Interfacing OCaml with Sundials for numerical simulation

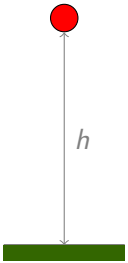
Timothy Bourke

Inria Paris

March, 2022

Bouncing ball

model



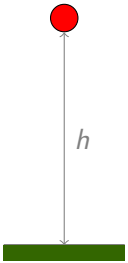
$$F = m \cdot a$$

$$m \cdot -g = m \cdot \frac{d^2 h(t)}{dt^2}$$

$$\frac{d^2 h(t)}{dt^2} = -g$$

Bouncing ball

model



$$F = m \cdot a$$

$$m \cdot -g = m \cdot \frac{d^2 h(t)}{dt^2}$$

$$\frac{d^2 h(t)}{dt^2} = -g$$

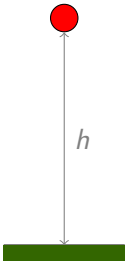
$$\dot{v} = -g \qquad v(0) = v_0$$

$$\dot{h} = v \qquad h(0) = h_0$$

First-order ODE

Bouncing ball

model



$$F = m \cdot a$$

$$m \cdot -g = m \cdot \frac{d^2 h(t)}{dt^2}$$

$$\frac{d^2 h(t)}{dt^2} = -g$$

$$\dot{v} = -g \qquad v(0) = v_0$$

$$\dot{h} = v \qquad h(0) = h_0$$

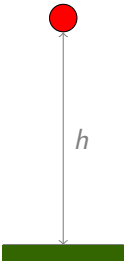
First-order ODE

$$v(t) = v_0 + \int_0^t -g \, .d\tau$$

$$h(t) = h_0 + \int_0^t v(\tau) \, .d\tau$$

Bouncing ball

model



$$F = m \cdot a$$

$$m \cdot -g = m \cdot \frac{d^2 h(t)}{dt^2}$$

$$\frac{d^2 h(t)}{dt^2} = -g$$

$[\dot{v}; \dot{h}] = \mathbf{f}(t, [v; h])$

Solver

approximation

$\mathbf{y_i} = [v_0; h_0]$

$$\begin{aligned} \dot{v} &= -g \\ \dot{h} &= v \end{aligned}$$

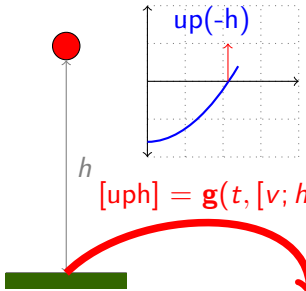
$$\begin{aligned} v(0) &= v_0 \\ h(0) &= h_0 \end{aligned}$$

$$\begin{aligned} v(t) &= v_0 + \int_0^t -g \, .d\tau \\ h(t) &= h_0 + \int_0^t v(\tau) \, .d\tau \end{aligned}$$

First-order ODE

Bouncing ball

model



$$F = m \cdot a$$

$$m \cdot -g = m \cdot \frac{d^2 h(t)}{dt^2}$$

$$\frac{d^2 h(t)}{dt^2} = -g$$

First-order ODE

Numerical simulation

- Forward Euler method
  - » Continuous model:
    $\dot{y}(t) = f(t, y(t)) \qquad y(0) = y_0$
  - » Discrete approximation:
    $y_{n+1} = y_n + h \cdot f(n \cdot h, y_n)$
    where h is the *step size*
  - » Approximates a curve with straight line segments
  - » Smaller h: more accurate, more computation

Numerical simulation

- Forward Euler method
  - » Continuous model:
    $\dot{y}(t) = f(t, y(t)) \qquad y(0) = y_0$
  - » Discrete approximation:
    $y_{n+1} = y_n + h \cdot f(n \cdot h, y_n)$
    where h is the *step size*
  - » Approximates a curve with straight line segments
  - » Smaller h: more accurate, more computation

- Backward Euler method
  - » $y_{n+1} = y_n + h \cdot f((n+1) \cdot h, y_{n+1})$
  - » *Implicit method*: needs to solve an algebraic equation for y_{n+1}
  - » Non-linear solver: Fixed-point iteration, Newton method, ...

Numerical simulation

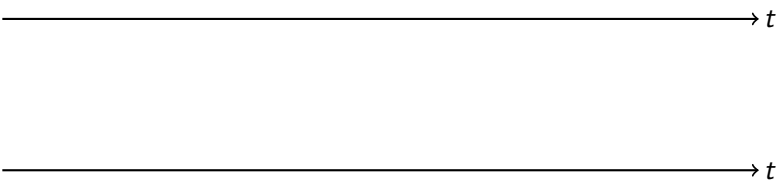
- Forward Euler method
  - » Continuous model:
    $\dot{y}(t) = f(t, y(t)) \qquad y(0) = y_0$
  - » Discrete approximation:
    $y_{n+1} = y_n + h \cdot f(n \cdot h, y_n)$
    where h is the *step size*
  - » Approximates a curve with straight line segments
  - » Smaller h: more accurate, more computation
- Backward Euler method
  - » $y_{n+1} = y_n + h \cdot f((n+1) \cdot h, y_{n+1})$
  - » *Implicit method*: needs to solve an algebraic equation for y_{n+1}
  - » Non-linear solver: Fixed-point iteration, Newton method, ...
- Runge-Kutta method (1900)

Numerical simulation

- Forward Euler method
  - » Continuous model:
    $\dot{y}(t) = f(t, y(t)) \qquad y(0) = y_0$
  - » Discrete approximation:
    $y_{n+1} = y_n + h \cdot f(n \cdot h, y_n)$
    where h is the *step size*
  - » Approximates a curve with straight line segments
  - » Smaller h: more accurate, more computation
- Backward Euler method
  - » $y_{n+1} = y_n + h \cdot f((n+1) \cdot h, y_{n+1})$
  - » *Implicit method*: needs to solve an algebraic equation for y_{n+1}
  - » Non-linear solver: Fixed-point iteration, Newton method, ...
- Runge-Kutta method (1900)
- Adams-Moulton Formulas
- Backward Differentiation Formulas (BDFs)

Solver execution

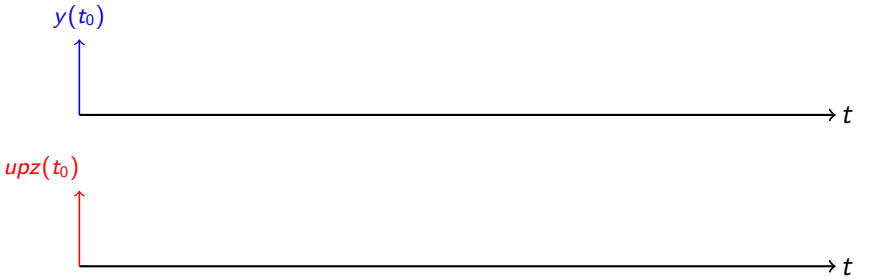
Give solver two functions: $\dot{y} = f_\sigma(t, y)$, $upz = g_\sigma(t, y)$



- Bigger and bigger steps (bound by h_{min} and h_{max})
- t does not necessarily advance monotonically
  - » No side-effects within f or g

Solver execution

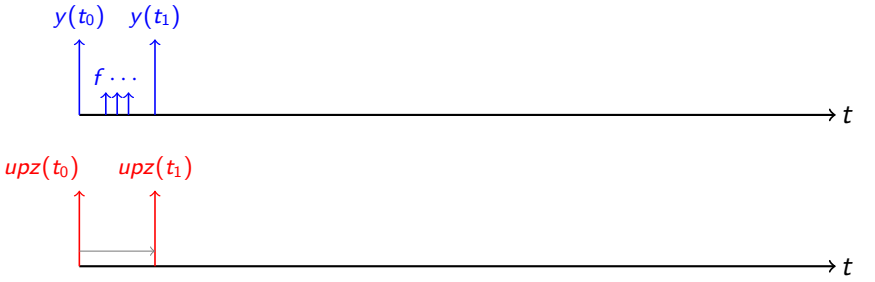
Give solver two functions: $\dot{y} = f_\sigma(t, y)$, $upz = g_\sigma(t, y)$



- Bigger and bigger steps (bound by h_{min} and h_{max})
- t does not necessarily advance monotonically
  - » No side-effects within f or g

Solver execution

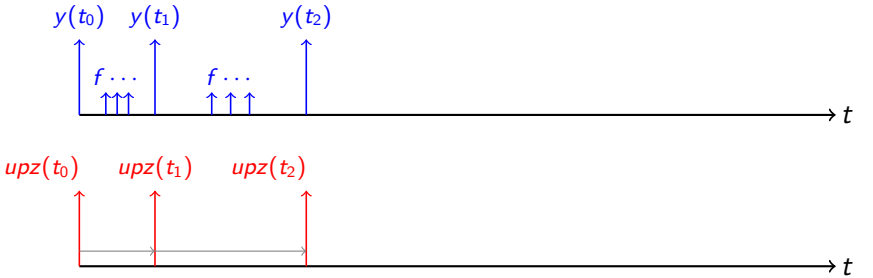
Give solver two functions: $\dot{y} = f_\sigma(t, y)$, $upz = g_\sigma(t, y)$



- Bigger and bigger steps (bound by h_{min} and h_{max})
- t does not necessarily advance monotonically
  - » No side-effects within f or g

Solver execution

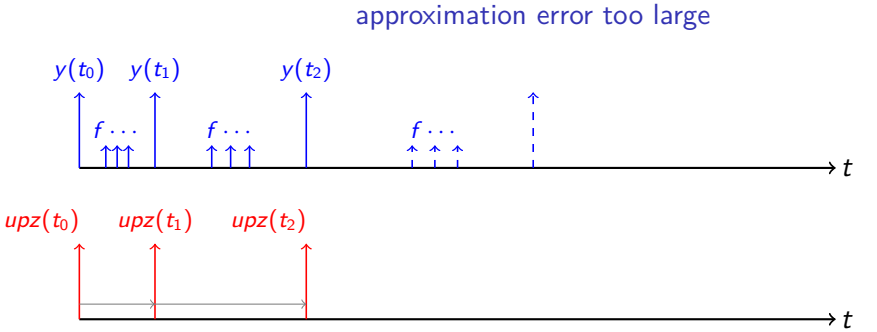
Give solver two functions: $\dot{y} = f_\sigma(t, y)$, $upz = g_\sigma(t, y)$



- Bigger and bigger steps (bound by h_{min} and h_{max})
- t does not necessarily advance monotonically
  - » No side-effects within f or g

Solver execution

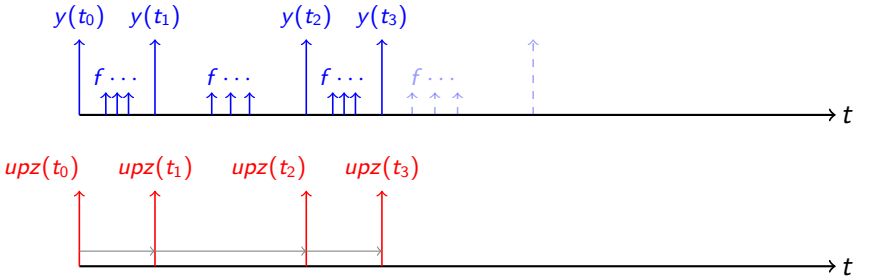
Give solver two functions: $\dot{y} = f_\sigma(t, y)$, $upz = g_\sigma(t, y)$



- Bigger and bigger steps (bound by h_{min} and h_{max})
- t does not necessarily advance monotonically
  - » No side-effects within f or g

Solver execution

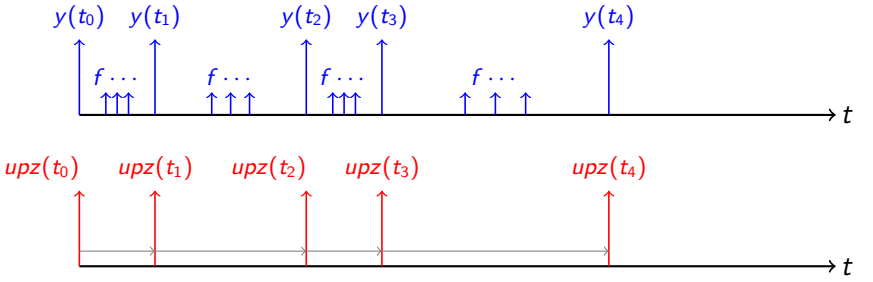
Give solver two functions: $\dot{y} = f_\sigma(t, y)$, $upz = g_\sigma(t, y)$



- Bigger and bigger steps (bound by h_{min} and h_{max})
- t does not necessarily advance monotonically
  - » No side-effects within f or g

Solver execution

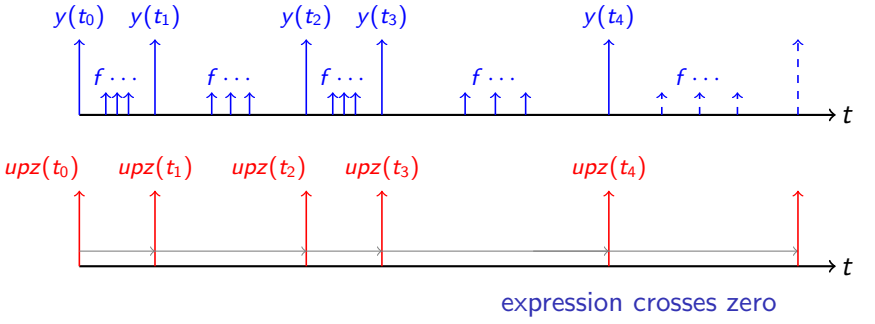
Give solver two functions: $\dot{y} = f_\sigma(t, y)$, $upz = g_\sigma(t, y)$



- Bigger and bigger steps (bound by h_{min} and h_{max})
- t does not necessarily advance monotonically
  - » No side-effects within f or g

Solver execution

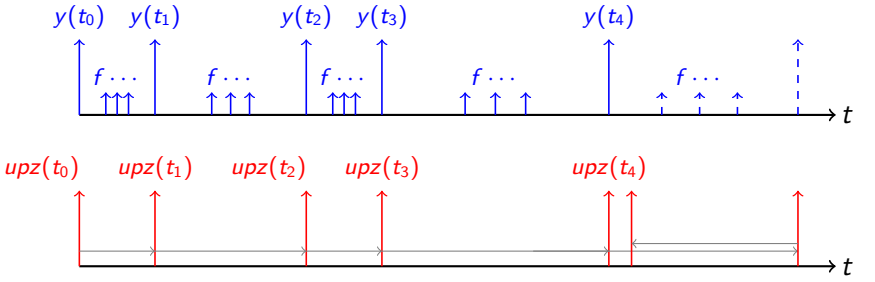
Give solver two functions: $\dot{y} = f_\sigma(t, y)$, $upz = g_\sigma(t, y)$



- Bigger and bigger steps (bound by h_{min} and h_{max})
- t does not necessarily advance monotonically
  - » No side-effects within f or g

Solver execution

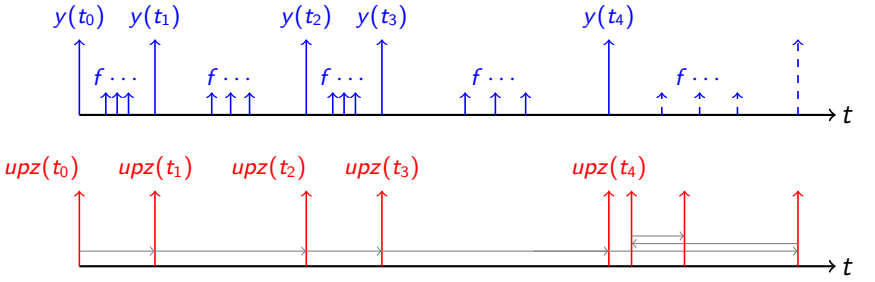
Give solver two functions: $\dot{y} = f_\sigma(t, y)$, $upz = g_\sigma(t, y)$



- Bigger and bigger steps (bound by h_{min} and h_{max})
- t does not necessarily advance monotonically
  - » No side-effects within f or g

Solver execution

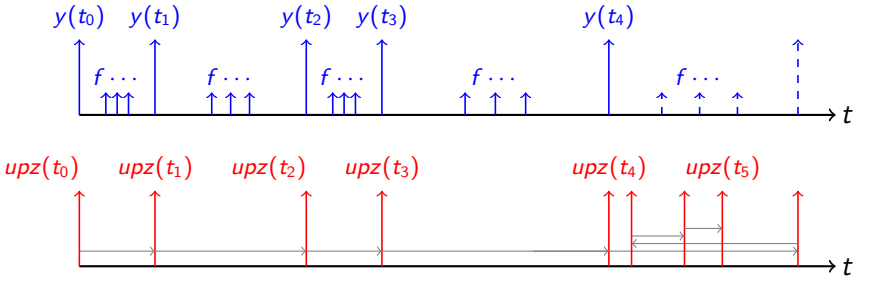
Give solver two functions: $\dot{y} = f_\sigma(t, y)$, $upz = g_\sigma(t, y)$



- Bigger and bigger steps (bound by h_{min} and h_{max})
- t does not necessarily advance monotonically
  - » No side-effects within f or g

Solver execution

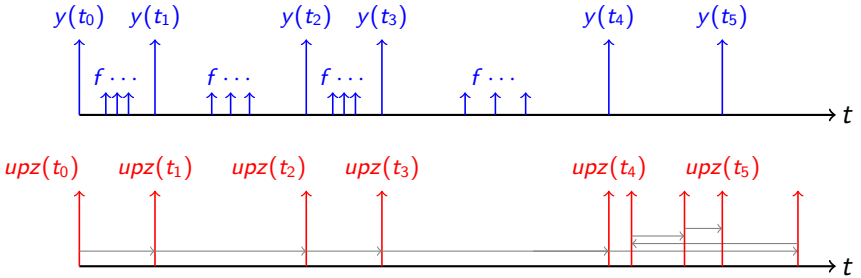
Give solver two functions: $\dot{y} = f_\sigma(t, y)$, $upz = g_\sigma(t, y)$



- Bigger and bigger steps (bound by h_{min} and h_{max})
- t does not necessarily advance monotonically
  - » No side-effects within f or g

Solver execution

Give solver two functions: $\dot{y} = f_\sigma(t, y)$, $upz = g_\sigma(t, y)$



- Bigger and bigger steps (bound by h_{min} and h_{max})
- t does not necessarily advance monotonically
  - » No side-effects within f or g

SUNDIALS
SUite of Nonlinear and DIfferential/ALgebraic Equation Solvers

[Hindmarsh, Brown, Grant, Lee, Serban, Shumaker, and Woodward (2005): SUNDIALS: Suite of Nonlinear and Differential/Algebraic Equation Solvers]

`https://computing.llnl.gov/projects/sundials`

6 solvers

CVODE	$\dot{Y} = f(Y)$ and $Y(t_0) = Y_0$.
CVODES	*CVODE* + sensitivity analysis
IDA	$F(Y, \dot{Y}) = 0,\ Y(t_0) = Y_0,\ \dot{Y}(t_0) = \dot{Y}_0$
IDAS	*IDA* + sensitivity analysis
ARKODE	$M\dot{Y} = f_E(Y) + f_I(Y),\ Y(t_0) = Y_0$
KINSOL	Calculate U from $F(U) = 0$ and initial guess U_0.

Common modules

- vectors of data
- matrices: dense, band, and sparse
- linear solvers
- nonlinear solvers

Written in C with an interface for Fortran

"a general movement away from Fortran and toward C in scientific computing"

Sundials/ML

An OCaml interface to (most of) Sundials

[Bourke, Inoue, and Pouzet (2016): Sundials/ML: interfacing with numerical solvers]

```
opam install sundialsml
```

- Exceptions
- Automatic Memory Management
- Algebraic Data Types
- Type and Module systems
- Dynamic checks
- Mostly follows Sundials

version	*date*	*OCaml*	*C*
first commit	Aug 2010	68	149
v2.5.0p0	Nov 2014	9 147	10 201
v2.6.0p0	Mar 2016	12 888	15 810
v2.7.0p0	Dec 2016	13 477	16 000
v3.1.1p0	Jul 2018	14 350	18 826
v4.1.0p0	Sep 2020	18 565	25 720
v5.8.0p0	Dec 2021	27 074	31 662
v6.0.0p0	Dec 2021	27 896	32 473
v6.1.1p0	Feb 2022	27 896	32 491

Used by https://Zelus.di.ens.fr, OWL (`Owl_ode_sundials`), http://Miking.org

```
1 cvode_mem = CVodeCreate(CV_BDF, CV_NEWTON);
2 if(check_flag((void *)cvode_mem, "CVodeCreate", 0)) return(1);

3 flag = CVodeSetUserData(cvode_mem, data);
4 if(check_flag(&flag, "CVodeSetUserData", 1)) return(1);

5 flag = CVodeInit(cvode_mem, f, T0, u);
6 if(check_flag(&flag, "CVodeInit", 1)) return(1);

7 flag = CVodeSStolerances(cvode_mem, reltol, abstol);
8 if (check_flag(&flag, "CVodeSStolerances", 1)) return(1);

9 LS = SUNSPGMR(u, PREC_LEFT, 0);
10 if(check_flag((void *)LS, "SUNSPGMR", 0)) return(1);

11 flag = CVSpilsSetLinearSolver(cvode_mem, LS);
12 if (check_flag(&flag, "CVSpilsSetLinearSolver", 1)) return 1;

13 flag = CVSpilsSetJacTimes(cvode_mem, jtv);
14 if(check_flag(&flag, "CVSpilsSetJacTimes", 1)) return(1);

15 flag = CVSpilsSetPreconditioner(cvode_mem, Precond, PSolve);
16 if(check_flag(&flag, "CVSpilsSetPreconditioner", 1)) return(1);
```

```
let cvode_mem =
  Cvode.(init BDF
    (Newton Spils.(solver (spgmr u)
                          ~jac_times_vec:(None, jtv data)
                          (prec_left ~setup:(precond data) (psolve data))))
    (SStolerances (reltol, abstol))
    (f data) t0 u)
in
```

Example: double pendulum with *CVODE*

Not logged in Talk Contributions Create account Log in

Article Talk Read Edit View history Search Wikipedia

WIKIPEDIA
The Free Encyclopedia

Main page
Contents
Current events
Random article
About Wikipedia
Contact us
Donate

Contribute
Help
Learn to edit
Community portal
Recent changes
Upload file

Tools
What links here
Related changes
Special pages

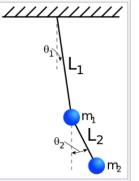
Double pendulum

From Wikipedia, the free encyclopedia

This article includes a list of general references, but **it lacks sufficient corresponding inline citations**. Please help to improve this article by introducing more precise citations. *(June 2013)* *(Learn how and when to remove this template message)*

In physics and mathematics, in the area of dynamical systems, a **double pendulum** is a pendulum with another pendulum attached to its end, is a simple physical system that exhibits rich dynamic behavior with a strong sensitivity to initial conditions.[1] The motion of a double pendulum is governed by a set of coupled ordinary differential equations and is chaotic.

Contents [hide]
1 Analysis and interpretation
1.1 Lagrangian
2 Chaotic motion
3 See also
4 Notes
5 References
6 External links



A double pendulum consists of two pendulums attached end to end.

- No analytical solution exists; i.e., no exact function from t to θ_1 and θ_2.
- No choice but to use numerical simulation.

Example: double pendulum with *CVODE*

In a compound pendulum, the mass is distributed along its length. If the mass is evenly distributed, then the center of mass of each limb is at its midpoint, and the limb has a moment of inertia of $I = \frac{1}{12}ml^2$ about that point.

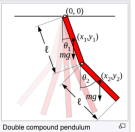
It is convenient to use the angles between each limb and the vertical as the generalized coordinates defining the configuration of the system. These angles are denoted θ_1 and θ_2. The position of the center of mass of each rod may be written in terms of these two coordinates. If the origin of the Cartesian coordinate system is taken to be at the point of suspension of the first pendulum, then the center of mass of this pendulum is at:

$$x_1 = \frac{l}{2}\sin\theta_1$$
$$y_1 = -\frac{l}{2}\cos\theta_1$$

and the center of mass of the second pendulum is at

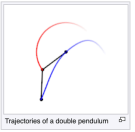
$$x_2 = l\left(\sin\theta_1 + \tfrac{1}{2}\sin\theta_2\right)$$
$$y_2 = -l\left(\cos\theta_1 + \tfrac{1}{2}\cos\theta_2\right)$$

This is enough information to write out the Lagrangian.



Double compound pendulum


Motion of the double compound pendulum (from numerical integration of the equations of motion)


Trajectories of a double pendulum

Lagrangian [edit]

The Lagrangian is

$$\begin{aligned} L &= \text{kinetic energy} - \text{potential energy} \\ &= \tfrac{1}{2}m\left(v_1^2 + v_2^2\right) + \tfrac{1}{2}I\left({\dot\theta_1}^2 + {\dot\theta_2}^2\right) - mg\left(y_1 + y_2\right) \\ &= \tfrac{1}{2}m\left({\dot x_1}^2 + {\dot y_1}^2 + {\dot x_2}^2 + {\dot y_2}^2\right) + \tfrac{1}{2}I\left({\dot\theta_1}^2 + {\dot\theta_2}^2\right) - mg\left(y_1 + y_2\right) \end{aligned}$$

The first term is the *linear* kinetic energy of the center of mass of the bodies and the second term is the *rotational* kinetic energy around the center of mass of each rod. The last term is the potential energy of the bodies in a uniform gravitational field. The dot-notation indicates the time derivative of the variable in question.

Substituting the coordinates above and rearranging the equation gives

$$L = \tfrac{1}{6}ml^2\left({\dot\theta_2}^2 + 4{\dot\theta_1}^2 + 3\dot\theta_1\dot\theta_2\cos(\theta_1 - \theta_2)\right) + \tfrac{1}{2}mgl\left(3\cos\theta_1 + \cos\theta_2\right).$$

There is only one conserved quantity (the energy), and no conserved momenta. The two generalized momenta may be written as

$$p_{\theta_1} = \frac{\partial L}{\partial \dot\theta_1} = \tfrac{1}{6}ml^2\left(8\dot\theta_1 + 3\dot\theta_2\cos(\theta_1 - \theta_2)\right)$$
$$p_{\theta_2} = \frac{\partial L}{\partial \dot\theta_2} = \tfrac{1}{6}ml^2\left(2\dot\theta_2 + 3\dot\theta_1\cos(\theta_1 - \theta_2)\right).$$

These expressions may be inverted to get

$$\dot\theta_1 = \frac{6}{ml^2}\frac{2p_{\theta_1} - 3\cos(\theta_1 - \theta_2)p_{\theta_2}}{16 - 9\cos^2(\theta_1 - \theta_2)}$$
$$\dot\theta_2 = \frac{6}{ml^2}\frac{8p_{\theta_2} - 3\cos(\theta_1 - \theta_2)p_{\theta_1}}{16 - 9\cos^2(\theta_1 - \theta_2)}.$$

The remaining equations of motion are written as

$$\dot p_{\theta_1} = \frac{\partial L}{\partial \theta_1} = -\tfrac{1}{2}ml^2\left(\dot\theta_1\dot\theta_2\sin(\theta_1 - \theta_2) + 3\frac{g}{l}\sin\theta_1\right)$$
$$\dot p_{\theta_2} = \frac{\partial L}{\partial \theta_2} = -\tfrac{1}{2}ml^2\left(-\dot\theta_1\dot\theta_2\sin(\theta_1 - \theta_2) + \frac{g}{l}\sin\theta_2\right).$$

Nvectors

C: Abstract Type

- representation of a vector
- + set of vector functions
- `sundials_nvector.h`

```
struct _generic_N_Vector {
  void *content;
  N_Vector_Ops ops;
};
```

OCaml

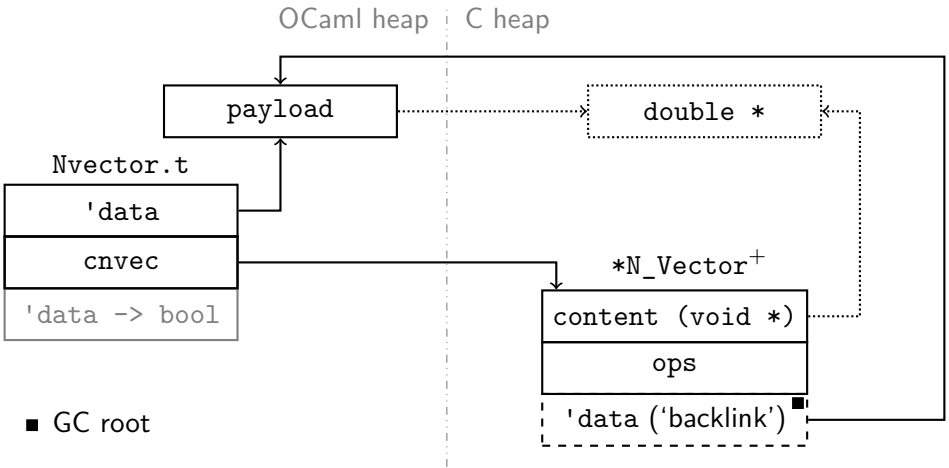
- `type ('data, 'kind) Nvector.t`
- `src/nvectors/nvector.mli`

Implementations

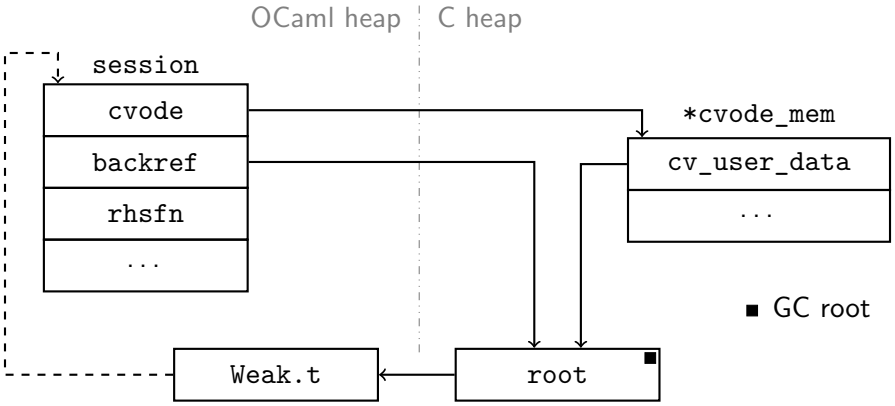
- Serial: array of doubles
- Pthreads: array of doubles
- OpenMP: array of doubles
- Parallel: local array + MPI
- Many: array of nvectors
- Custom: provided by user

```
struct _N_VectorContent_Serial {
  sunindextype length;
  booleantype own_data;
  realtype *data;
};
```

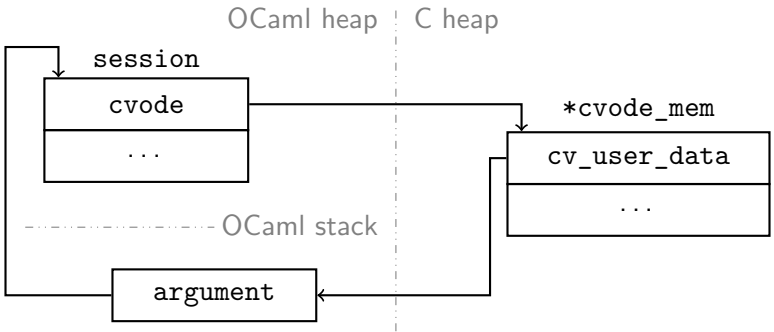
Nvectors: OCaml/C Interface



Sessions (*CVODE*): OCaml/C Interface



Sessions (*CVODE*): Rejected OCaml/C Interface

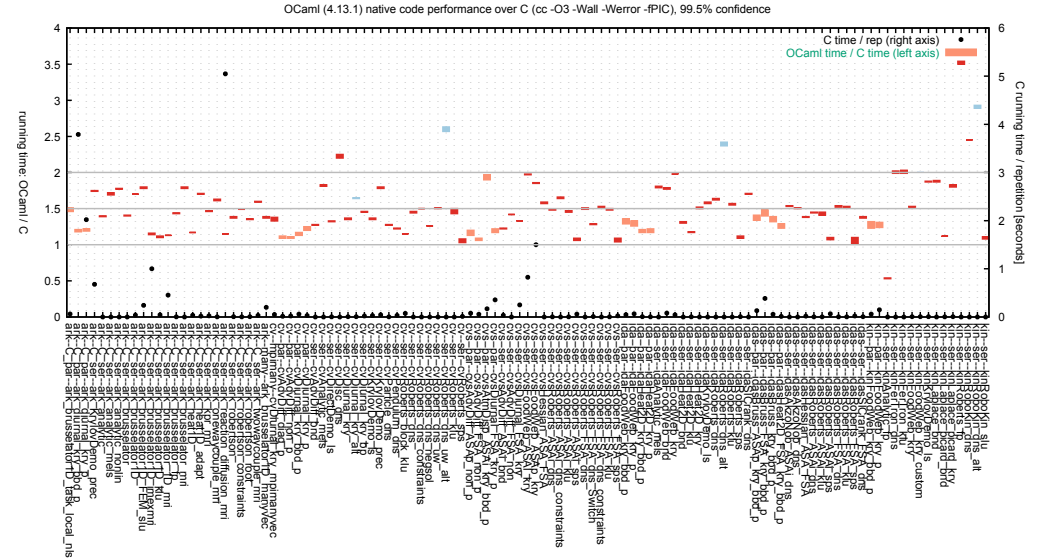


Benchmarking and Debugging

- Reimplement 124 examples from Sundials (34 429 lines of OCaml)
- Helped to improve the OCaml interface

- Run very many times with `Gc.compact`
- Also run with `valgrind`
- Found lots of bugs (and a few in Sundials itself)

- Benchmarking
- `kcachegrind` for diagnostics


OCaml (4.13.1) native code performance over C (cc -O3 -Wall -Werror -fPIC), 99.5% confidence
C time / rep (right axis)
OCaml time / C time (left axis)
running time OCaml / C
C running time / repetition [seconds]
0
0.5
1
1.5
2
2.5
3
3.5
4
5
6

Running faster

1. Explicit type annotations on all vector arguments.

   ```
   let f t y yd = ...
   ```

   becomes

   ```
   type rarray = (float, float64_elt, c_layout) Bigarray.Array1.t
   let f t (y : rarray) (yd : rarray) = ...
   ```

   Not polymorphic; `ocamlopt` optimizes for bigarrays

2. Avoid functions `Bigarray.(Array1.sub` and `Array2.slice_left)`
   - » They allocate fresh big arrays on the major heap
   - » Avoid by passing and manipulating array offsets

3. Write numeric expressions and loops according to [Leroy (2002): Writing efficient numerical code in Objective Caml] to avoid float boxing

Conclusion

- An OCaml interface to (most of) Sundials
- Very convenient for understanding and using the library (types, exceptions, dynamic checks, garbage collection)
- Ideal for mixing symbolic manipulation with numerical simulation

- Improve formatting of online documentation?
- Ongoing work: OCaml 5.x...

References I

- Bourke, T., J. Inoue, and M. Pouzet (Sept. 2016). "Sundials/ML: interfacing with numerical solvers". In: *ACM Workshop on ML*. ACM. Nara, Japan.
- Hindmarsh, A. C., P. N. Brown, K. E. Grant, S. L. Lee, R. Serban, D. E. Shumaker, and C. S. Woodward (Sept. 2005). "SUNDIALS: Suite of Nonlinear and Differential/Algebraic Equation Solvers". In: *ACM Trans. Mathematical Software* 31.3, pp. 363–396.
- Leroy, X. (July 2002). *Writing efficient numerical code in Objective Caml*. `http://caml.inria.fr/pub/old_caml_site/ocaml/numerical.html`.