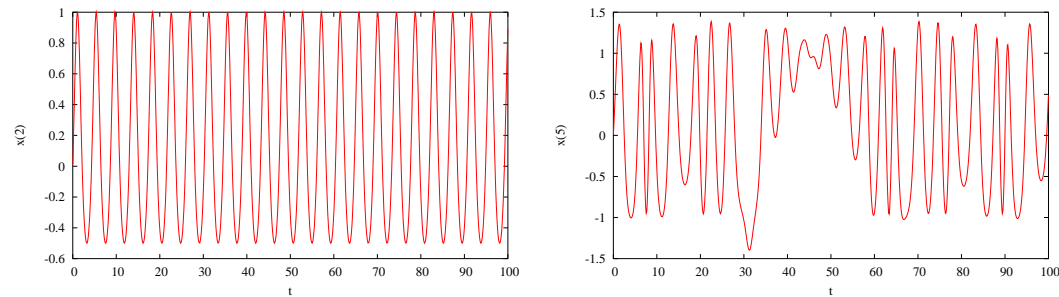


Figure 2: $x(2) = y$, $x(5) = v$ versus t

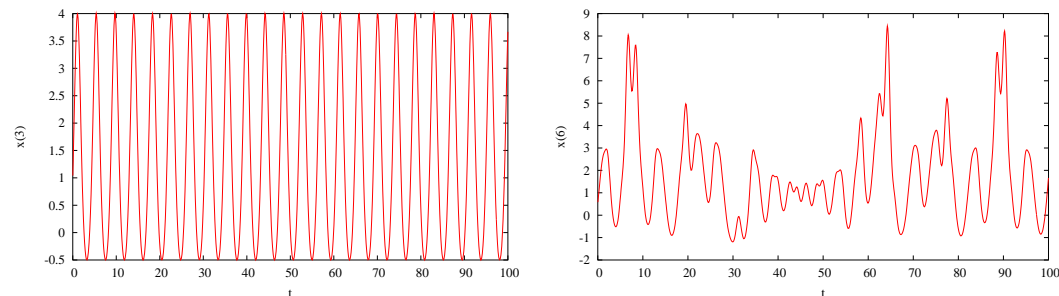


Figure 3: $x(3) = \lambda$, $x(6) = \kappa$ versus t