Sub-ODEs Simplify Taylor Series Algorithms for Ordinary Differential Equations

Ned Nedialkov
McMaster University, Hamilton, Canada

John Pryce
Cardiff University, UK

Go20 Conference on Scientific Computing and Software
May 19–23, 2025, Marsalforn, Gozo, Malta

Outline

Reference: NN & JP, *Sub-ODEs Simplify Taylor Series Algorithms for Ordinary Differential Equations*, Submitted to SISC, December 2024. https://arxiv.org/abs/2503.21078

Introduction

We are concerned with Taylor series solution of an initial-value problem (IVP) for an ordinary differential equation (ODE) system:

$$\dot{x} = f(t, x), \quad x(t_0) = x_0.$$

- We assume $f(t, x)$ consists of basic arithmetic operations $(+, -, \times, /)$ and standard functions such as $\exp, \sin$, etc.
- Non-differentiable functions like $|\cdot|$, $\min$, $\max$, and conditional structures (e.g. "if" statements) are excluded.

Taylor method

A Taylor method (TM) for $\dot{x} = f(t, x),\ x(t_0) = x_0$

- computes the Taylor series (TS), to some order, of the solution at the current t, and
- sums the TS with a suitable step size to move to the next point.

▷ A TM requires $f(t, x)$ to be provided as code, which is executed in the form of recurrences that compute Taylor coefficients.

▷ Recurrences are needed for:
  - ▷ basic arithmetic operations (BAOs)
  - ▷ standard functions: $\exp$, $\sin$, etc.

▷ Most TM implementations use individual recurrences for each standard function.

▷ Our approach eliminates the need for individual recurrences.

Taylor coefficients (TCs)

Automatic (or algorithmic) differentiation, AD, gives a practical scheme for computing coefficients.

Example: Consider $\dot{x} = x^2$, $x(t_0) = x_0$.

- Assume a series $x(t_0 + s) = x_0 + x_1 s + x_2 s^2 + \cdots$.
  Then

$$\dot{x} = x_1 + 2x_2 s + 3x_3 s^2 + \cdots$$
$$x^2 = x_0^2 + (2x_0 x_1)s + (2x_0 x_2 + x_1^2)s^2 + \cdots .$$

- Starting with x_0, and equating these term by term:

$$x_1 = x_0^2$$
$$x_2 = \tfrac{1}{2}(2x_0 x_1)$$
$$x_3 = \tfrac{1}{3}(2x_0 x_2 + x_1^2)$$
$$\vdots$$

ODE's code-list

- Consider an ODE of size n, i.e., $x = (x_1, \ldots, x_n)$.
- The code for f is implemented as a code-list (CL), a sequence of assignments

$$\begin{cases} \dot{x}_j = \widehat{x}_j, & j = 1{:}n, \\ x_{n+1} = t, & j = n{+}1, \\ x_j = \phi_j(x_{\prec j}), & j = n{+}2 : N. \end{cases}$$

- Input values are $x_1, \ldots, x_n, t$; x_{n+1} is an alias for t. The CL sets intermediate variables $x_{n+2}, \ldots, x_N$.
- $x_{\prec j}$ means the x_i's on which x_j depends; they must all have $i < j$, i.e. have already been computed.
- ϕ_j are given functions.
- CL specifies indices $\mathit{out}(1), \ldots, \mathit{out}(n)$ in $1{:}N$ such that $\widehat{x}_j = x_{\mathit{out}(j)}$ form the output vector $\widehat{x} = f(t, x)$.

Example

- Consider the simple ODE $\dot{x} = e^{-x}$.
- Its code-list is

$$\dot{x} = \widehat{x}$$
$$u = -x$$
$$\widehat{x} = \exp(u)$$

Sub-ODE idea

Take $\widehat{x} = \exp(u)$ in the above CL.

- A Taylor solver needs to compute

$$\widehat{x}(t) = \exp(u(t)), \quad \text{where } u(t) \text{ is a TS}$$

(both input and output are truncated to some order).

- We use that

$$\widehat{x} = \exp(u) \quad \text{obeys the ODE } \mathrm{d}\widehat{x}/\mathrm{d}u = \widehat{x}, \text{ whence}$$
$$\dot{\widehat{x}} = \widehat{x}\dot{u}, \qquad \text{since } u, \widehat{x} \text{ are both functions of } t.$$

- The sub-ODE method replaces in the CL

$$\widehat{x} = \exp(u) \quad \text{by its sub-ODE} \quad \dot{\widehat{x}} = \widehat{x}\dot{u}$$

Sub-ODEs make a DAE

- CL of $\dot{x} = e^{-x}$:

$$\dot{x} = \widehat{x}, \qquad u = -x, \qquad \widehat{x} = \exp(u) \tag{1}$$

- Replace $\widehat{x} = \exp(u)$ by its sub-ODE $\dot{\widehat{x}} = \widehat{x}\dot{u}$:

$$\dot{x} = \widehat{x}, \qquad u = -x, \qquad \dot{\widehat{x}} = \widehat{x}\dot{u} \tag{2}$$

- The CL (2) represents a differential-algebraic equation (DAE), so ODE theory no longer applies.
- Namely, to cast (2) as an ODE, one must differentiate $u = -x$:

$$\dot{x} = \widehat{x}, \qquad \dot{u} = -\dot{x}, \qquad \dot{\widehat{x}} = \widehat{x}\dot{u}$$

- How to get the TCs for x?

- Write $x(t) = x(t_0 + s) = x_0 + x_1 s + x_2 s^2 + \cdots$; similarly for $u, \widehat{x}$. Inserting in (2) gives

$$\underbrace{1x_1 + 2x_2 s + 3x_3 s^2 + \cdots}_{\dot{x}} = (\underbrace{\widehat{x}_0 + \widehat{x}_1 s + \widehat{x}_2 s^2 + \cdots}_{\widehat{x}}),$$

$$\underbrace{u_0 + u_1 s + u_2 s^2 + \cdots}_{u} = -(\underbrace{x_0 + x_1 s + x_2 s^2 + \cdots}_{x}),$$

$$\underbrace{1\widehat{x}_1 + 2\widehat{x}_2 s + 3\widehat{x}_3 s^2 + \cdots}_{\dot{\widehat{x}}} = (\underbrace{\widehat{x}_0 + \widehat{x}_1 s + \widehat{x}_2 s^2 + \cdots}_{\widehat{x}})(\underbrace{1u_1 + 2u_2 s + 3u_3 s^2 + \cdots}_{\dot{u}}).$$

- Expanding and equating terms on either side gives

$k = 0$	$k = 1$	$k = 2$	$\cdots$
$x_0 =$ user's IV	$x_1 = \frac{1}{1}\widehat{x}_0$	$x_2 = \frac{1}{2}\widehat{x}_1$	
$u_0 = -x_0$	$u_1 = -x_1$	$u_2 = -x_2$	
$\widehat{x}_0 = \exp(u_0)$	$\widehat{x}_1 = \frac{1}{1}(\widehat{x}_0 u_1)$	$\widehat{x}_2 = \frac{1}{2}(\widehat{x}_1 u_1 + \widehat{x}_0 2u_2)$	

IV: initial value

$k=0$	$k=1$	$k=2$	$\cdots$
$x_0 =$ user's IV	$x_1 = \frac{1}{1}\widehat{x}_0$	$x_2 = \frac{1}{2}\widehat{x}_1$	
$u_0 = -x_0$	$u_1 = -x_1$	$u_2 = -x_2$	
$\widehat{x}_0 = \exp(u_0)$	$\widehat{x}_1 = \frac{1}{1}(\widehat{x}_0 u_1)$	$\widehat{x}_2 = \frac{1}{2}(\widehat{x}_1 u_1 + \widehat{x}_0 2u_2)$	

$k = 0$:

- Set $x_0 =$ the user's IV to start the recursion.
- When u_0 is available, set $\widehat{x}_0 = \exp(u_0)$, the built-in IV of the sub-ODE.

In general:

- A standard function is called at $k = 0$.
- For $k > 0$, only BAOs are used.

Despite the sub-ODE making a DAE, the recurrences work!

The sub-ODE concept

Suppose:

- $v = g(u)$ is an m-vector function of scalar u,
- u, v are variables in the CL of the ODE,
- we can express $\frac{\mathrm{d}v}{\mathrm{d}u}$ as $h(u, v)$, i.e.

$$g'(u) = h(u, g(u)). \tag{3}$$

Then

- u, v denote $u(t), v(t)$, and $v = g(u)$ means $v(t) = g(u(t))$.
- (3) implies

$$\dot{v} = h(u, v)\dot{u} \quad \text{for short, termed a sub-ODE.}$$

- The sub-ODE method replaces in ODE's CL each assignment $v = g(u)$ by $\dot{v} = h(u, v)\dot{u}$.

Examples of sub-ODEs produced by standard functions:

(i) $v = \exp(u)$ has $\dot{v} = v\dot{u}$, $h(u, v) = v$

(ii) $v = u^c$, (c constant) has $\dot{v} = (cv/u)\dot{u}$, $h(u, v) = cv/u$

and, defining $\mathrm{cs}(u) = \left[\begin{smallmatrix} \cos(u) \\ \sin(u) \end{smallmatrix}\right]$,

(iii) $v = \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} = \mathrm{cs}(u)$ has $\dot{v} = \begin{bmatrix} -v_2 \\ v_1 \end{bmatrix} \dot{u}$, $h(u, v) = \begin{bmatrix} -v_2 \\ v_1 \end{bmatrix}$

Computational kernel

Consists of five operations: $+,\ -,\ \times,\ \div,\ \odot$.

- BAOs Given TCs of u and v to order k:

$$(u \pm v)_k = u_k \pm v_k,$$

$$(u \times v)_k = \sum_{r=0}^{k} u_r v_{k-r},$$

$$(u/v)_k = \frac{1}{v_0}\Big(u_k - \sum_{r=0}^{k-1} v_{k-r}(u/v)_r\Big), \quad (v_0 \neq 0).$$

- $\odot$ (sub-ODE) $v = g(u)$ with sub-ODE $\dot{v} = h(u,v)\dot{u}$.

  Given TCs of u to order k:

$$v_k = (h \odot_g u)_k = \begin{cases} g(u_0) & k = 0 \\ \frac{1}{k}\sum_{i=1}^{k} i u_i h_{k-i} & k > 0 \end{cases}$$

Sub-ODEs from a DAE viewpoint

- Inserting sub-ODEs unavoidably turns the ODE into a DAE.
- E.g. in previous $\dot{x} = e^{-x}$ example:

ODE

$$\begin{cases} \dot{x} = \widehat{x} \\ u = -x \\ \widehat{x} = \exp(u) \end{cases}$$

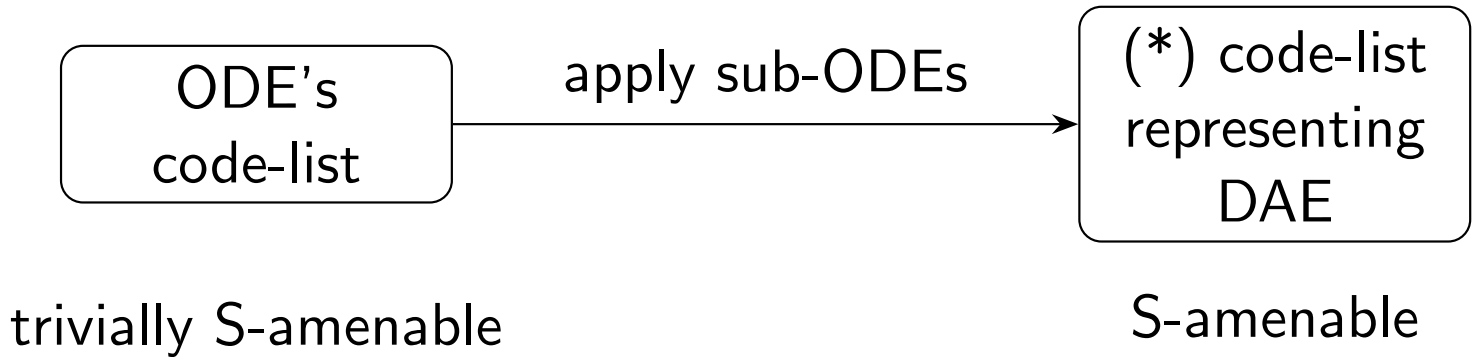
$\longrightarrow$

DAE

$$\begin{cases} \dot{x} = \widehat{x} \\ u = -x \\ \dot{\widehat{x}} = \widehat{x}\dot{u} \end{cases}$$

- ODE existence & uniqueness theory no longer applies.
- If we insert many, how do we know they don't interfere with each other and make nonsense?
- We resolved this using DAE S-amenability theory.

- A DAE is S-amenable if analyzing its sparsity tells you a scheme to solve it numerically.
- S-amenability test: a certain Jacobian must be non-singular.
- Most modeling software (e.g. Modelica) that creates DAEs implicitly assumes they are S-amenable.
- Every S-amenable DAE has a solution scheme by TS.



- An ODE is trivially an S-amenable DAE.
- We have shown that an S-amenable DAE remains so after inserting a single sub-ODE anywhere in it.
- By induction, it remains S-amenable after inserting any number of sub-ODEs.
- DAE's standard solution scheme by TS coincides with the recurrences defined by code-list (*).

ODETS solver

- We have implemented the sub-ODE method in a MATLAB solver **odets**, ODE by Taylor Series.
- The CL generation is done through operator overloading.
- **odets** evaluates the CL on each integration step to compute TCs.
- This is a variable-stepsize integrator, of fixed order determined at the start purely as a function of the accuracy tolerances.

ODETS performance

We compare the performance of **odets** against MATLAB 's built-in ODE solvers on the following problems:

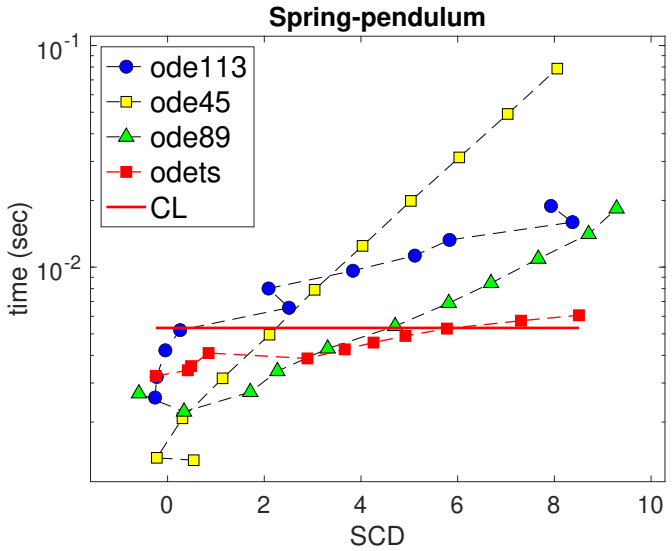
- Nonlinear Spring-Pendulum: A 2D mechanical system modeled by two second-order ODEs, reduced to four first-order equations.
- Pleiades Problem: A problem from the Test Set for IVP Solvers; describes the motion of 7 stars, with a size of 28 when reduced to first-order.
- Brusselator System: Models diffusion in a chemical reaction, becoming increasingly stiff and sparse as the problem size increases.

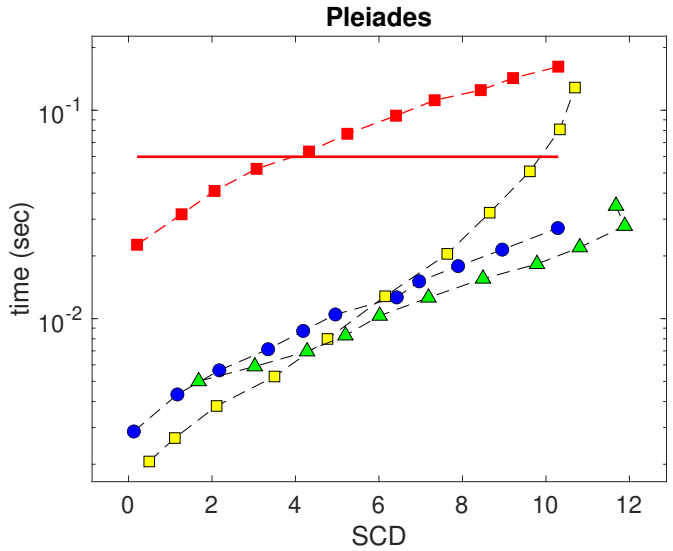
Work-precision diagrams

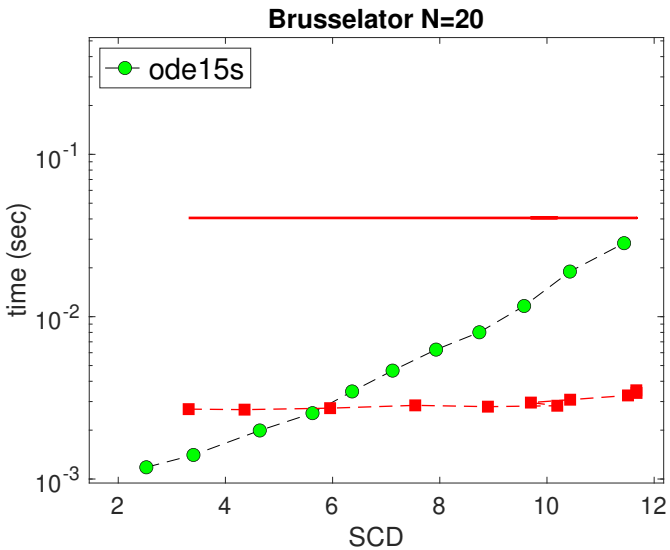
- We plot work-precision diagrams: CPU time versus significant correct digits (SCD), defined as

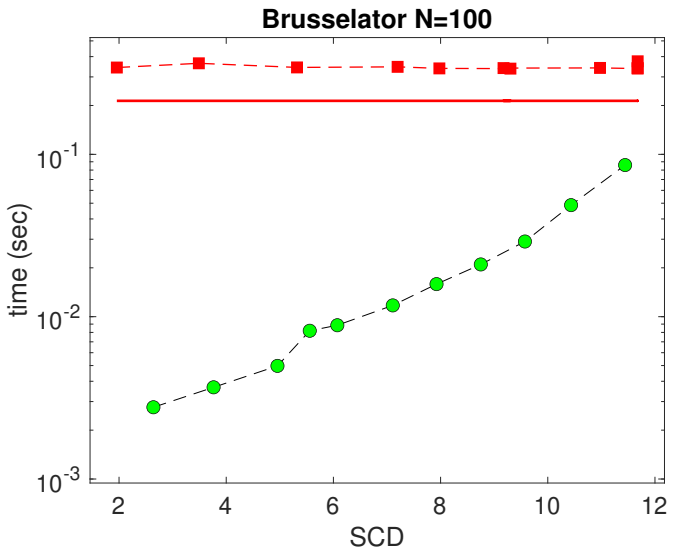
$$\text{SCD} = -\log_{10}\left(\|\text{relative error at } t_{\text{end}}\|_\infty\right).$$

- Work-precision diagrams are a standard way to compare the performance of different solvers.
- CPU times are measured on an Apple M3 MacBook Air 2024.
- Spring-pendulum, Pleiades: Comparison between nonstiff solvers `ode113`, `ode45`, `ode89`, and `odets`.
- Brusselator: Results for problem sizes $N = 20$ and $N = 100$, comparing the stiff solver `ode15s` and `odets`.









CL: time for CL generation, **odets**: time for integration only.

Back story

- Automatic solution of IVPs for ODEs by TS goes back (at least) to Moore (1966).
- So does the idea of converting an ODE to an equivalent rational ODE; he gives an example where $\cos$, $\exp$, and $\log$ occur and are eliminated.
- To our knowledge, Neidinger (2013) is the first person to give an embedding *method* that can, in principle, be automated—an operator he calls `ddot`, nearly equivalent to our $\odot$.
- M&N focus on recurrence relations for calculating TCs without considering the differential system these might represent.
- Corless and Ilie (2008) note that one can convert a DAE to an equivalent one using only BAOs.
  They treat the differential system but do not consider recurrence relations.

ADTAYL

ADTAYL, Automatic Differentiation Taylor Series is a MATLAB package with two parts:

- **adtayl** class for manipulating functions represented as TS, and
- **odets**.
- An **adtayl** object is an array with each element a polynomial, representing a function $x(t)$ expanded about t_0 up to order p:

$$x(t_0 + s) = x_0 + x_1 s + \cdots + x_p s^p + \cdots, \quad x_k = \frac{x^{(k)}(t_0)}{k!}$$

- Each element stores coefficients to order p, e.g. $x_0, x_1, \ldots, x_p$.

- **adtayl** objects behave like MATLAB arrays:
  - Indexing: `x(i,j)`, `x(:,2:end)`, ...
  - Elementwise arithmetic: `x + y`, `x .* y`, `x .^ 2`, ...
  - Standard functions: `sin(x)`, `exp(x)`, ...
  - Linear algebra, `x*y`, `x\y`, inverse `inv(x)`, ...
  - ...
- Coefficients can be real or complex:
  - `double` (default)
  - `vpa` (for variable precision arithmetic)
- **adtayl** implements power series algebra numerically, where all terms of order higher than p are dropped.

Application: Computing Lie derivatives

Let $f : \mathbb{R}^n \to \mathbb{R}^n$ be a smooth vector field, and let $h : \mathbb{R}^n \to \mathbb{R}$ be a smooth output function.

- The Lie derivative (LD) of h along f is defined as

$$L_f h(x) = \frac{\partial h}{\partial x}(x) \cdot f(x).$$

- Higher-order LD are defined recursively:

$$L_f^0 h(x) = h(x), \quad L_f^k h(x) = \frac{\partial}{\partial x}\left(L_f^{k-1} h(x)\right) \cdot f(x), \quad k \geq 1.$$

- Higher-order LD arise in control engineering, e. g. in the design of high-gain observers and differentiators.

- Given

$$\dot{x} = f(x), \quad y(t) = h(x), \quad x(0) = x_0,$$

we wish to compute $y^{(k)}(0)$ for $k \geq 1$.

- From the definition of Lie derivatives:

$$y(0) = L_f^0 h(x_0) = h(x_0), \qquad y^{(k)}(0) = L_f^k h(x_0), \quad k \geq 1.$$

- Then

$$y(t) = \sum_{k=0}^{\infty} \frac{y^{(k)}(0)}{k!} t^k = \sum_{k=0}^{\infty} y_k t^k = \sum_{k=0}^{\infty} \frac{L_f^k h(x_0)}{k!} t^k.$$

- Given TCs of $x(t)$ up to order p, and since $y = h(x)$, compute the TCs of y to order p:

$$h(x_0 + x_1 t + \cdots + x_p t^p) \quad \rightarrow \quad y_0 + y_1 t + \cdots + y_p t^p.$$

Computation with ADTAYL

Given f, h, x_0, and desired order p:

- Compute the TCs $x_0, x_1, \ldots, x_p$ of the solution $x(t)$ to $\dot{x} = f(x), \quad x(0) = x_0$.
- Store them in an **adtayl** object `x`.
- Evaluate

```
y = h(x)
```

- The k-th Lie derivative at x_0 is:

$$L_f^k h(x_0) = k! \cdot y_k = \texttt{factorial(k) * y.gettc(k)}$$

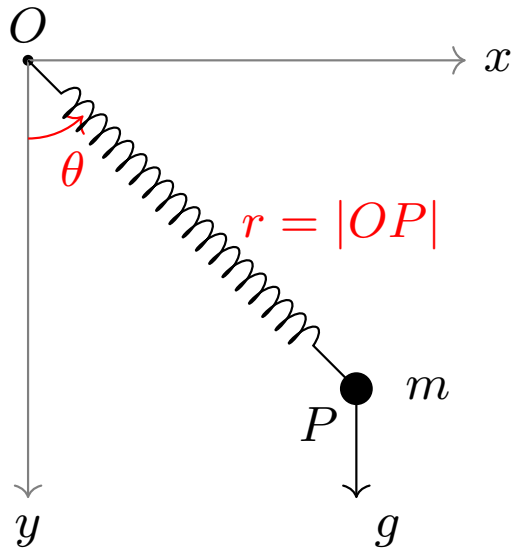
`gettc(k)` extracts the k-th Taylor coefficient of `y`.

Conclusion

- Sub-ODEs are a way to formulate recurrences for TCs.
- Proving they are theoretically sound was non-trivial: in general, they unavoidably transform an ODE to a DAE.
- A broad class of standard functions is expressible in terms of five operations $+$, $-$, $\times$, $\div$, $\odot$.
- **odets**'s performance is comparable to that of MATLAB's built-in nonstiff solvers.
- **odets** can reliably integrate with small tolerances (e.g. 10^{-100}) using MATLAB's variable precision arithmetic (vpa).
- Lie derivatives of arbitrary order can be computed efficiently using **odets** and **adtayl**.
- We are completing a user guide and implementing features such as event location and the computation of high-order Lie derivatives.

Code-list example: a nonlinear spring-pendulum

- Massless spring with a point mass m
- a natural length, k constant
- Nonlinear potential energy



With coordinates (r, θ), the Euler-Lagrange equations are

$$\begin{aligned}
\dot{r} &= s, \\
\dot{s} &= r\omega^2 + g\cos\theta - \tfrac{k}{m}\left((r-a) + 1 - e^{-(r-a)}\right) \\
\dot{\theta} &= \omega, \\
\dot{\omega} &= (-g\sin\theta - 2s\omega)/r
\end{aligned} \quad .$$

Line	Kind	Op	Mode	R1	R2	Imm	Expression
1	ODE		R	2			$\dot{x}_1 = x_2$
2	ODE		R	18			$\dot{x}_2 = x_{18}$
3	ODE		R	4			$\dot{x}_3 = x_4$
4	ODE		R	23			$\dot{x}_4 = x_{23}$
5	ALG	sub	RI	1		1	$x_5 = x_1 - 1$
6	ALG	mul	RR	1	4		$x_6 = x_1 \cdot x_4$
7	ALG	mul	RR	6	4		$x_7 = x_6 \cdot x_4$
8	SUB	cs	RR	10	3		$\begin{bmatrix} x_8 \\ x_9 \end{bmatrix} = \begin{bmatrix} x_{10} \\ x_8 \end{bmatrix} \odot_{\text{cs}} x_3$
9	SUB		RR	8	3		
10	ALG	sub	IR		9	0	$x_{10} = 0 - x_9$
11	ALG	mul	IR		8	9.81	$x_{11} = 9.81 \cdot x_8$
12	ALG	add	RR	7	11		$x_{12} = x_7 + x_{11}$
13	ALG	add	RI	5		1	$x_{13} = x_5 + 1$
14	ALG	sub	IR		5	0	$x_{14} = 0 - x_5$
15	SUB	exp	RR	15	14		$x_{15} = x_{15} \odot_{\text{exp}} x_{14}$
16	ALG	sub	RR	13	15		$x_{16} = x_{13} - x_{15}$
17	ALG	mul	IR		16	40	$x_{17} = 40 \cdot x_{16}$
18	ALG	sub	RR	12	17		$x_{18} = x_{12} - x_{17}$
19	ALG	mul	IR		2	-2	$x_{19} = -2 \cdot x_2$
20	ALG	mul	RR	19	4		$x_{20} = x_{19} \cdot x_4$
21	ALG	mul	IR		9	9.81	$x_{21} = 9.81 \cdot x_9$
22	ALG	sub	RR	20	21		$x_{22} = x_{20} - x_{21}$
23	ALG	div	RR	22	1		$x_{23} = x_{22}/x_1$