

Interval Tools for ODEs and DAEs

Ned Nedialkov

Department of Computing and Software
McMaster University, Hamilton, Ontario
Canada

SWIM 2008
Montpellier, France
19 June 2008

Outline

Interval methods for IVP ODEs

The initial value problem

Software

Applications

Theory

The VNODE-LP solver

Motivation

Overview

Performance

On solving DAEs

Pryce's structural analysis

Work to date

Conclusion

The IVP Problem

We consider the IVP

$$y'(t) = f(y), \quad y(t_0) = y_0, \quad y \in \mathbb{R}^n, \quad t \in \mathbb{R}$$

The initial condition can be in an interval vector, $y_0 \in \mathbf{y}_0$

We denote the solution by $y(t; t_0, y_0)$

Denote

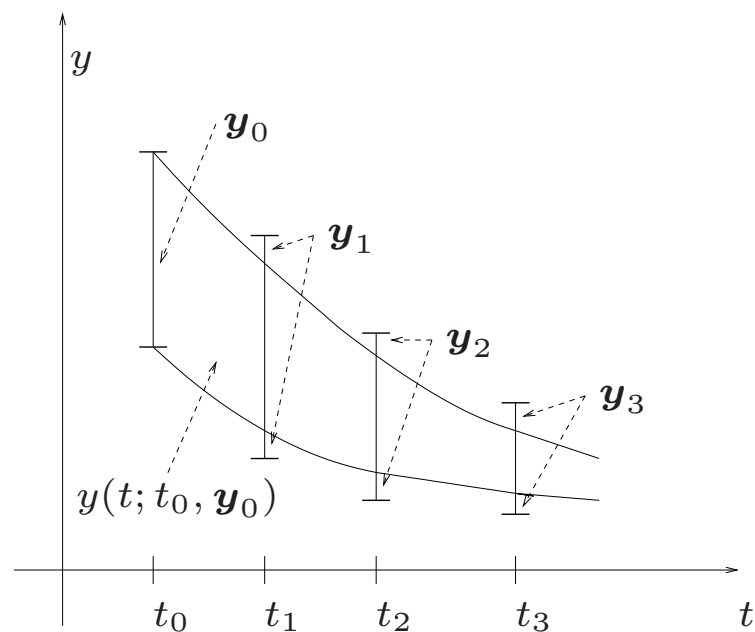
$$y(t; t_0, \mathbf{y}_0) = \{ y(t; t_0, y_0) \mid y_0 \in \mathbf{y}_0 \}$$

Compute \mathbf{y}_j such that

$$y(t_j; t_0, \mathbf{y}_0) \subseteq \mathbf{y}_j$$

at points

$$t_0 < t_1 < t_2 < \cdots < t_N = t_{\text{end}}$$



Software

package	year	author(s)	language	avail.
AWA	1988	R. Lohner	Pascal-XSC	✓
ADIODES	1997	O. Stauning	C++	✓
COSY	1997	M. Berz, K. Makino	Fortran	✓
			C++ interface	✓
VNODE	2001	N. Nedialkov	C++	✓
VODESIA	2003	S. Dietrich	Fortran-XSC	
VSPODE	2005	Y. Lin, M. Stadtherr	C++	
ValEncIA-IVP	2005	V. Rauh, E. Auer	C++	✓
VNODE-LP	2006	N. Nedialkov	C++	✓

The automatic differentiation (AD) packages TADIFF and FADBAD, and now [FADBAD++](#) (O. Stauning, C. Bendtsen), are instrumental in VNODE, VSPODE, ValEncIA-IVP, and VNODE-LP

Applications

Long-term stability of particle accelerators (1998–) M. Berz, K. Makino

Rigorous shadowing (2001) W. Hayes

Computing eigenvalue bounds (2003) B. M. Brown, M. Langer, M. Marletta, C. Tretter, M. Wagenhofer

Multibody simulations (2004) E. Auer, A. Kecskeméthy, M. Tändl, H. Traczinski

Reliable surface intersection (2004) H. Mukundan, K. H. Ko, T. Maekawa, T. Sakkalis, N. M. Patrikalakis

Parameter and state estimation (2004) M. Kieffer, E. Walter;
(2005) N. Ramdani, N. Meslem, T. Raïssi, Y. Candau

Robust evaluation of differential geometry properties (2005) Chih-kuo Lee

Chemical engineering (2005) Lin, Stadtherr

Rigorous parameter reconstruction for differential equations with noisy data
(2007) Johnson, Tucker

...

Theory

One step of a “traditional” method

Suppose that we have computed \mathbf{y}_j at t_j such that

$$\mathbf{y}(t_j; t_0, \mathbf{y}_0) \subseteq \mathbf{y}_j$$

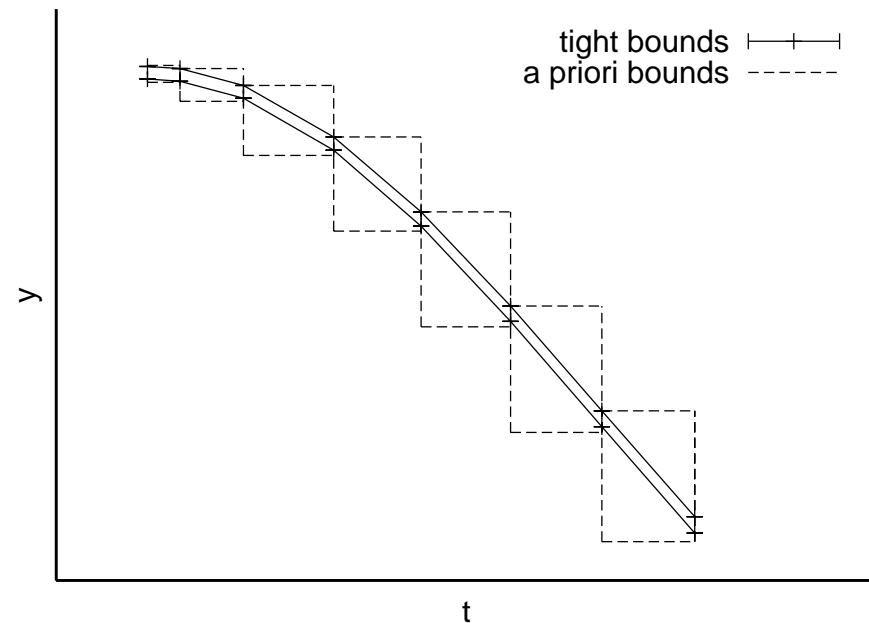
ALGORITHM I validates existence and uniqueness and computes an a priori enclosure $\tilde{\mathbf{y}}_j$ such that

$$\mathbf{y}(t; t_j, \mathbf{y}_j) \subseteq \tilde{\mathbf{y}}_j$$

for all $t \in [t_j, t_{j+1}]$

ALGORITHM II computes a tighter enclosure

$$\mathbf{y}_{j+1} \subseteq \tilde{\mathbf{y}}_j$$



Taylor coefficients

Denote

$$f^{[0]}(y) = y$$

$$f^{[i]}(y) = \frac{1}{i} \left(\frac{\partial f^{[i-1]}}{\partial y} f \right) (y), \quad \text{for } i \geq 1$$

Given the IVP

$$y'(t) = f(y), \quad y(t_j) = y_j,$$

the i th Taylor coefficient (TC) of $y(t)$ at t_j satisfies

$$\frac{y^{(i)}(t_j)}{i!} = f^{[i]}(y_j)$$

We require at most $O(k^2)$ work to compute $f^{[1]}(y_j), f^{[2]}(y_j), \dots, f^{[k]}(y_j)$

Given stepsize h , one can generate scaled TCs, i.e. $h^i f^{[i]}(y_j)$

Computing a priori bounds

High-Order Enclosure (HOE) method (NN, Jackson & Pryce)

Main result: If $y_j \in \text{int}(\tilde{\mathbf{y}}_j)$ and

$$y_j + \sum_{i=1}^{k-1} (t - t_j)^i f^{[i]}(y_j) + (t - t_j)^k f^{[k]}(\tilde{\mathbf{y}}_j) \subseteq \tilde{\mathbf{y}}_j$$

for all $t \in [t_j, t_{j+1}]$ and all $y_j \in \mathbf{y}_j$, then

$$y(t; t_j, y_j) \in y_j + \sum_{i=1}^{k-1} (t - t_j)^i f^{[i]}(y_j) + (t - t_j)^k f^{[k]}(\tilde{\mathbf{y}}_j)$$

for all $t \in [t_j, t_{j+1}]$ and all $y_j \in \mathbf{y}_j$

When $k = 1$, we obtain the method in AWA, but it restricts the stepsizes similarly to Euler's method

Computing tight bounds

Interval Taylor Series (ITS) Method

Using $\tilde{\mathbf{y}}_j$, compute a tighter enclosure \mathbf{y}_{j+1} :

$$y(t_{j+1}; t_0, \mathbf{y}_0) \subseteq \mathbf{y}_{j+1}$$

Basic approach: Taylor series + remainder term

We can compute

$$\mathbf{y}_{j+1} = \sum_{i=0}^{k-1} h_j^i f^{[i]}(\mathbf{y}_j) + h_j^k f^{[k]}(\tilde{\mathbf{y}}_j),$$

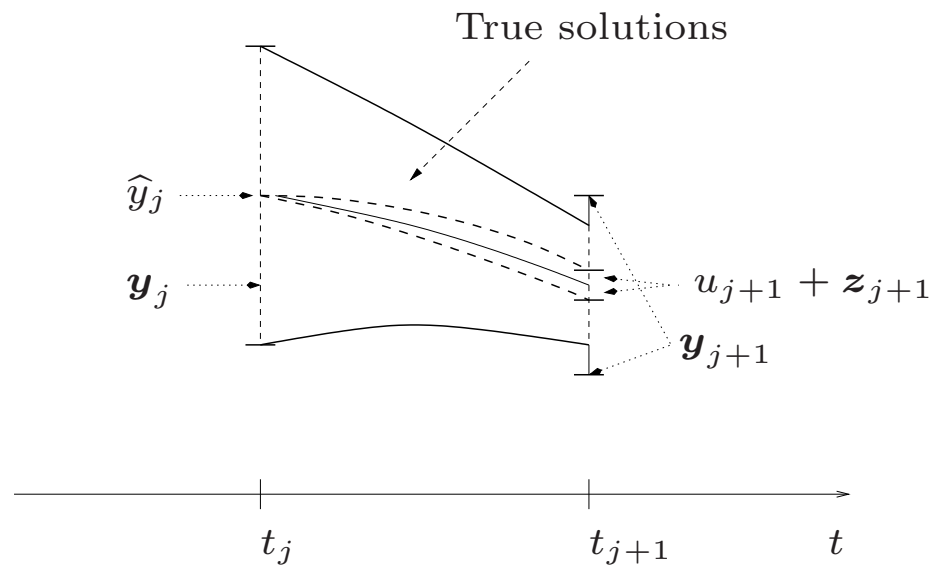
but the width is

$$w(\mathbf{y}_{j+1}) \geq w(\mathbf{y}_j), \quad \text{and usually } w(\mathbf{y}_{j+1}) > w(\mathbf{y}_j),$$

even if the solutions are contracting (“naive” method)

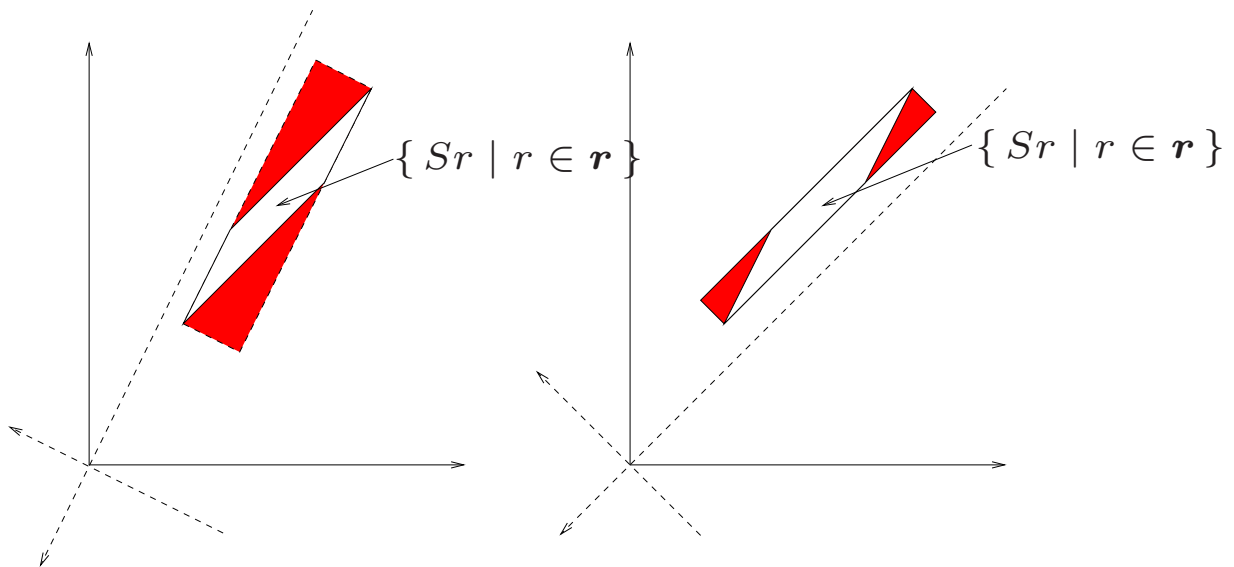
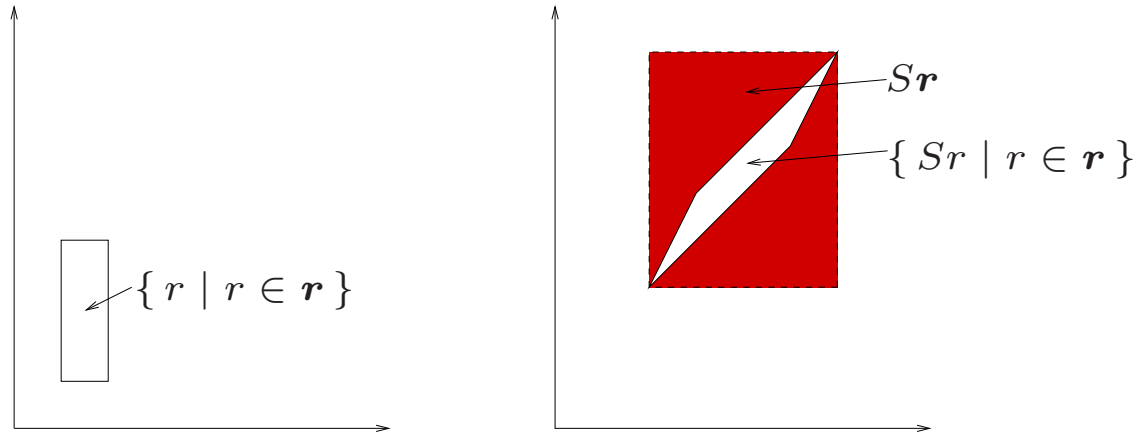
Use the **mean-value evaluation**: for any $y_j, \hat{y}_j \in \mathbf{y}_j$,

$$\begin{aligned} & \sum_{i=0}^{k-1} h_j^i f^{[i]}(y_j) + h_j^k f^{[k]}(\tilde{\mathbf{y}}_j) \\ & \subseteq \sum_{i=0}^{k-1} h_j^i f^{[i]}(\hat{y}_j) + h_j^k f^{[k]}(\tilde{\mathbf{y}}_j) + \left\{ \sum_{i=0}^{k-1} h_j^i \frac{\partial f^{[i]}}{\partial y}(\mathbf{y}_j) \right\} (\mathbf{y}_j - \hat{y}_j) \\ & =: \mathbf{y}_{j+1} = u_{j+1} + \mathbf{z}_{j+1} + \mathbf{S}_j(\mathbf{y}_j - \hat{y}_j) \end{aligned}$$



Can follow contracting solutions, but wrapping effect

Wrapping effect



Reducing the wrapping effect

On each step, represent the enclosure in the form

$$\mathbf{y}_j \in \{ \hat{\mathbf{y}}_j + A_j \mathbf{r}_j \mid \mathbf{r}_j \in \mathbf{r}_j \}, \quad A_j \in \mathbb{R}^{n \times n} \text{ nonsingular}$$

Instead of computing

$$\mathbf{y}_{j+1} = u_{j+1} + \mathbf{z}_{j+1} + \mathbf{S}_j(\mathbf{y}_j - \hat{\mathbf{y}}_j),$$

compute

$$\begin{aligned} \mathbf{y}_{j+1} &= u_{j+1} + \mathbf{z}_{j+1} + (\mathbf{S}_j A_j) \mathbf{r}_j \\ \mathbf{r}_{j+1} &= \{ A_{j+1}^{-1} (\mathbf{S}_j A_j) \} \mathbf{r}_j + A_{j+1}^{-1} \{ \mathbf{z}_{j+1} - m(\mathbf{z}_{j+1}) \}, \end{aligned}$$

where $\mathbf{r}_0 = \mathbf{y}_0 - \hat{\mathbf{y}}_0$, $A_0 = I$, $m(\cdot)$ is midpoint

How to select A_{j+1} ?

The Parallelepiped Method

$$A_{j+1} = m(\mathbf{S}_j A_j)$$

The A_j usually become ill conditioned

Lohner's QR Method

$$A_{j+1} = Q_{j+1} \quad \text{from the QR factorization } Q_{j+1} R_{j+1} = m(\mathbf{S}_j A_j)$$

We enclose in a moving orthogonal coordinate system

We can always “match” the longest edge of the enclosed set

The QR method provides **better stability** than the parallelepiped method
(NN, K. Jackson)

Taylor models

Major source of overestimation in traditional methods is the dependency problem

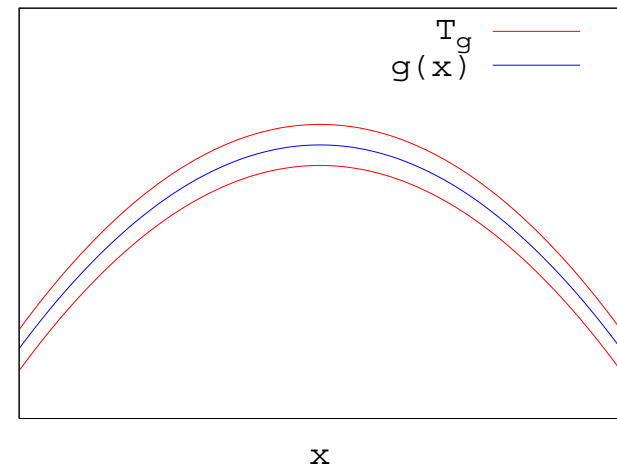
Taylor models (COSY, VSPODE) help to reduce it

Let \mathcal{F} be the set of continuous functions on $\mathbf{x} \in \mathbb{IR}^n$ to \mathbb{R} ,
let $p : \mathbb{R}^n \rightarrow \mathbb{R}$ be a polynomial of order m , and
let \mathbf{r} be an interval

A Taylor model is

$$\{ f \in \mathcal{F} \mid f(x) \in p(x) + \mathbf{r} \text{ for all } x \in \mathbf{x} \}$$

Arithmetic operations and elementary functions can be implemented on Taylor models



Taylor models in VSPODE

The IVP problem is

$$y' = f(y(t), \theta), \quad y(t_0) = y_0 \in \mathbf{y}_0, \quad \theta \in \boldsymbol{\theta} \quad (\text{parameter})$$

Set

$$T_{y_0} = (m(\mathbf{y}_0) + (y_0 - m(\mathbf{y}_0)), [0, 0]), \quad T_\theta = (m(\boldsymbol{\theta}) + (\theta - m(\boldsymbol{\theta})), [0, 0])$$

Assume at t_j

$$y(t_j; t_0, y_0, \theta) \in T_{y_j} = p_j(y_0, \theta) + \mathbf{v}_j \quad (p_j \text{ is a polynomial})$$

At t_{j+1} ,

$$\begin{aligned} y(t_{j+1}; t_j, y_j, \theta) &\in \sum_{i=0}^{k-1} h_j^i f^{[i]}(\mathbf{y}_j, \boldsymbol{\theta}) + h_j^k f^{[k]}(\tilde{\mathbf{y}}_j, \boldsymbol{\theta}) \\ &\subseteq \sum_{i=0}^{k-1} h_j^i f^{[i]}(p_j + \mathbf{v}_j, T_\theta) + \mathbf{z}_{j+1} \end{aligned}$$

Enclosures grow as in the “naive” TS method, but likely much slower

Apply the mean-value theorem to the $f^{[i]}$:

$$y(t_{j+1}; t_j, y_j, \theta) \in \underbrace{\sum_{i=0}^{k-1} h_j^i f^{[i]}(p_j, T_\theta) + z_{j+1}}_{p_{j+1}(y_0, \theta) + \mathbf{u}_{j+1}} + \underbrace{\left(\sum_{i=0}^{k-1} h_j^i \frac{\partial f^{[i]}}{\partial y}(y_j, \theta) \right)}_{\mathbf{S}_j} \mathbf{v}_j$$

To reduce wrapping effect from $\mathbf{S}_j \mathbf{v}_j$, use the representation

$$T_{y_j} = p_j(y_0, \theta) + B_j w, \quad w \in \mathbf{w}_j, \quad B_j \in \mathbb{R}^{n \times n} \text{ is nonsingular}$$

Recall: in Lohner’s method the solution is in

$$\{ \hat{y}_j + A_j r \mid r \in \mathbf{r}_j \}$$

For the next step, B_{j+1} and \mathbf{w}_{j+1} are computed like in Lohner’s method

The VNODE-LP Solver

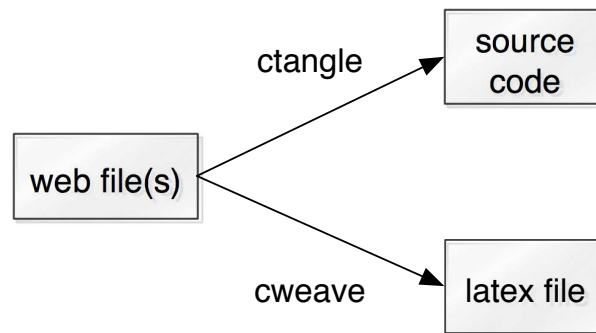
Motivation

In general, interval methods produce **rigorous results**

If we miss including a **single roundoff error**, they **may not be rigorous**

Goal: produce an interval ODE solver such that it can be verified for correctness by a human expert

VNODE-LP is produced entirely using **Literate Programming** (D. Knuth) and CWEB (D. Knuth, S. Levy)



Theory and code can be checked for correctness at the same time

Overview

VNODE-LP computes bounds on

$$y' = f(t, y), \quad y(t_0) \in \mathbf{y}_0, \quad t \in [t_0, t_{\text{end}}]$$

(or $t \in [t_{\text{end}}, t_0]$)

VNODE-LP implements:

- Algorithm I: HOE method
- Algorithm II: Hermite-Obreschkoff method (NN)
- Variable stepsize control
- Constant order: typical values can be between 20 and 30
- Improved wrapping effect control compared to VNODE

In general, applicable with point initial conditions, or interval initial conditions with a sufficiently small width

Packages and platforms

VNODE-LP builds on

PROFIL/BIAS *or* **FILIB++** (interval arithmetic)

Specified at compile time

FADBAD++ (automatic differentiation)

LAPACK and **BLAS** (linear algebra)

Installs with GCC:

IA	OS	Arch
FILIB++	Linux	x86
	Solaris	Sparc
PROFIL	Linux	x86
	Solaris	Sparc
	Mac OSX	PowerPC
	Windows with Cygwin	x86

Performance

Experiments on 3 GHz dual core Pentium, 2GB RAM, 4 MB L2 Cache;
Fedora Linux, gcc version 4.1.1 with -O2; with PROFIL/BIAS

Work versus order

Lorenz system

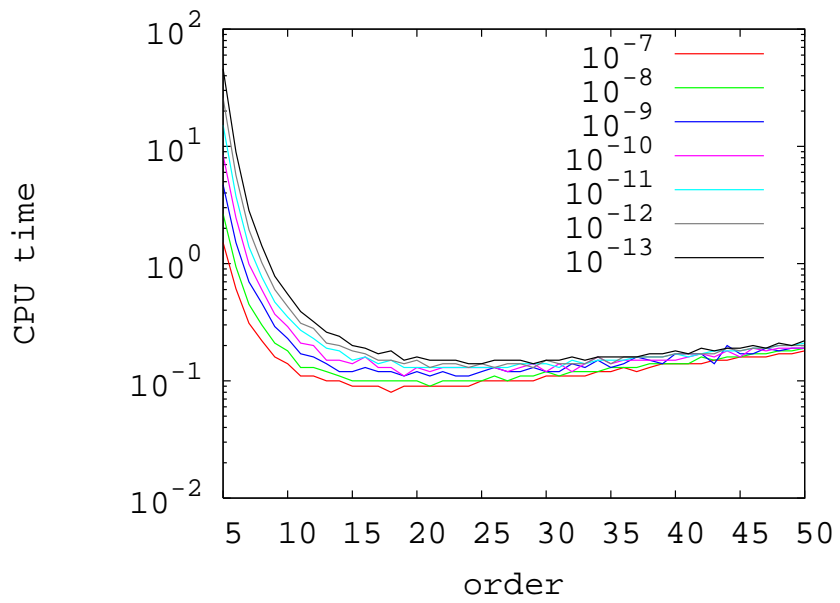
$$y_1' = 10(y_2 - y_1)$$

$$y_2' = y_1(28 - y_3) - y_2$$

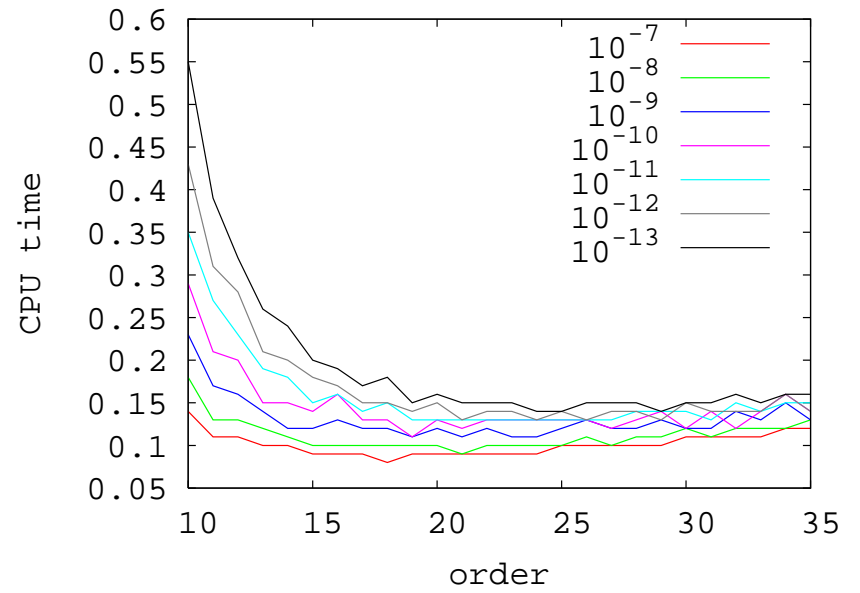
$$y_3' = y_1 y_2 - 8/3 y_3$$

$$y(0) = (15, 15, 36)^T, \quad t \in [0, 20]$$

integrated with $\text{atol} = \text{rtol} = 10^{-7}, \dots, 10^{-13}$



(a) $\log_{10}(\text{CPU time})$ vs. order



(b) CPU time vs. order

Work versus problem size

DETEST problem C3

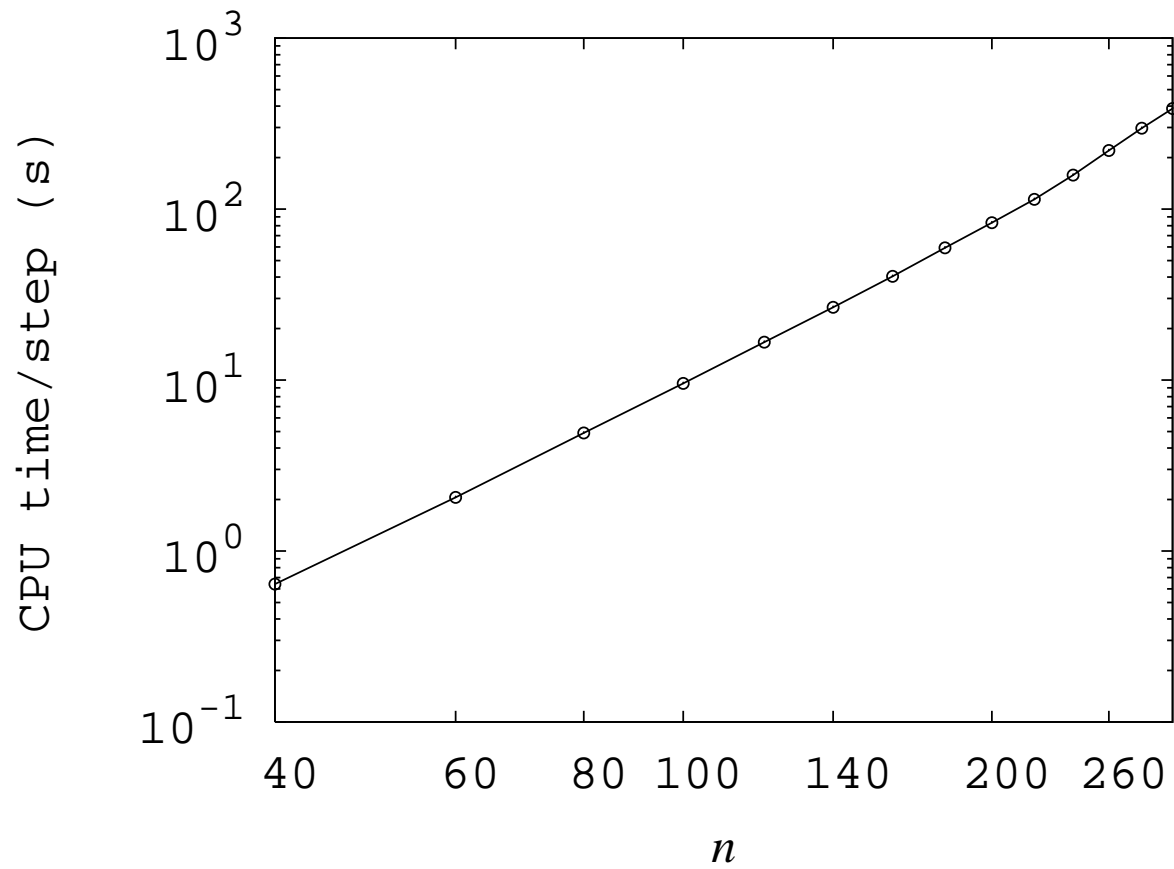
$$y' = \begin{pmatrix} -2 & 1 & 0 & 0 & \cdots & 0 \\ 1 & -2 & 1 & 0 & \cdots & 0 \\ 0 & 1 & -2 & 1 & \cdots & 0 \\ & & \vdots & & & \\ 0 & \cdots & & 1 & -2 & 1 \\ 0 & \cdots & & 0 & 1 & -2 \end{pmatrix} y$$

with $y(0) = (1, 0, \dots, 0)^T$

We integrate with problem sizes $n = 40, 60, \dots, 300$ for $t \in [0, 5]$

Order is 20, $\text{atol} = \text{rtol} = 10^{-12}$

For each n , VNODE-LP takes 8 steps



The computing time grows like n^3

Stiff problems

We integrate Van der Pol's equation (written as a first-order system)

$$y_1' = y_2$$

$$y_2' = \mu(1 - y_1^2)y_2 - y_1$$

with

$$y(0) = (2, 0)^T, \quad t_{\text{end}} = 200.$$

Number of steps and CPU time used by VNODE-LP:

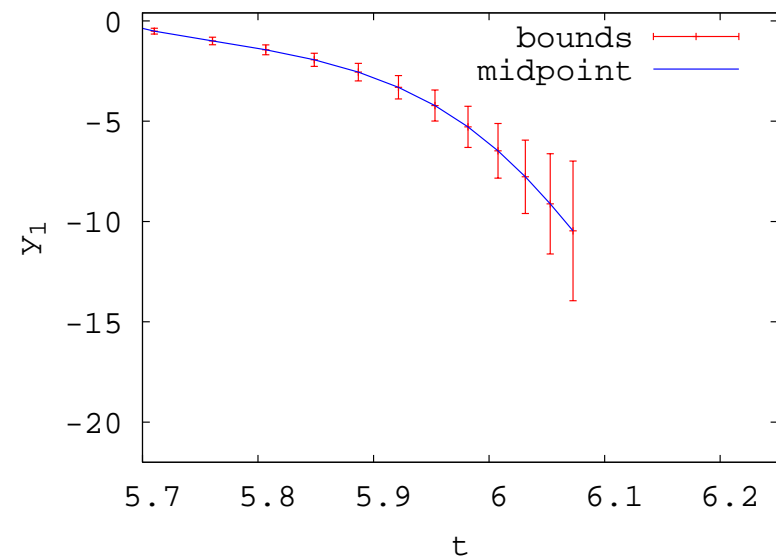
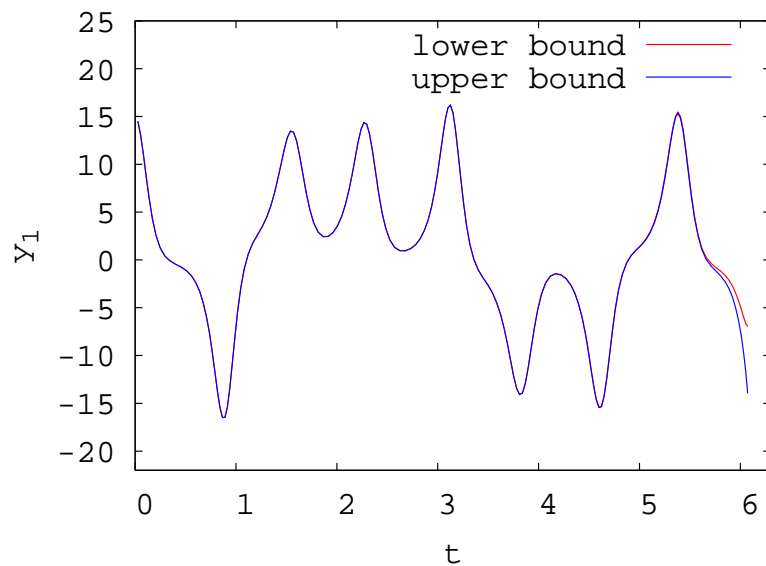
μ	steps	time (secs)
10^1	2377	0.8
10^2	11697	3.6
10^3	126459	36.1
10^4	1180844	336.4

We need an efficient method for stiff problems!

Interval initial conditions

Lorenz example with

$$y(0) \in \begin{pmatrix} 15 + [-10^{-4}, 10^{-4}] \\ 15 + [-10^{-4}, 10^{-4}] \\ 36 + [-10^{-4}, 10^{-4}] \end{pmatrix}$$



Bounds on y_1 versus t

On solving DAEs

Solve an IVP for a DAE system with n equations f_i in n dependent variables $x_j = x_j(t)$ of the form

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

Fully implicit; derivatives of order > 1 are allowed

Informally, the **index** of a DAE is the minimum number of differentiations needed to reduce it to an ODE

ODEs have index 0

The higher the index, the more difficult is to solve a DAE

Pryce's Structural Analysis (SA) + Taylor series expansion of the solution does not find high index difficult

Steps of Pryce's structural analysis

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

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

2. Find a Highest Value Transversal (HVT): n positions (i, j) in Σ with one entry in each row & column such that $\sum \sigma_{ij}$ is maximized

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

$$\begin{aligned} d_j - c_i &\geq \sigma_{ij} && \text{for all } i, j = 1, \dots, n \\ d_j - c_i &= \sigma_{ij} && \text{on the HVT} \end{aligned}$$

4. Form the **System Jacobian** \mathbf{J} , where

$$\mathbf{J}_{ij} = \begin{cases} \frac{\partial f_i}{\partial x_j^{(\sigma_{ij})}} & \text{if } d_j - c_i = \sigma_{ij} \\ 0 & \text{otherwise} \end{cases}$$

5. If there is a consistent point of the DAE at which \mathbf{J} is nonsingular, then

- DAE is solvable in a neighborhood of this point
- method shows how to reduce the DAE to an ODE system
- or alternatively **solve by Taylor series**

Example: simple pendulum, index-3 DAE ($g > 0$ and $L > 0$ are constants)

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

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

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

$$\Sigma = \begin{array}{ccccc} & x & y & \lambda & c_i \\ f & \left(\begin{array}{ccc} 2 & -\infty & 0 \\ -\infty & 2 & 0 \\ 0 & 0 & -\infty \end{array} \right) & 0 \\ g & & & & 0 \\ h & & & & 2 \\ d_j & 2 & 2 & 0 & \end{array}, \quad \mathbf{J} = \begin{bmatrix} \frac{\partial f}{\partial x''} & 0 & \frac{\partial f}{\partial \lambda} \\ 0 & \frac{\partial g}{\partial y''} & \frac{\partial g}{\partial \lambda} \\ \frac{\partial h}{\partial x} & \frac{\partial h}{\partial y} & 0 \end{bmatrix}$$

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

Work to date

J. Hoeffkens: the original DAE is converted into a generalized ODE and then Taylor models are applied to integrate the resulting ODE system

NN, J. Pryce

- theory of TC computation and evaluation of the System Jacobian
- DAETS code (C++): computes point, approximate solutions
- We know in principal how to do an interval DAETS

A. Walter and A. Griewank report of a similar implementation, but using the ADOL-C package

R. Barrio uses MATHEMATICA to compute Σ , set up a generalized ODE system, and then generate FORTRAN 77 code for evaluating TCs for the ODE system

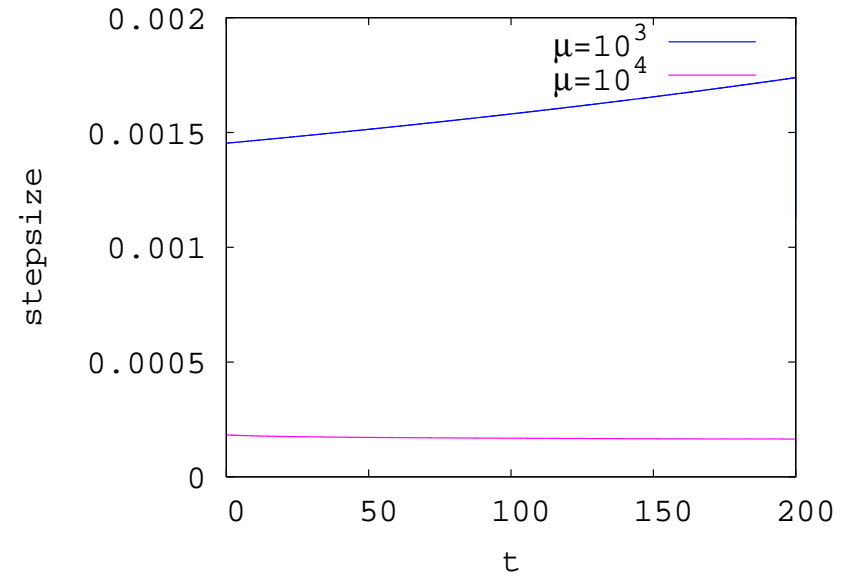
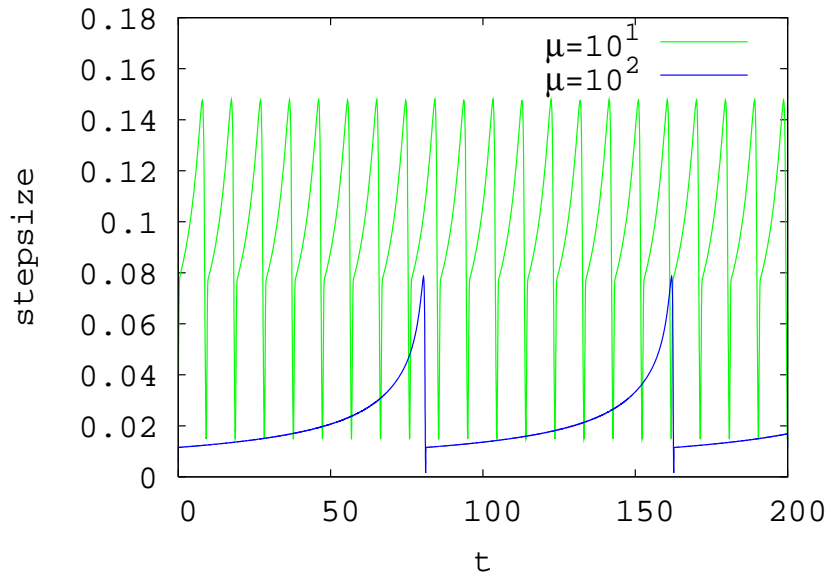
Conclusion

- Good progress has been made since AWA in speed, tightness of bounds, and applications
- The $O(n^3)$ complexity in fighting the wrapping effect is an obstacle towards solving larger problems
 - It seems very difficult to overcome it in general
 - Developing efficient methods for classes of problems may be a feasible approach
- An efficient method for stiff problems is needed
- An “interval version“ of DETEST, a test set for assessing approximate solvers for IVP ODEs, is needed

References

- [1] G. Corliss, *Survey of interval methods for ODE's*,
<http://www.eng.mu.edu/corlissg/Pubs/03Lyngby/Slides/>
- [2] N. S. Nedialkov, *Interval tools for ODEs and DAEs*,
<http://www.cas.mcmaster.ca/~nedialk/PAPERS/intvtools/intvtools.pdf>
- [3] N. S. Nedialkov and J. D. Pryce, *DAETS User Guide*,
<http://www.cas.mcmaster.ca/~nedialk/>
- [4] AWA, http://www.math.uni-wuppertal.de/~xsc/xsc/pxsc_software.html
- [5] COSY, http://bt.pa.msu.edu/index_cosy.htm
- [6] ValEncIA-IVP, <http://valencia-ivp.com/>
- [7] VNODE-LP, <http://www.cas.mcmaster.ca/~nedialk/vnodelp/>
- [8] DAETS
 - Demo version at <http://www.cas.mcmaster.ca/~nedialk/daets/>
 - Academic and commercial versions at Flintbox
<http://www.flintbox.com>

Van Der Pol: stepsize plots



Stepsize versus t for $\mu = 10, 10^2, 10^3, 10^4$

VNODE-LP: Example

The user has to

- specify an ODE problem of the form $y' = f(t, y)$ and
- provide a main program

Problem definition

Consider the Lorenz system

$$y_1' = \sigma(y_2 - y_1)$$

$$y_2' = y_1(\rho - y_3) - y_2$$

$$y_3' = y_1 y_2 - \beta y_3$$

σ , ρ , and β are constants

The Lorenz system is encoded as

```
1 < Lorenz 1 > ≡  
  template<typename var_type>  
  void Lorenz(int n, var_type *yp, const var_type *y, var_type t,  
              void *param)  
  {  
    interval sigma(10.0), rho(28.0);  
    interval beta = interval(8.0)/3.0;  
    yp[0] = sigma * (y[1] - y[0]);  
    yp[1] = y[0] * (rho - y[2]) - y[1];  
    yp[2] = y[0] * y[1] - beta * y[2];  
  }
```

This code is used in chunk 2.

Main program

```
2 < simple main program 2 > ≡  
  < Lorenz 1 >  
  int main()  
  {  
    < set initial condition and endpoint 3 >  
    < create AD object 4 >  
    < create a solver 5 >  
    < integrate (basic) 6 >  
    < check if success 7 >  
    < output results 8 >  
    return 0;  
  }
```

This code is used in chunk 10.

The initial condition and endpoint are represented as intervals

3 \langle set initial condition and endpoint 3 $\rangle \equiv$

```
const int n = 3;  
interval t = 0.0, tend = 20.0;  
iVector y(n);  
y[0] = 15.0;  
y[1] = 15.0;  
y[2] = 36.0;
```

This code is used in chunk 2.

We create an automatic differentiation (AD) object of type

FADBAD_AD

4 \langle create AD object 4 $\rangle \equiv$

```
AD *ad = new FADBAD_AD(n, Lorenz, Lorenz);
```

This code is used in chunk 2.

Now, we create a solver:

```
5 < create a solver 5 > ≡  
  VNODE *Solver = new VNODE(ad);
```

This code is used in chunk 2.

The integration is carried out by the *integrate* function

When *integrate* returns, either $t = tend$ or $t \neq tend$

In both cases, *y* contains the ODE solution at *t*

```
6 < integrate (basic) 6 > ≡  
  Solver→integrate(t, y, tend);
```

This code is used in chunk 2.

We check if an integration is successful by calling *Solver*→*successful*():

```
7 < check if success 7 > ≡  
  if (¬Solver→successful( ))  
    cout << "VNODE-LP_ could_ not_ reach_ t_ =_" << tend << endl;
```

This code is used in chunk 2.

We output the computed enclosure of the solution at *t* by

```
8 < output results 8 > ≡  
  cout << "Solution_ enclosure_ at_ t_ =_" << t << endl;  
  printVector(y);
```

This code is used in chunk 2.

We store our program in the file `basic.cc`

```
10 <basic.cc 10 > ≡  
#include <ostream>  
#include "vnode.h"  
using namespace std;  
using namespace vnodelp;  
<simple main program 2 >
```

The output is

```
Solution enclosure at t = [20,20]  
14.30 [38147841691429,44705774970780]  
9.5 [785914121310753,801299927130011]  
39.038 [2370498042590,4113960724549]
```

Execution is about 8 times faster than AWA for similar tightness