Assimulo - a Python package for solving differential equations with interface to equation based languages

Christian Andersson, Claus Führer

Johan Åkesson

LCCC workshop Lund
September 2012

Let's move the focus ...

Problem

Solver

User friendly description

$$\dot{x} = f(t, x)$$

$$F(t, x, \dot{x}) = 0$$

$$x(0) = x_0$$
$$\dot{x}(0) = \dot{x}_0$$

$$t \in [t_0, t_\mathrm{e}]$$

$$\sum \alpha_i x_{n-1} - h \sum \beta_i \dot{x}_{n-1} = 0$$

Mathematical description

ODE and DAE solvers in two disjoint worlds

Industrial Simulation Tasks

- highly complex models
- high robustness standards
- high documentation standards
- long life cycle

$\longrightarrow$ *one or two ODE/DAE packages meet these requirements.*

Academic Simulation Tasks

- a few, low scale test models
- lab standard quality (validation of concept)
- good analyzed algorithms, poor code documentation
- short life cycle, often coupled on individual career steps.

$\longrightarrow$ *dozen of codes produced (and forgotten) this way.*

ODE and DAE solvers in two disjoint worlds

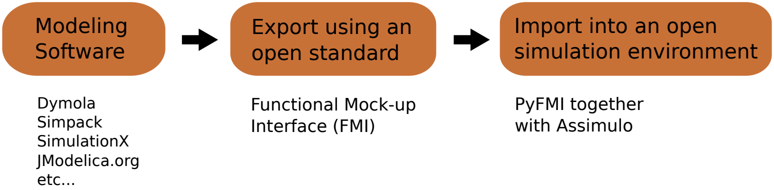
The harsh requirement that a useful numerical method must permit an efficient, robust, and reliable implementation has withered the beautiful flowers which bloomed on thousands of journal pages.

Hans Stetter in:
Mathematics of computation, 1943-1993: a half-century of computational mathematics : Mathematics of Computation 50th Anniversary Symposium, August 9-13, 1993, Vancouver, British Columbia

... highly valid still today.

Motivation

- Give the academic world access to complex models $\longrightarrow$ FMI
- Give the industrial world access to a variety of ODE/DAE codes (even experimental ones): $\longrightarrow$ ASSIMULO
- Give students in scientific computing an intuitive access to industrial standard solvers: $\longrightarrow$ ASSIMULO



Functional Mock-up Interface (FMI)

FMI is an open interface for model exchange with the idea that tools may generate and exchange dynamic system models.

The **FMI** supports model defined as discontinuous ordinary differential equations.

- **Model interface** The equations are evaluated and the model interaction is performed by standardized *C* functions.
- **Model description** The variable information of the model is contained in an XML-file.
- **Additional data** Model data, such as tables and maps may also exists.

$\Rightarrow$ Talk by Torsten Blochwitz on Wednesday.

Assimulo is written in Python, why?

Benefits of using Python:

- Open-source language
- Interpreted
- Object-oriented
- Many freely available packages
  - NumPy
  - SciPy
  - Matplotlib
  - Cython
- Highly flexible for interfacing to *C*, FORTRAN ...
- Ideal in teaching.

Assimulo

Python workbench for simulation of ordinary differential equations.

The intention is to provide a common high-level interface for a variety of different solvers.

Supports

- problems formulated as first or second order ordinary differential equations
- problems formulated as implicit ordinary differential equations including overdetermined problems.

ASSIMULO, problem formulations

- Explicit hybrid ODEs

$$\dot{y} = f(t, y, sw), \quad y(t_0) = y_0, \quad sw(t_0) = sw_0$$

- Implicit hybrid ODEs (also called DAEs)

$$F(t, y, \dot{y}, sw) = 0, \quad y(t_0) = y_0, \quad \dot{y}(t_0) = \dot{y}_0, \quad sw(t_0) = sw_0$$

- Mechanical systems in second order explicit ODE form

$$\ddot{p} = M(p)^{-1} f(t, p, \dot{p})$$

- Mechanical systems in (overdetermined) implicit ODE form

$$\begin{aligned} \dot{p} &= v \\ M(p)\dot{v} &= f(t, p, v) - G^T(p)\lambda \\ 0 &= g_{\mathrm{constr}}(p) \\ 0 &= G(p)v \end{aligned}$$

- Delay (retarded) differential equations.

ASSIMULO, solvers

Currently, solvers written in Python, FORTRAN and *C* are available.

- **IDA** - Multistep method for DAEs
- **CVode** - Multistep methods for ODEs
- **ODASSL** - Multistep methods for overdetermined DAEs
- **RADAU5** - Runge–Kutta method for DAEs
- **GLIMDA** - General linear methods methods for DAEs
- and we are working on a **"solver museum"** (oldest code in restoration 1983).

IDA and CVode are production quality solvers from the **SUNDIALS** suite.

ASSIMULO, overview

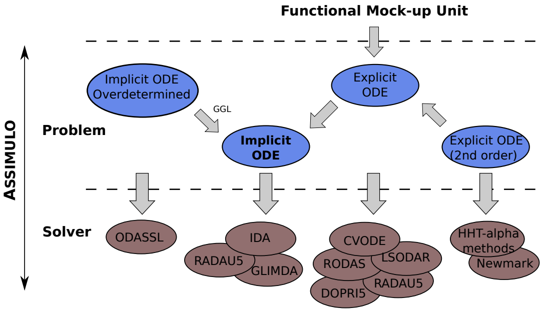


Figure : Connection between the different problem formulations and the different solvers available in ASSIMULO. The connection of the Functional Mock-up Interface to ASSIMULO is also shown.

Simple example workflow

Make a problem

```
def rhs(t,y):
  A = array([[0, 1], [-2, -1]])
  yd = N.dot(A, y)
  return yd

y0 = array([1.0, 1.0])
t0 = 0.0

linmodel = Explicit_Problem(rhs, y0, t0)
```

Create a solver instance

```
sim = CVode(linmodel)
```

... and simulate

```
t, y = sim.simulate(tfinal)
```

Assimulo can be quite verbose...

```
Final Run Statistics: Linear Test ODE

 Number of Error Test Failures              = 4
 Number of F-Eval During Jac-Eval           = 0
 Number of Function Evaluations             = 153
 Number of Jacobian Evaluations             = 0
 Number of Nonlinear Convergence Failures   = 0
 Number of Nonlinear Iterations             = 149
 Number of Root Evaluations                 = 0
 Number of Steps                            = 84

Solver options:

 Solver                    :  CVode
 Linear Multistep Method   :  Adams
 Nonlinear Solver          :  FixedPoint
 Maxord                    :  12
 Tolerances (absolute)     :  1e-06
```

Controlling the method

```
sim.atol=N.array([1.0,0.1])*1.e-5
sim.rtol=1.e-8
sim.maxord=3
sim.discr='BDF'
sim.iter='Newton'
```

Discontinuities – a Continuous Challenge

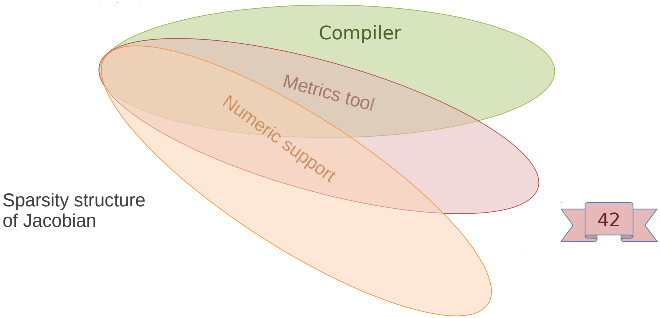
```
class Extended_Problem(Explicit_Problem):
  #Sets the initial conditions directly into the problem
  y0 = [0.0, -1.0, 0.0]
  sw0 = [False,True,True]

  #The right-hand-side function (rhs)
  def rhs(self, t, y, sw):
      ....

  #The event function
  def state_events(self, t, y, sw):
     event_0 = y[1] - 1.0
      ...
     return array([event_0,event_1,event_2])

  #Responsible for handling the events.
  def handle_event(self, solver, event_info):
    event_info = event_info[0]
    while True: #Event Iteration
      self.event_switch(solver, event_info) #Turns the swi
       ...
```

Languages have the potential to inform

- Are there discontinuities?
- State/Time events?
- Are there linear components?
- What are differential, what are algebraic variables? ("loop closure" conditions versus algebraic equations)
- Derivatives?

The compiler might know more

Why extensible compilers?



(Sorry Görel for changing your slide ...)

Plans/ideas/wishes for the future

- Would like to stimulate to open the **FMI** for a wider range of problem formulations - higher index DAES(?)
- Continue to expand the solvers available in ASSIMULO
  - Work on the museum.
  - Introduce problem formulation for delay differential equations
  - Generalize solvers for discontinuity handling
- Potentials of language/compiler aided numerics.
- Automatic differentiation: a separate tool or an integrated part of the language-solver chain?

Thank you!

... and feel free to try it out!

- **Assimulo** www.assimulo.org
- **PyFMI** www.pyfmi.org