Symbolic Jacobian of ODE: An overlooked tool to improve simulation speed and accuracy

Cyril Garneau and Peter A. Vanrolleghem

Stats and Control Meeting
Québec 2017

Outline

- Context
- The Jacobian
- The Symbolic Jacobian
- Test cases
- Results
- Discussion
- Conclusion

Context – Modelling Water Resource Recovery Facilities

- WRRF are traditionally modelled as a set of Ordinary Differential Equations (ODE) expressing mass-balance and reactions processes.

State variable

$$\boxed{\frac{dM}{dt}} = \boxed{M_{in} - M_{out}} + \boxed{R}$$

Reaction term

Mass variation

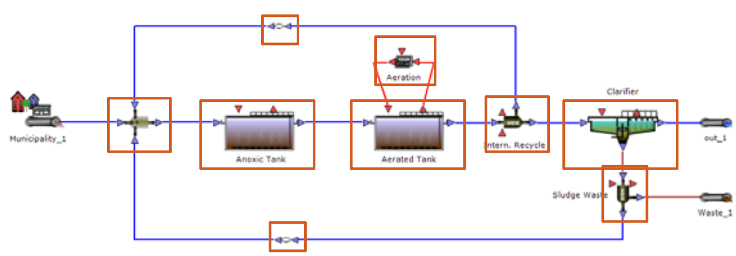
Input - outputs

- Inputs and outputs allow to describe the hydraulics through tank-in-series models.
- Reactions refer to physical (i.e. sedimentation), chemical (i.e. precipitation) or biological (i.e. biomass growth) processes.
  - In WRRF models: Activated Sludge Model (ASM).

Context – Modelling Water Resource Recovery Facilities

- A simple WRRF model:

State variables (ASM0):
- Water
- Biomass
- Substrate
- Oxygen



State variables (settler):
- Total suspended solids in each vertical layer

Continuity → ; Mass Balance ↓

Component → i / j Process ↓	1 X_B	2 S_S	3 S_O	Process Rate, ρ_j $[ML^{-3}T^{-1}]$
1 Growth	1	$-\frac{1}{Y}$	$-\frac{1-Y}{Y}$	$\frac{\hat{\mu} S_S}{K_S + S_S} X_B$
2 Decay	−1		−1	bX_B
Observed Conversion Rates $ML^{-3}T^{-1}$	$r_i = \sum_j r_{ij} = \sum_j \nu_{ij} \rho_j$			Kinetic Parameters: Maximum specific growth rate: $\hat{\mu}$ Half-velocity constant: K_S Specific decay rate: b
Stoichiometric Parameters: True growth yield: Y	Biomass [M(COD) L^{-3}]	Substrate [M(COD) L^{-3}]	Oxygen (negative COD) [M(−COD) L^{-3}]	

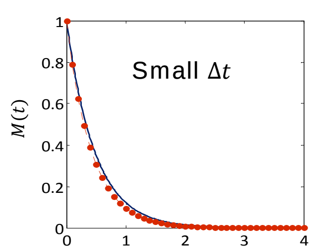
Context – Solving WRRF models

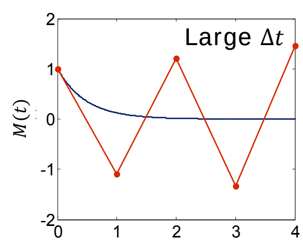
- Ordinary Differential Equations (ODE) describing WRRF model:
  - Large diversity of the dynamics of the state variables
    - Oxygen is consumed in minutes
    - Biomass takes weeks to grow
  - The model is stiff and highly non-linear
  - Control strategies can involve discrete events (discontinuities).
- ODE solvers range from very simple to very complex:
  - Explicit Euler method
  - Runge-Kutta 4
  - Implicit Euler
  - Matlab ODE suite (ode45, ode23, ode15s, etc…)
  - Adams-Moulton / Adams-Bashford (CVODE)
  - Diagonally Implicit Runge-Kutta method (DIRK)
  - Etc…

Context – Euler solver

- The simplest ODE solver is the Euler method:
  For $\frac{dM}{dt} = f(M)$ and $M(t_0) = M_0$:

$$M_1 = M_0 + f(M_0) \times \Delta t$$





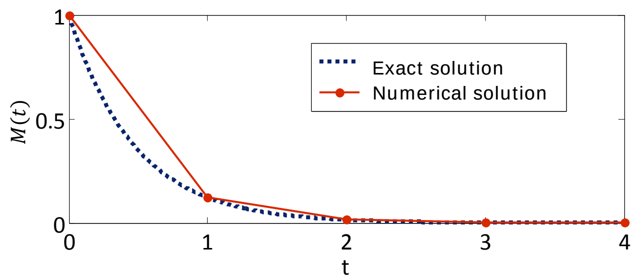
- If $f(M)$ is stiff and Δt is large, instability ruins the solution, unless we solve:

$$M_1 = M_0 + f(M_1) \times \Delta t$$
$$0 = M_0 - M_1 + f(M_1) \times \Delta t$$

Context – Euler solver

- The simplest ODE solver is the Euler method:
  For $\frac{dM}{dt} = f(M)$ and $M(t_0) = M_0$:

$$M_1 = M_0 + f(M_0) \times \Delta t$$



- If $f(M)$ is stiff and Δt is large, instability ruins the solution, unless we solve:

$$M_1 = M_0 + f(M_1) \times \Delta t$$
$$0 = M_0 - M_1 + f(M_1) \times \Delta t$$

Context – Solving WRRF models

- Since it is not possible to solve directly

$$0 = M_0 - M_1 + f(M_1) \times \Delta t$$

The Jacobian matrix

$$J(M) = \begin{bmatrix} \frac{\partial f_1}{\partial m_1} & \cdots & \frac{\partial f_1}{\partial m_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial m_1} & \cdots & \frac{\partial f_n}{\partial m_n} \end{bmatrix}$$

provides a linear approximation of the function

$$f(M_1) \cong f(M_0) + J(M_0) \times (M_1 - M_0)$$

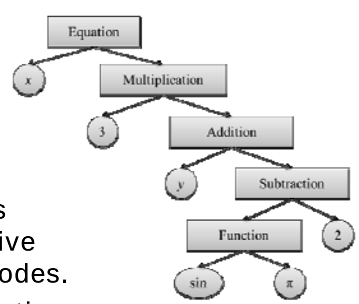
The Jacobian

- How to estimate the Jacobian?
  - Finite differences: $f'(M) \cong \frac{f(M+\Delta M)-f(M)}{\Delta M}$
    Requires $n+1$ model evaluations. Subject to round-off error.
  - Automatic Differentiation (AD): Numerical evaluation of the Jacobian through specialized libraries.
    Requires in-depth dependency of the model to additional code.
  - Matrix-free techniques (i.e. Krylov subspace): Efficient on very large models, but less stable and biased solution.
  - Symbolic derivation: Exact derivative expression computed before the execution of the model.

Symbolic derivation of large set of equations

- WEST, the WRRF model simulator, generates various representations of a model:
  - Object-oriented Modelica
  - Flat Modelica
  - Abstract Syntax Tree (AST)
  - Plain and compiled C-code

$$x = 3 * y + (sin\,\pi\ - 2)$$



- The AST structure of an equation is easily derived with a simple recursive derivation function applied on all nodes.
- Chain derivation then allows generating an arbitrarily complex Jacobian

Symbolic derivation of large set of equations

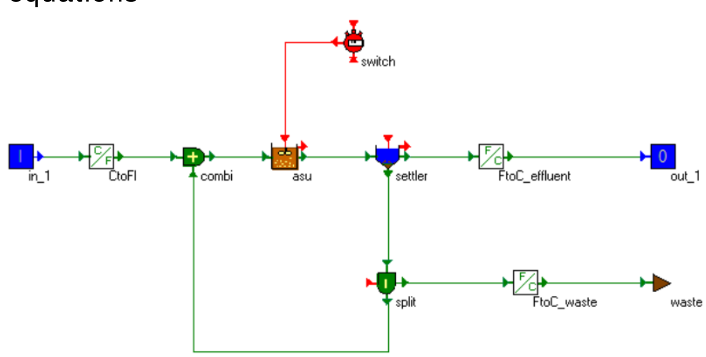
- Non-analytical constructs must be derived as well.
  - IF-Test, Min, Max, Abs, etc.
    Example of IF-Test:

$$\frac{d}{du}(if(COND)\ then\ A\ else\ B) = if(COND)\ then \frac{dA}{du}\ \ else \frac{dB}{du}$$

- Algorithmic constructs or external computations are managed through Finite Differences (**work in progress**).
  - Ex: PHREEQC = m chemical species and n chemical compounds. Solution computed through complex mathematical algorithms.

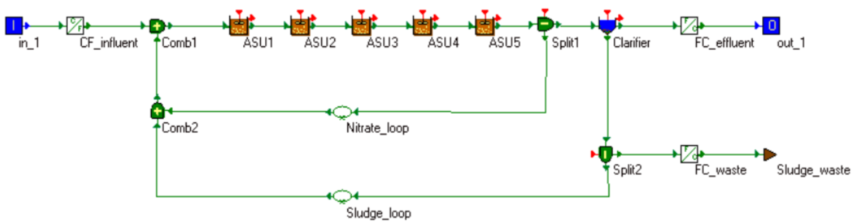
Test case no. 1 for symbolic derivation

- ASM2d_ASU: simplest layout of one activated sludge unit and one settler
- 30 state variables
  - Jacobian = 30 x 30 matrix = 900 partial derivatives
- 418 equations



Test case no. 2 for symbolic derivation

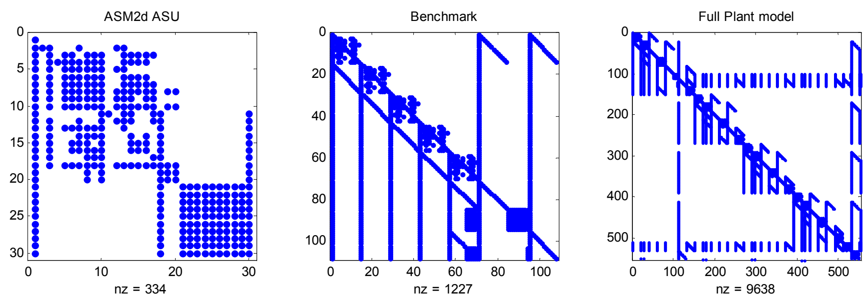
- Benchmark Simulation Model No. 1 (BSM1)
- Simple WRRF plant layout of 5 ASU and one settler used in many articles to test control strategies
- 108 state variables
  - Jacobian = 11664 partial derivatives
- 946 intermediate equations



Test case No. 3 for symbolic derivation

- Full scale WRRF model
- 17 ASU and 2 settlers
- 554 state variables
  - Jacobian = 306 916 partial derivatives
- 5694 intermediate equations

Results



FR = 0.37 | FR = 0.11 | FR = 0.03

- Jacobians are sparse matrices -> Use sparse matrix tools!
- The Filling Ratio (FR, ratio of non-zero elements versus total number of elements) decreases as model complexity increases.
- The structure of the WRRF model is apparent (ASU, settlers, etc.)

Results

- Investment versus Reward: Comparison of Jacobian calculation: Symbolic derivation (SD) vs Finite difference (FD)

	ASM2d_ASU 30 state var	Benchmark 108 state var	Full plant 554 state var
Time to generate and compile the Symbolic Jacobian	16 s	98 s	505 s*
Speedup of Jacobian calculation	12	23	28
Speedup of simulation time (Diagonally Implicit Runge-Kutta method)	1.5	4.7	40**

* Compilation was done without optimisation (insufficient memory)
** 80% of the speedup was attributable to sparse matrix operations.

Discussion

- Developing a symbolic Jacobian provided a deep insight in the matrix structure
  - Sparse matrix tools were overlooked and provide an easy way to speedup simulations without affecting accuracy.
  - The structure of the WRRF model can be recovered from the Jacobian structure -> Automatic model analysis possible
- Improved numerical performance was demonstrated
  - Investing in a symbolic Jacobian pays back in 1 to 10 simulations
    - New virtual experiments may need 100s to 1000s of simulations (e.g. sensitivity analysis, Monte Carlo experiment, etc.)!
  - Round-off free Jacobian: New solution options for challenging ODE (i.e. chemical speciation models)

Conclusion

- Symbolic manipulations allow faster, more stable and more precise computations than traditional finite differences.
- Large symbolic Jacobian computation is not trivial, but possible thanks to the available computer power.
- Non-differentiable functions and algorithms can still be evaluated numerically (finite differences), but their integration to a generic framework is challenging.
- Symbolic Jacobian offered optimal use of sparse matrix tools.
- A reliable and inexpensive Jacobian provides a useful approximation of a complex model.