

DAE TOOLS SOFTWARE

INTRODUCTION

D.D. Nikolić

Updated: 20 March 2018

DAE Tools Project, <http://www.daetools.com>







1. General Information
2. Motivation
3. Main Features
4. Programming Paradigms
5. Use Cases

GENERAL INFORMATION

What is DAE Tools?

MODELLING, SIMULATION, OPTIMISATION & PARAMETER ESTIMATION software ¹

- Areas of application:
 - Initially: **CHEMICAL PROCESS INDUSTRY** (mass, heat and momentum transfers, chemical reactions, separation processes, thermodynamics, electro-chemistry)
 - Nowadays: **MULTI-DOMAIN**
- **FREE/OPEN SOURCE SOFTWARE** (GNU GPL) 
- **CROSS-PLATFORM**   
- **MULTIPLE ARCHITECTURES** (32/64 bit x86, ARM, ...)

¹Nikolić DD. (2016) *DAE Tools: equation-based object-oriented modelling, simulation and optimisation software*. PeerJ Computer Science 2:e54

What is DAE Tools? (cont'd)

- DAE Tools **IS NOT**:
 - A modelling language (such as Modelica)
 - An integrated software suite of data structures and routines for scientific applications (such as PETSc, Sundials, ...)
- DAE Tools **IS**:
 - An **ARCHITECTURAL DESIGN OF INTERDEPENDENT SOFTWARE COMPONENTS** providing an API for:
 - **MODEL SPECIFICATION**
 - Activities on developed models (**SIMULATION**, **OPTIMISATION**, ...)
 - **PROCESSING OF THE RESULTS**
 - **REPORT GENERATION**
 - **CODE GENERATION** and **MODEL EXCHANGE**
- DAE Tools apply a **HYBRID APPROACH** between **MODELLING** and **GENERAL PURPOSE** programming languages, **COMBINING THE STRENGTHS OF BOTH APPROACHES** into a single one

What can be done with DAE Tools?

- **SIMULATION**
 - Steady-State
 - Transient
- **SENSITIVITY ANALYSIS**
 - Local methods (derivative-based)
 - Global methods (Morris, FAST, Sobol variance-based)
- **OPTIMISATION**
 - Non-Linear Programming (NLP)
 - Mixed Integer Non-Linear Programming (MINLP)
- **PARAMETER ESTIMATION**
- **CODE-GENERATION, MODEL-EXCHANGE, CO-SIMULATION**
 - Modelica, gPROMS, Functional Mockup Interface (FMI)
 - Matlab MEX-functions, Simulink user-defined S-functions
 - C99, C++ MPI (embedded and distributed systems)

Types of systems that can be modelled

INITIAL VALUE PROBLEMS OF IMPLICIT FORM:

- Described by **SYSTEMS OF LINEAR, NON-LINEAR, AND (PARTIAL-)DIFFERENTIAL** algebraic equations
- **CONTINUOUS** with some elements of **EVENT-DRIVEN** systems (discontinuous equations, state transition networks and discrete events)
- **STEADY-STATE** or **DYNAMIC**
- With **LUMPED** or **DISTRIBUTED** parameters (finite difference, finite volume and finite element methods)
- Only **INDEX-1** DAE systems at the moment

MOTIVATION



Why modelling software?

In general, two scenarios:

- **DEVELOPMENT** of a **NEW** product/process/...
 - Reduce the time to market (TTM)
 - Reduce the development costs (no physical prototypes)
 - Maximise the performance, yield, productivity, purity, ...
 - Minimise the capital and operating costs
 - Explore the new design options in less time and no risks
- **OPTIMISATION** of an **EXISTING** product/process/...
 - Increase the performance, yield, productivity, purity, ...
 - Reduce the operating costs, energy consumption, ...
 - Debottleneck

Why YET ANOTHER modelling software?

Current approaches to mathematical modelling:

1. Use of **MODELLING LANGUAGES** (domain-specific or multi-domain): **MODELICA**, **ASCEND**, **gPROMS**, **DYMOLA**, **APMonitor**
2. Use of **GENERAL-PURPOSE PROGRAMMING LANGUAGES**:
 - Lower level third-generation languages such as C, C++ and Fortran (**PETSc**, **SUNDIALS**)
 - Higher level fourth-generation languages such as **PYTHON** (NumPy, SciPy, Assimulo), **JULIA** etc.
 - Multi-paradigm numerical languages (**MATLAB**, **MATHEMATICA**, **MAPLE**, **SCILAB**, and **GNU OCTAVE**)

Why YET ANOTHER modelling software? (cont'd)

The advantages of the **HYBRID** approach over the **MODELLING** and **GENERAL-PURPOSE** programming languages:

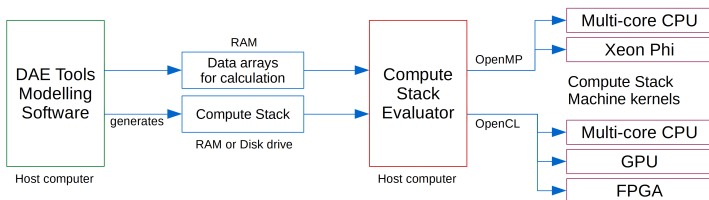
1. Support for the **RUNTIME MODEL GENERATION**
2. Support for the **RUNTIME SIMULATION SET-UP**
3. Support for **COMPLEX SCHEDULES** (operating procedures)
4. **INTEROPERABILITY** with the **THIRD-PARTY SOFTWARE**
5. Suitability for **EMBEDDING** and use as a **WEB APPLICATION** or **SOFTWARE AS A SERVICE**
6. **CODE-GENERATION**, **MODEL EXCHANGE** and **CO-SIMULATION** capabilities

MAIN FEATURES

Parallel Computing

The **SHARED-MEMORY** parallel programming model supported.

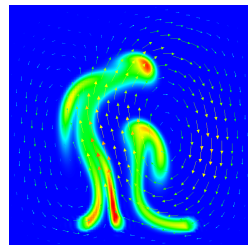
- **EVALUATION OF EQUATIONS AND DERIVATIVES**
 - **OPENMP**: general purpose processors and manycore devices
 - **OPENCL**: streaming processors (GPU, GPGA) and heterogeneous systems
- **ASSEMBLY OF FINITE ELEMENT SYSTEMS** (OpenMP)
- **SOLUTION OF SYSTEMS OF LINEAR EQUATIONS** (SuperLU_MT, Pardiso and Intel Pardiso solvers)
- **GLOBAL SENSITIVITY ANALYSIS** (multiprocessing.Pool)



Multiphysics Capabilities

Modelling of multiple simultaneous physical phenomena

- **FINITE DIFFERENCE** (FD), **FINITE VOLUME** (FV) and **FINITE ELEMENT** (FE) methods
- Mixed coupled systems of equations (FD, FV and FE methods)
- DAE Tools variables for boundary conditions, source terms and non-linear coefficients
- Additional constraints and auxiliary equations



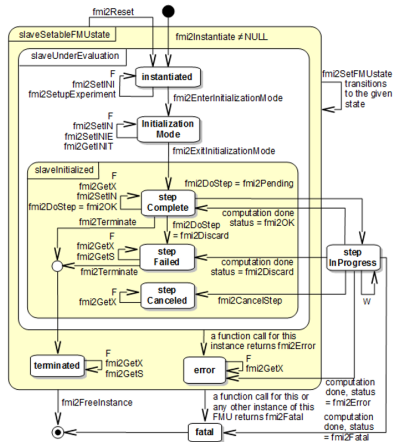
Code Generation & Co-Simulation

○ CODE-GENERATION

- Modelica
- gPROMS
- C99 (embedded systems)
- C++ MPI (distributed systems)

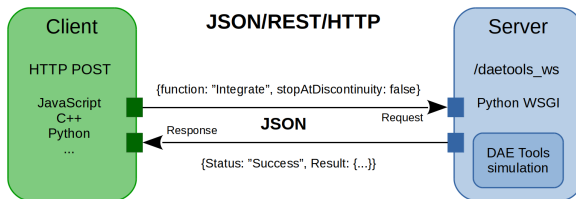
○ CO-SIMULATION

- Matlab MEX-functions
- Simulink user-defined S-functions
- Functional Mockup Interface (FMI) for Co-Simulation



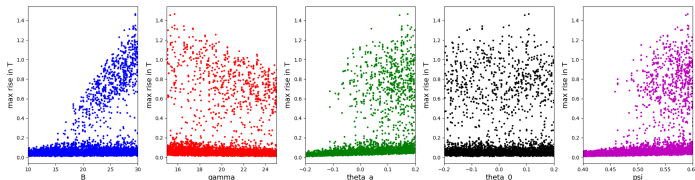
Software As a Service

- **WEB SERVICE WITH THE RESTFUL API**
 - DAE Tools simulations (daetools_ws)
 - DAE Tools FMU objects (daetools_fmi_ws)
- **LANGUAGE INDEPENDENT** (JavaScript, Python, C++, ...)
- **BENEFITS:**
 - Application servers
 - Individual simulations as a web service
 - Attractive Graphical User Interface



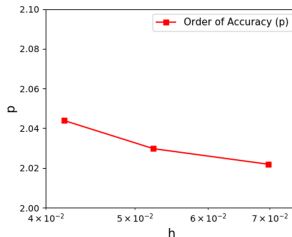
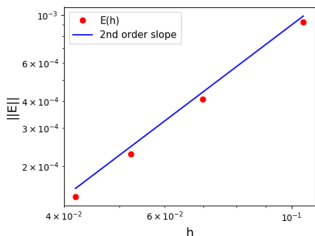
Sensitivity Analysis

- **LOCAL SENSITIVITY ANALYSIS** (derivative-based)
- **GLOBAL SENSITIVITY ANALYSIS** (SALib library):
 - 1st and 2nd order sensitivities and confidence intervals
 - Total sensitivity indices and confidence intervals
 - Scatter plots
- Methods available:
 - **METHOD OF MORRIS** (elementary effect method)
 - **FAST** (variance-based)
 - **SOBOL** (variance-based)
- Simulations performed in parallel (multiprocessing.Pool)



Code Verification

- The **FORMAL CODE VERIFICATION TECHNIQUES** applied to test almost all aspects of the software
- The **MOST RIGOROUS CODE VERIFICATION METHODS** used:
 - The **METHOD OF EXACT SOLUTIONS** (MES)
 - The **METHOD OF MANUFACTURED SOLUTIONS** (MMS)
- The **MOST RIGOROUS ACCEPTANCE CRITERIA** used:
 - Percent error
 - Normalised global error
 - **ORDER-OF-ACCURACY**



Additional DAE TOOLS features

- Support for **MULTIPLE PLATFORMS / ARCHITECTURES**
- Support for the **AUTOMATIC DIFFERENTIATION** (ADOL-C)
- Support for a large number of **DAE**, **LA** and **NLP** solvers
- Support for the generation of **MODEL REPORTS** (XML + MathML, Latex)
- **EXPORT** of the **SIMULATION RESULTS** to various file formats (Matlab, Excel, json, xml, HDF5, Pandas, VTK)

PROGRAMMING PARADIGMS

The HYBRID approach

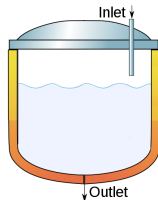
- DAE Tools approach is a **TYPE OF A HYBRID APPROACH**
- Combines strengths of **MODELLING** and **GENERAL PURPOSE** programming languages:
 1. **DEVELOPED IN C++** with the **PYTHON BINDINGS**
 2. Provides **API** (Application Programming Interface) that **RESEMBLES A SYNTAX OF MODELLING LANGUAGES** as much as possible
 3. **TAKES ADVANTAGE OF THE HIGHER LEVEL LANGUAGES** for:
 - Model specification, simulation setup and schedules
 - Access to the operating system
 - Access to the standard/third-party libraries

The HYBRID approach (cont'd)

- Modelica/gPROMS grammars vs. DAE Tools API

- A simple model:

Cylindrical tank containing a liquid with an inlet and an outlet flow; the outlet flowrate depends on the liquid level in the tank



```
PARAMETER
  Density as Real
  CrossSectionalArea as Real
  Alpha as Real

VARIABLE
  HoldUp as Mass
  FlowIn as Flowrate
  FlowOut as Flowrate
  Height as Length

EQUATION
  # Mass balance
  $HoldUp = FlowIn - FlowOut;

  # Relation between liquid level and holdup
  HoldUp = CrossSectionalArea * Height * Density;

  # Relation between pressure drop and flow
  FlowOut = Alpha * sqrt(Height);
```

gPROMS grammar

```
model BufferTank
  /* Import libs */
  import Modelica.Math.*;

  parameter Real Density;
  parameter Real CrossSectionalArea;
  parameter Real Alpha;

  Real HoldUp(start = 0.0);
  Real FlowIn;
  Real FlowOut;
  Real Height;

equation
  // Mass balance
  der(HoldUp) = FlowIn - FlowOut;

  // Relation between liquid level and holdup
  HoldUp = CrossSectionalArea * Height * Density;

  // Relation between pressure drop and flow
  FlowOut = Alpha * sqrt(Height);

end BufferTank;
```

Modelica grammar

The HYBRID approach (cont'd)

```
class BufferTank(daeModel):
    def __init__(self, Name, Parent = None, Description = ""):
        daeModel.__init__(self, Name, Parent, Description)

        self.Density = daeParameter("Density", unit(), self)
        self.CrossSectionalArea = daeParameter("CrossSectionalArea", unit(), self)
        self.Alpha = daeParameter("Alpha", unit(), self)

        self.HoldUp = daeVariable("HoldUp", no_t, self)
        self.FlowIn = daeVariable("FlowIn", no_t, self)
        self.FlowOut = daeVariable("FlowOut", no_t, self)
        self.Height = daeVariable("Height", no_t, self)

    def DeclareEquations(self):
        # Mass balance
        eq = self.CreateEquation("MassBalance")
        eq.Residual = self.HoldUp.dt() - self.FlowIn() + self.FlowOut()

        # Relation between liquid level and holdup
        eq = self.CreateEquation("LiquidLevelHoldup")
        eq.Residual = self.HoldUp() - self.CrossSectionalArea() * self.Height() * self.Density()

        # Relation between pressure drop and flow
        eq = self.CreateEquation("PressureDropFlow")
        eq.Residual = self.FlowOut() - self.Alpha() * Sqrt(self.Height())
```

DAE Tools API

The HYBRID approach (cont'd)

Modelling language approach	DAE Tools approach
Solutions expressed in the idiom and at the level of abstraction of the problem domain	Must be emulated in the API or in some other way
Clean and concise way of building models	Verbose and less elegant
Could be and often are simulator independent	Simulator dependent (but with code-generation)
Cost of designing, implementing, and maintaining a language and a compiler/lexical parser/interpreter, error handling and grammar ambiguities	A compiler/lexical parser/interpreter is an integral part of C++/Python with a robust error handling, universal grammar and massively tested
Cost of learning a new language vs. its limited applicability (yet another language grammar)	No learning of a new language required
Difficult to integrate with other components	Calling external libraries is a built-in feature
Models usually cannot be created/modified in the runtime (or at least not easily)	Models can be created/modified in the runtime
Setting up a simulation embedded in the language; difficult to obtain initial values from other software	Setting up a simulation done programmatically and the initial values can be obtained from other software
Schedules limited to the options allowed by the language grammar	Schedules completely flexible (within the limits of a programming language itself)

The OBJECT-ORIENTED approach

- Everything is an **OBJECT** (variables, equations, models ...)
- All objects can be **MANIPULATED** in **THE RUNTIME**
- **ALL** C++/Python **OBJECT-ORIENTED CONCEPTS SUPPORTED**
- Models, simulations, optimisations:
 - **DERIVED FROM** the corresponding **BASE CLASSES**
 - **INHERIT** the **COMMON FUNCTIONALITY** from the base classes
 - Perform the **FUNCTIONALITY** in **OVERLOADED FUNCTIONS**
- The **HIERARCHICAL MODEL DECOMPOSITION** possible:
 - Models can contain instances of other models
 - Complex, re-usable model definitions can be created
 - Models at different scales can be loosely coupled

The EQUATION-ORIENTED (ACAUSAL) approach

- EQUATIONS GIVEN IN AN IMPLICIT FORM (as a residual)

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

- INPUT-OUTPUT CAUSALITY is NOT FIXED:

- Increased model re-use
- Support for DIFFERENT SIMULATION SCENARIOS (based on a single model) by specifying different degrees of freedom

- An example:

- The equation given in the following form:

$$x_1 + x_2 + x_3 = 0$$

- Can be used to determine either x_1 , x_2 or x_3 depending on what combination of variables is known:

$$x_1 = -x_2 - x_3, \text{ OR } x_2 = -x_1 - x_3, \text{ OR } x_3 = -x_1 - x_2$$

Separation of MODEL DEFINITION from its APPLICATIONS

- MODEL STRUCTURE specified in the MODEL CLASS
- RUNTIME INFORMATION specified in the SIMULATION CLASS
- SOLVERS/AUXILIARY OBJECTS declared in the MAIN PROGRAM
- SINGLE MODEL DEFINITION, but ONE OR MORE:
 - Different SIMULATION SCENARIOS
 - Different OPTIMISATION SCENARIOS

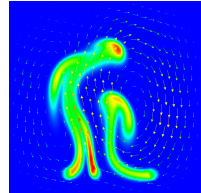
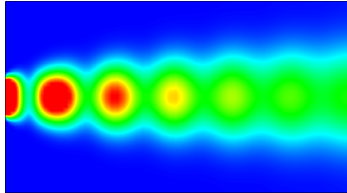
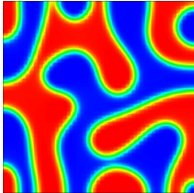
USE CASES

Use Case 1 - Chemical Engineering

- CONTINUOUSLY STIRRED TANK REACTOR (Van de Vusse) ↗ ↗
- PLUG FLOW REACTOR ↗
- DISTILLATION COLUMN ↗ ↗
- BATCH CRYSTALLISER ↗ ↗
- DISCRETISED POPULATION BALANCE EQUATIONS ↗ ↗ ↗
- NEWMAN POROUS ELECTRODE THEORY (PET) ↗ ↗ ↗
- MULTIPHASE POROUS ELECTRODE THEORY (MPET) ↗ ↗
- HYDROXIDE EXCHANGE MEMBRANE FUEL CELLS (HEMFCs) ↗
- MAXWELL-STEFAN EQUATIONS (porous membranes) ↗ ↗
- PRESSURE SWING ADSORPTION ↗ ↗

Use Case 2 - Finite Element Method

- TRANSIENT HEAT CONDUCTION/CONVECTION ↗
- CAHN-HILLIARD EQUATION ↗
- FLOW THROUGH THE POROUS MEDIA ↗
- DIFFUSION/REACTION IN AN IRREGULAR CATALYST SHAPE ↗
- STOKES FLOW DRIVEN BY THE DIFFERENCES IN BUOYANCY ↗



Use Case 3 - Parameter Estimation & Optimisation

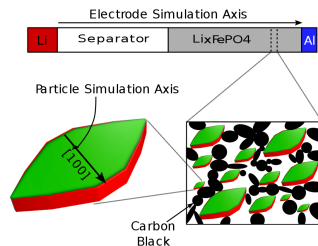
LARGE-SCALE CONSTRAINED OPTIMISATION PROBLEM SET (COPS)

- Determination of the reaction coefficients in the thermal isomerization of α -pinene (COPS 5) [↗](#)
- Determination of stage specific growth and mortality rates for species at each stage as a function of time (COPS 6) [↗](#)
- Determination of the reaction coefficients for the catalytic cracking of gas oil and other byproducts (COPS 12) [↗](#)
- Determination of the reaction coefficients for the conversion of methanol into various hydrocarbons (COPS 13) [↗](#)
- Catalyst mixing in a tubular plug flow reactor (COPS 14) [↗](#)

Use Case 4 - Multi-scale modelling

MULTI-SCALE MODEL OF PHASE-SEPARATING BATTERY ELECTRODES ²

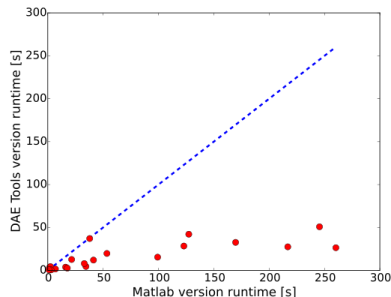
- Approach: **POROUS ELECTRODE THEORY**
- Lithium transport in:
 - Particles (small length scale)
 - Electrolyte (large length scale)
- Two phases are coupled via a volume-averaged approach
- Particles act as volumetric source/sink terms as they interact with the electrolyte via reactions
- The code available at **BITBUCKET**



²Li et al. (2014) *Current-induced transition from particle-by-particle to concurrent intercalation in phase-separating battery electrodes*. Nature Materials 13(12):1149–1156. doi:10.1038/nmat4084.

Use Case 4 - Multi-scale modelling (cont'd)

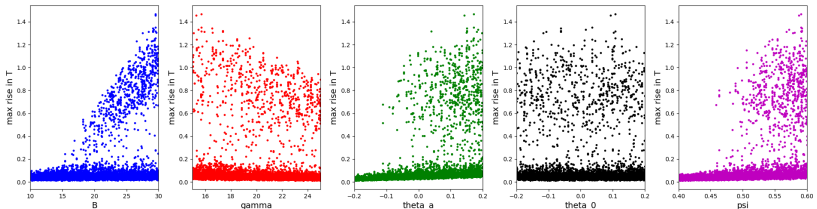
- Spatial discretisation: finite-volume method
- Large DAE system:
 - Discretised transport eqns.
 - Algebraic constraints (electrostatic eqns.)
 - Constraints on the current
- Implementations
 - MATLAB (ode15s solver)
 - DAE Tools (Sundials IDAS)
- DAE Tools up to 10x faster (average 4.22x) due to:
 - Built-in support for auto-differentiation
 - Rapid derivative evaluation
 - Accurate derivatives



Use Case 5 - Sensitivity Analysis

THERMAL ANALYSIS OF A BATCH REACTOR & EXOTHERMIC REACTION

- The global sensitivity analysis methods available via Python **SALIB** library
- Three sensitivity analysis methods applied:
 - **MORRIS** (Elementary Effect/Screening method)
 - **FAST** and **SOBOL** (Variance-based methods)
- **CALCULATIONS CAN BE PERFORMED IN PARALLEL** (Python **MULTIPROCESSING** module)
- Available information:
 - **1st AND 2nd ORDER SENSITIVITIES** and confidence intervals
 - **TOTAL SENSITIVITY INDICES** and confidence intervals
 - **SCATTER PLOTS**



Use Case 6 - Embedded simulator (back end)

NETWORK INTERCHANGE FORMAT FOR NEUROSCIENCE (NINEML)

XML-based DSL for modelling of networks of spiking neurones ↗

DAE Tools embedded into a **REFERENCE IMPLEMENTATION SIMULATOR**

○ ABSTRACTION LAYER (AL)

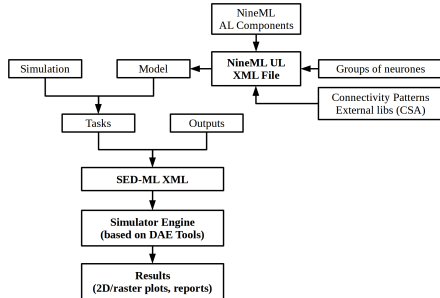
- Mathematical description
- Modelling concepts

○ USER LAYER (UL)

- Parameters values
- Instantiations

○ NineML concepts → DAE Tools concepts

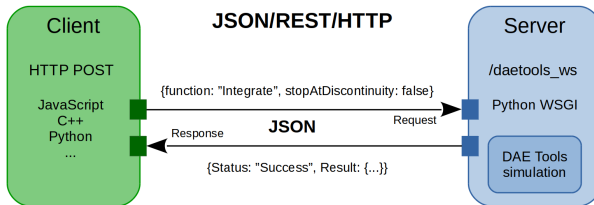
- Neurone models
- Synapse models
- Populations of neurones
- Layers of neurones



Use Case 7 - Software as a service

DAE TOOLS SIMULATIONS AS A WEB SERVICE

- RESTful API (JavaScript, Python, C++, ...)
 - DAE Tools simulations (daetools_ws)
 - DAE Tools FMU objects (daetools_fmi_ws)
- Benefits:
 - Application servers or individual simulations as a service
 - Attractive Graphical User Interface



Use Case 7 - Software as a service (cont'd)

Sample HTML GUI (JavaScript + plotly.js plotting library):

DAE Tools Web Service Client (status)

Load one of the tutorials

1 4 5 14 che_1 che_9

or:

Load simulation by name

Available simulations:

Loader function arguments (JSON format):

Load

Model Name:

tutorial_che_1

Time Horizon:

10800

Reporting Interval:

600

Run

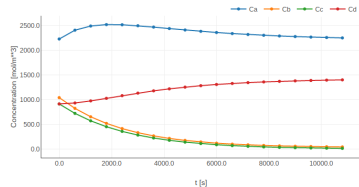
Simulation output:

```
Integrating from 5400.00 to 6000.00 ...  
Integrating from 6000.00 to 6600.00 ...  
Integrating from 6600.00 to 7200.00 ...  
Integrating from 7200.00 to 7800.00 ...  
Integrating from 7800.00 to 8400.00 ...  
Integrating from 8400.00 to 9000.00 ...  
Integrating from 9000.00 to 9600.00 ...  
Integrating from 9600.00 to 10200.00 ...  
Integrating from 10200.00 to 10800.00 ...  
The simulation has finished successfully!  
Preparing the plots...
```

100.0%

Plots:

Tutorial Che. 1 Concentration Plots



Tutorial Che. 1 Temperature Plots

