

# DAETS: a Differential-Algebraic Equation Code in C++ for High Index and High Accuracy

Ned Nedialkov<sup>1</sup> and John Pryce<sup>2</sup>

<sup>1</sup>Dept. of Computing and Software, McMaster U., Canada  
nedialk@mcmaster.ca

<sup>2</sup>Dept. of Information Systems, Cranfield U., UK  
j.d.pryce@ntlworld.com

SciCADE 2009  
May 2009, Beijing

# Contents

Overview of DAETS

Numerical method

Numerical results

Hybrid DAEs

Summary

## References

- ▶ N. S. Nedialkov and J. D. Pryce
  - ▶ Solving differential-algebraic equations by Taylor series (I): computing Taylor coefficients  
BIT, 45(3), pp. 561–591, 2005
  - ▶ (II): computing the System Jacobian  
BIT, 47(1), pp. 121–135, 2007
  - ▶ (III): the DAETS code  
JNAIAM 3 (2008), pp. 61–68
  - ▶ DAETS User Guide  
Tech. Report, Dept. Computing & Software, McMaster University, 2008–2009
- ▶ N. Nedialkov and N. Ramdani. [Towards integrating hybrid DAEs with a high-index DAE solver](#)  
Tech. Report, Dept. Computing & Software, McMaster University, 2008; also Tech. Report at INRIA

## DAETS solves DAEs by Taylor series expansion

- ▶ DAETS— **D**ifferential **A**lgebraic **E**quations by **T**aylor **S**eries
- ▶ Solves DAE initial value problems, for state variables  $x_j(t)$ ,  $j = 1, \dots, n$ , of the general form

$$f_i(t, \text{the } x_j \text{ and derivatives of them}) = 0, \quad i = 1, \dots, n$$

- ▶ Can be **fully implicit**
- ▶ **d/dt can appear anywhere** in the expressions for  $f_i$   
e.g. one of the equations could be

$$\frac{((x'_1 \sin t)')^2}{1 + (x'_2)^2} + t^2 \cos x_2 = 0$$

## DAETS solves high-index problems

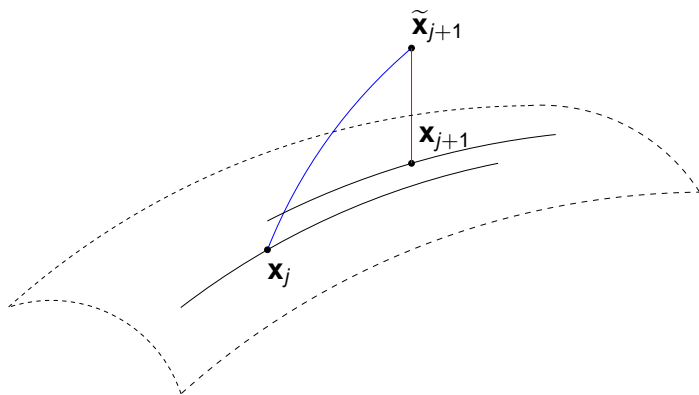
- ▶ Index  $\nu$  measures how “hard” DAE is to solve
- ▶ For traditional methods,  $\nu \geq 3$  considered hard
- ▶ DAETS based on **structural analysis** of DAE + **Taylor series**
- ▶ In principle unaffected by index
- ▶ Have solved artificial problems up to  $\nu = 47$   
(Any physical sense? ... is another matter)

## DAETS is a new kind of DAE solver

- ▶ Excellent at **high-index** DAEs
- ▶ Excellent for getting **high accuracy**
- ▶ Returns useful data about **structure** of problem
- ▶ Doesn't compete on speed at moderate accuracies
- ▶ ... or on handling very large problems

## Numerical method summary

- ▶ Start with code for the  $f_i$  that define the DAE
- ▶ Use automatic differentiation (AD), **FADBAD++** package, to evaluate suitable derivatives  $f_i^{(r)} = \frac{d^r f_i}{dt^r}$  at given  $t = t_j$
- ▶ For each step:
  - ▶ Equate these to zero “in batches” to get Taylor coefficients of (vector) solution  $\mathbf{x}(t)$  at current  $(t_j, \mathbf{x}_j)$
  - ▶ Sum Taylor series to get approximation  $\tilde{\mathbf{x}}_{j+1}$  at  $t_{j+1} = t_j + h_j$
  - ▶ Project this  $\tilde{\mathbf{x}}_{j+1}$  on DAE’s constraints to get a **consistent**  $\mathbf{x}_{j+1}$
- ▶ Repeat, to step along range in usual way



## Numerical method, cont.

- ▶ Before all this, do Structural Analysis:  
preprocess the DAE code to find the  $2n$  integer **offsets**,  
one for each variable, one for each equation
- ▶ These prescribe the “batches” in the overall process of  
solving for TCs
- ▶ They imply the Initial Values (IVs) data is **not a flat vector**  
unlike with most DAE solvers
- ▶ Following example illustrates

# The simple pendulum

Index-3 system with equations

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

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

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

$G =$  gravity

$L =$  length of pendulum

State variables  $x(t)$ ,  $y(t)$ ,  $\lambda(t)$

Item	$x$	$y$	$\lambda$	$f$	$g$	$h$
Offset	2	2	0	0	0	2

## User needs offsets to understand IVs

- ▶ Offsets tell what initial values (IVs) should be provided

- ▶ Offsets  $\begin{matrix} x & y & \lambda \\ 2 & 2 & 0 \end{matrix}$

mean that IVs comprise values for  $x, x'$ ;  $y, y'$

$x$	$x'$
$y$	$y'$

- ▶ Except when DAETS sees DAE is **non-linear in leading derivatives** (here  $x'', y'', \lambda$ )

It requires an extra set of derivatives

E.g. if first equation were

$$0 = (x'')^3 + x\lambda$$

then IVs must comprise  $x, x', x''$ ;  $y, y', y''$ ;  $\lambda$

- ▶ Reason: to assure **local uniqueness** of solution

$x$	$x'$	$x''$
$y$	$y'$	$y''$
$\lambda$		

## User needs offsets to understand constraints

Offsets tell what constraints the provided IVs must meet for consistency

$$\text{Offsets} \quad \begin{array}{ccc} f & g & h \\ 0 & 0 & 2 \end{array}$$

mean they must satisfy

in the linear case

$$h, h' = 0:$$

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

$$0 = h' = 2xx' + 2yy'$$

in the non-linear case

$$f; g; h, h', h'' = 0,$$

so in our example, add these:

$$0 = f = (x'')^3 + x\lambda$$

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

$$0 = h'' = 2(xx'' + (x')^2 + yy'' + (y')^2)$$

## Finding consistent point

- ▶ Solution  $\mathbf{x}(t)$  must satisfy algebraic constraints **for all  $t$**  to be consistent
- ▶ Constraint can be **obvious**, as (for Pendulum)  $h = 0$ , or **hidden**, as  $h' = 0$
- ▶ Finding **initial consistent point** can be hardest part of solving DAE
- ▶ Not built in to most solvers
- ▶ Fits naturally into DAETS workflow  
We formulate it as a nonlinear minimization problem and give it to **IPOPT** package

# Numerical results

- ▶ Accuracy
- ▶ Efficiency
- ▶ DAETS on a High-index problem
- ▶ DAETS on a Continuation problem

## Accuracy vs. tolerance

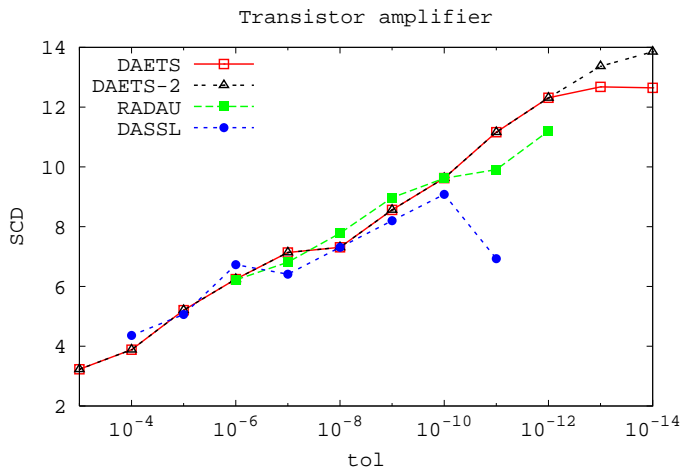
- ▶ We plot Significant Correct Digits:

$$\text{SCD} = -\log_{10} \|\text{componentwise rel. error at end of integration}\|$$

as a function of tolerance

- ▶ Problem is **Transistor Amplifier**, index-1 DAE of size  $n = 8$ , from *Test Set for Initial Value Problem Solvers*, Bari University, Italy
- ▶ “DAETS”, “RADAU”, “DASSL” curves compare with reference solution in Test Set documentation
- ▶ “DAETS-2” curve uses reference solution computed by DAETS with  $\text{tol} = 10^{-16}$

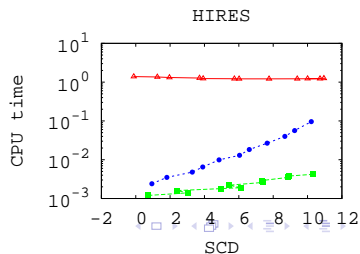
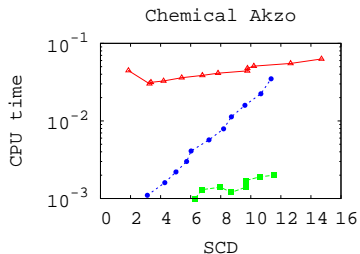
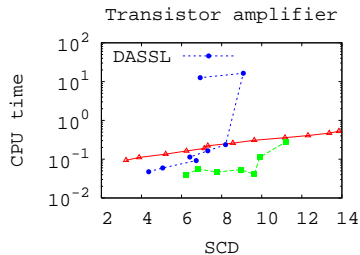
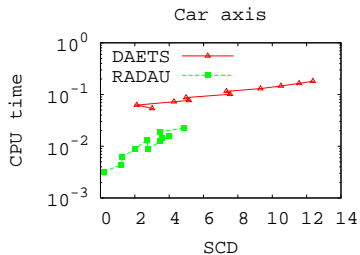
# Plots of accuracy vs. tolerance



## Efficiency experiments

- ▶ Plot CPU time vs. SCD
- ▶ Problems are (from the ODE/DAE test set)
  - ▶ Car axis index-3 DAE,  $n = 10$
  - ▶ Transistor amplifier index-1 DAE,  $n = 8$
  - ▶ Chemical Akzo Nobel index-1 DAE,  $n = 6$
  - ▶ HIRES ODE,  $n = 8$
- ▶ Produce **work-precision diagrams** for DAETS, DASSL, and RADAU (as described in the test set)
- ▶ Compare with DASSL and RADAU solvers

# Work-precision diagrams



## “Multi-pendula” — a class of high-index problems

- ▶ System is a “chain” of  $P$  simple pendula with coupling
- ▶ Pendulum 1 is as normal
- ▶ Length of pendulum  $p$  depends on  $\lambda_{p-1}$ , for  $p = 2, \dots, P$
- ▶ For  $P = 2$

$$0 = x_1'' + \lambda_1 x_1$$

$$0 = x_2'' + \lambda_2 x_2$$

$$0 = y_1'' + \lambda_1 y_1 - G \quad \text{and} \quad 0 = y_2'' + \lambda_2 y_2 - G$$

$$0 = x_1^2 + y_1^2 - L^2$$

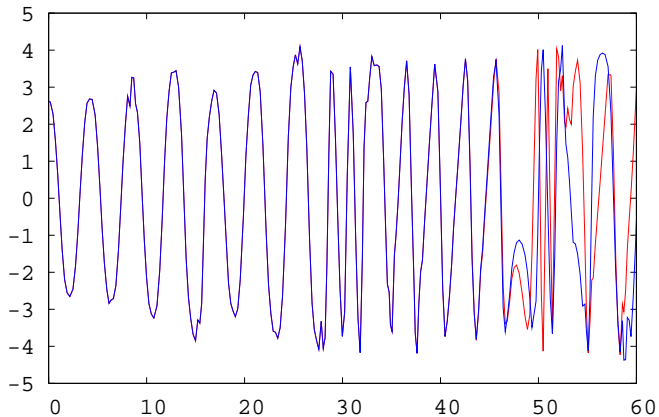
$$0 = x_2^2 + y_2^2 - (L + c\lambda_1)^2$$

where  $c$  is a constant

- ▶ Chain of length  $P$  has **size**  $n = 3P$  and **index**  $2P + 1$
- ▶ DAETS has solved systems for  $P$  up to 23 giving **index 47**

## $P = 7$ , index 15, $x_7(t)$ plot

Seven pendula,  $\text{tol} = 10^{-9}$ ,  $x_7$  with two slightly differing ICs



## Continuation problems

- ▶ No need for derivatives to be present  
Can solve  $n$  purely algebraic equations

$$\mathbf{f}(\lambda, \mathbf{x}) = \mathbf{0}$$

to find  $\mathbf{x} = (x_1, \dots, x_n)$  as a function of  $\lambda$

- ▶ To handle **turning points**, use **arc-length continuation**
- ▶ Define Euclidean arc-length  $s$  by

$$(d\lambda/ds)^2 + (dx_1/ds)^2 + \dots + (dx_n/ds)^2 = 1$$

- ▶ Find  $(\lambda, \mathbf{x})$  as function of  $s$
- ▶ Gives an index-1 DAE of size  $n + 1$

## Continuation example

- ▶ Difficulty is **path tracking**
- ▶ Illustrate with problem from Layne Watson (1979)  
Solve  $\mathbf{g}(\mathbf{x}) = \mathbf{x}$  for  $\mathbf{g} = (g_1, \dots, g_n)$ , where

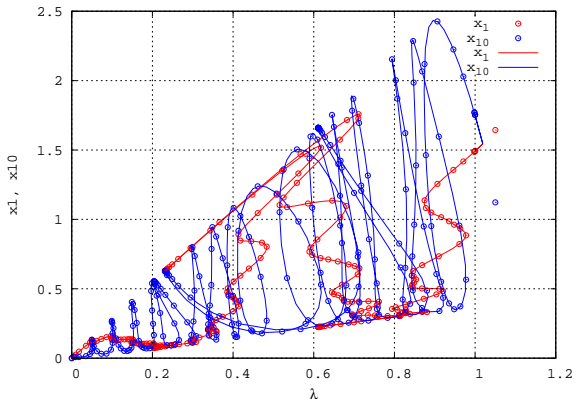
$$g_i(\mathbf{x}) = g_i(x_1, \dots, x_n) = \exp\left(\cos\left(i \sum_{k=1}^n x_k\right)\right), \quad i = 1, \dots, n$$

- ▶ Many solutions. Hard for even  $n$  around 10
- ▶ Formulate as

$$\mathbf{0} = \mathbf{f}(\lambda, \mathbf{x}) = \mathbf{x} - \lambda \mathbf{g}(\mathbf{x})$$

and “continue” from  $\lambda = 0$  to  $\lambda = 1$  using arc-length

## Two components of Layne Watson curve for $n = 10$



- ▶ Lots of turning points!
- ▶ Tracking failure is serious problem if step size  $h$  unlimited  
Restricting  $h \leq 0.3$  cured it

## Present work: hybrid DAEs

- ▶ Hybrid DAEs comprise continuous and discrete behavior
- ▶ In DAETS, continuous behavior is described by a DAE of the form

$$f_i(t, \text{the } x_j \text{ and derivatives of them}) = 0, \quad i = 1, \dots, n$$

- ▶ Discrete change when a scalar-valued event function

$$g(t, x_j \text{ and derivatives of them})$$

becomes zero

- ▶ Equations, initial conditions, or both may change

## Example: bouncing pendulum

- ▶ Pendulum of length  $L$  attached at  $(0, 0)$
- ▶ Peg at point  $(0, -a)$ ,  $0 < a < L$
- ▶ Vertical wall through  $(0, 0)$  and above  $(0, -a)$

$x < 0$	$x \geq 0$
$0 = x'' + \lambda x$	$0 = x'' + \lambda x$
$0 = y'' + \lambda y - g$	$0 = y'' + \lambda(y - a) - g$
$0 = x^2 + y^2 - L^2$	$0 = x^2 + (y - a)^2 - l^2$

- ▶ When pendulum hits the wall,  $x'$  becomes  $-kx'$   
 $k$  is coefficient of restitution,  $0 < k < 1$
- ▶ Event function is  $g(x) = x$

## Some issues

- ▶ Structure of DAE —index, linearity, size— may change at an event  
DAE structure is discovered at run-time by DAETS
- ▶ Computing consistent point
  - ▶ given initial guess  $\tilde{\mathbf{x}}$ , fix DAE
  - ▶ compute consistent point  $\mathbf{x}$  with this DAE
  - ▶  $\mathbf{x}$  may imply different DAE

▶ Consider

if  $x > 0$

$$F_1(t, x, \dots) = 0$$

else

$$F_2(t, x, \dots) = 0$$

- ▶ If initial guess for  $x$  is  $> 0$ , try to find a consistent point with  $F_1$
- ▶ If the computed  $x < 0$  we need to integrate with  $F_2$ , but the computed  $x$  may not be consistent with  $F_2$
- ▶ Currently, simple algorithm in DAETS for finding a consistent point

We plan to use continuation

## Summary

- ▶ We now have a robust DAE code based on Structural Analysis theory that J Pryce began, with George Corliss, in 1996
- ▶ Excellent for high index and high accuracy
- ▶ Gives useful information about DAE structure
- ▶ Available for Linux (x86, x86-64), Mac OS X (x86, PowerPC), Windows/Cygwin (x86)
  - ▶ Demo version at <http://www.cas.mcmaster.ca/~nedialk/daets>
  - ▶ Full version at [Flintbox](#)
- ▶ Version 1.0, June 2008
- ▶ Version 1.1, June 2009
  - ▶ tested extensively with CppUnit
  - ▶ code coverage analysis

## Example: simple pendulum—code for function

```

#include "DAEsolver.h"

template <typename T>
void fcn(T t, const T *x, T *f, void *param)
{
    double G = 9.8, L = 10;

    f[0] = Diff(x[0],2) + x[0]*x[2];           //  $0 = x_1'' + \lambda_1 x_1$ 
    f[1] = Diff(x[1],2) + x[1]*x[2] - G;     //  $0 = y_1'' + \lambda_1 y_1 - G$ 
    f[2] = sqr(x[0]) + sqr(x[1]) - sqr(L);    //  $0 = x_1^2 + y_1^2 - L^2$ 
}

```

## Example: simple pendulum—main program

```
int main() {
    const int n = 3; // size of the problem
    DAEsolver Solver(n, DAE_FCN(fcn)); // create a solver
                                        // and analyze DAE
    Solver.printDAEinfo(); // print info about the DAE
    Solver.printDAEpointStructure(); // .. and more info

    DAEsolution x(Solver); // create a DAE solution object
    x.setT(0.0); // initial value of t
    x.setX(0, 0, -1.0).setX(0, 1, 0.0); // .. and of x and x'
    x.setX(1, 0, 0.0).setX(1, 1, 1.0); // .. and of y and y'

    double tend = 100.0;
    DAEexitflag flag;
    Solver.integrate(x, tend, flag); // integrate until tend
    if (flag!=success)
        printDAEexitflag(flag); // check the exit flag
    x.printSolution(); // print solution
    x.printStats(); // print integration statistics
    return 0;
}
```

# Simple pendulum — output

```
DAE
```

```
---
```

```
quasi-linear
```

```
size.....3
```

```
index.....3
```

```
POINT STRUCTURE
```

```
-----
```

```
Initialize:
```

```
variable    derivatives
```

```
  0          0    1
```

```
  1          0    1
```

```
-
```

# Simple pendulum — output cont.

SOLUTION

-----

t = 1.000000e+02

	x	x'
0	8.037130e+00	6.453216e+00
1	5.950171e+00	-8.716614e+00
2		

STATISTICS

-----

TIME.....0.06

STEPS.....388

    accepted.....388

    rejected.....0

    %.....0.00

STEPSIZES

    smallest.....0.020

    largest .....0.352

ORDER.....15

TOLERANCE

    relative...1.0e-12

    absolute...1.0e-12

## Bouncing pendulum: DAE and event functions

```

// DAE
template <typename T>
void fcn(T t, const T *x, T *f, void *p) {
    PendPegParams *params = (PendPegParams*)p;
    double L = params->L, l = params->l, a = L-l;
    double g = 9.81;

    f[0] = Diff(x[0], 2) + x[0]*x[2];           //  $0 = x'' + x\lambda$ 
    if (params->mode == 1) { //  $x < 0$ 
        f[1] = Diff(x[1], 2) + x[1]*x[2] - g;   //  $0 = y'' + y\lambda - g$ 
        f[2] = sqr(x[0]) + sqr(x[1]) - sqr(L);   //  $0 = x^2 + y^2 - L^2$ 
    }
    else { //  $x \geq 0$ 
        f[1] = Diff(x[1], 2) + (x[1]-a)*x[2] - g; //  $0 = y'' + (y-a)\lambda - g$ 
        f[2] = sqr(x[0]) + sqr(x[1]-a) - sqr(l); //  $0 = x^2 + (y-a)^2 - l^2$ 
    }
}
// Event function
void event_fcn(TCdiff t, const TCdiff *x, TCdiff *g, void *param) {
    g[0] = x[0];
}

```

## Bouncing pendulum: parameters class

```
#include "eventvalues.h"

// Parameter class for passing parameters to the problem
class PendPegParams : public EventValues {
public:
  PendPegParams() : EventValues(1) {};
  int mode; // mode = 1, if  $x < 0$ , 2 otherwise
  double L, l; // pendulum lengths
} ;
```

## Bouncing pendulum: main program

```
#include "HybridDAEsolver.h"
using namespace daets;
#include "pendulumpeg.h"
// Forward declaration of output function
void outfcn(const DAEsolution &x, void *params, void *output_params);

int main() {
    int n = 3, m = 1;    // problem size, number of event functions
    double t0 = 0, tend = 60;
    double k = 0.7;     // coefficient of restitution

    PendPegParams params;
    params.L = 10, params.l = 4; // lengths L, l
    params.mode = 1;           // we start in mode 1

    // Create solver and solution objects
    HybridDAEsolver Solver(n, DAE_FCN(fcn), &params, m, event_fcn);
    DAEsolution x(Solver);
```

```
// Set initial t,x,y,x',y'
x.setT(t0).setX(0,0, -8).setX(1,0, 6)
    .setX(0,1, -30).setX(1,1, -40);

// Open file for writing solution components
// and set an output function
FILE *fh;
fh = fopen("pendpegsol.out", "w");
x.setOutputFunction(outfcn, &params, fh);

SolverExitFlag state = success;
// Low order to keep the stepsize small, suitable for plotting
Solver.setOrder(5);
```

```
do {
  int prevmode = params.mode;
  // Find consistent point
  Solver.compConsistentPoint(x, DAE_FCN(fcn), &params, state);
  // Integrate up to an event or tend
  Solver.integrate(x, tend, state);
  if (state != success) break;

  double xp = x.getX(0,1), y = x.getX(1,0);
  // If the pendulum hits the wall, reset x and x'
  if (fabs(params.L+y) < 1e-10 ||
      fabs(params.L-2*params.l-y) < 1e-10) {
    x.setFirstEntry();
    x.setX(0,0, 0).setX(0,1, -k*xp);
  }
  else // change modes
    params.mode = (prevmode==1)? 2:1;

} while (x.getT() != tend);
if (state!=success) printSolverExitFlag(state);
fclose(fh);
return 0;
}
```