Open-source energy and chemical system models using ASCEND

John Pye# and Ben Allan+

#ANU Research School of Engineering, Canberra, Australia
+Sandia National Laboratories, Livermore CA
Outline

• Introduction: overview of ASCEND
• Architecture and development perspectives
• Examples: energy and chemical engineering
Introduction
ASCEND is a modelling environment

A specialised modelling language based around variables and relations

```plaintext
REQUIRE "atoms.a4l";
MODEL example1;
   x,y1,y2 IS_A factor;
   y1 = (x - 3)^2;
   y2 = 0.5*x + 1;
   y1 = y2;
END example1;
```

A compiler and solver framework

A user interface that supports examining the model, reconfiguring it, solving it, and playing with it.

ASCEND I,II,II and IV originally developed at CMU ~1978-2004

GTK GUI developed by Pye
How to use it

1. Edit your model in a text editor.
2. Load model in ASCEND and attempt to solve.
3. If you make a mistake, try again.
4. Once you have a solution, perform parameter studies, or use the model as a component in a large system.
Models are built up from other types

- fluid
- stream_state
- stream_node

- pump_simple
- turbine_simple

- heatex_pinch
- regenerator

- brayton
- rankine
- rankine_reheat

see models/johnpye/fprops/rankine_fprops.a4c
John Pye (ANU) and Ben Allan (volunteer)

Model: rankine;

- **BO** IS_A boiler_simple;
- **TU** IS_A turbine_simple;
- **CO** IS_A condenser_simple;
- **PU** IS_A pump_simple;

- **BO.outlet**, **TU.inlet** ARE_THESAME;
- **TU.outlet**, **CO.inlet** ARE_THESAME;
- **CO.outlet**, **PU.inlet** ARE_THESAME;
- **PU.outlet**, **BO.inlet** ARE_THESAME;

- **Qdot_loss** ALIASES CO.Qdot;
- **T_H** ALIASES BO.outlet.T;

Declare sub-models as instances of known model types

'Wire up' the flowsheet by merging input/output ports on the component models

Pull out 'aliases' of interesting variables from the submodels

Tree view GUI

- simroot
  - **BO**
  - **CO**
  - **PU**
    - **Qdot_loss**
    - **TU**
      - **Wdot**
      - **Wdot_eq**
      - **dp**
      - **eta**
      - **eta_eq**
      - **inlet**

<table>
<thead>
<tr>
<th></th>
<th>rankine</th>
<th>boiler_simple</th>
<th>condenser_simple</th>
<th>pump_simple</th>
<th>energy_rate</th>
<th>turbine_simple</th>
<th>energy_rate</th>
<th>relation</th>
<th>delta_pressure</th>
<th>fraction</th>
<th>relation</th>
<th>steam_node</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td>-19424.0414339 kW</td>
<td></td>
<td></td>
<td>14626.2971135 kW</td>
<td>5.98447513767e-10</td>
<td>-137.82571267 bar</td>
<td>1.0</td>
<td>0.0</td>
</tr>
</tbody>
</table>
'Squaring' the model

Typically there will be more variables in our model than equations.

We normally need to identify and fix some variables.

Sometimes if we are over-zealous, we may over-constrain the model (too many equations, not enough free variables).

It's not just a question of having equal numbers of variables and equations: the structure of the incidence matrix must be correct.

ASCEND provides tools to help achieve a square incidence matrix, and can do it interactively. This can be tough with really big models, nevertheless.
METHODS: procedural initialisation

- Automate the process of squaring the model
- Set starting guesses and upper/lower bounds, scaling that help the solver to work
- Structured approach: methods can call methods on submodels, parent models`
- Specify a solver and set its solver parameters

```c
METHOD default_self;
  RUN combinedcycle_fprops_common::default_self;
  (* starting guess, for easy solving *)
  HE.outlet_cold.h := 400 \{kJ/kg\};
  CO.outlet.h := 400 \{kJ/kg\};
END default_self;
METHOD on_load;
  RUN ClearAll;
  RUN default_self;
  RUN combinedcycle_fprops_common::specify;
  FIX Wdot; Wdot := 100 \{MW\};
  (* ambient conditions *)
  FIX GC.inlet.T, GC.inlet.p;
```

Calculations are possible, too, eg initialising a variable to be a sum of to other things, etc.

Often, _order_ is important, eg using upstream results to initialise downstream variables to realistic values.
Solvers

- QRSlv: steady nonlinear non-conditional
- CMSlv: steady nonlinear conditional
- LRSlv: boolean relations
- CONOPT: nonlinear optimisation (GRG)
- LSODE: multi-step ODE solver (AB/BDF)
- IPOPT: nonlinear optimisation
- IDA: multi-step DAE solver (AB/BDF)*
- DOPRI5: explicit Runge-Kutta ODE solver
- RADAU5: implicit Runge-Kutta ODE solver
- Linsolqr: sparse linear equation

*IDA has been extended recently to include boundary-traversal features (variable-structure dynamic modelling)

Results are returned back via the tree-view GUI.
Custom post-processing/reporting

External methods can access nodes in the simulation, get/set values, operate the solver, plot graphs, compute additional data if desired. We have a Python 'bridge' allowing methods to run arbitrary user code.

```
def cycle_plot_rankine(self):
    
    import loading
    loading.load_matplotlib(throw=True)
    ioff()
    figure()
    hold(1)
    D = fprops.fprops_fluid(str(self.cd.comp))
    sat_curve(D)

    boiler_curve = pconst(self.BO.inlet, self.BO.outlet)
    condenser_curve = pconst(self.CO.inlet, self.CO.outlet)
    SS = [self.PU.outlet, self.BO.inlet] + boiler_curve + condenser_curve + [self.CO.outlet, self.PU.outlet]
    plot_Ts(SS)
```

This example is extracting the name of the fluid from the simulation, plotting its saturation curves, and then plotting constant-pressure parts on a temperature-enthalpy diagram, which then appears as a pop-up window in the GUI.
Parameter studies

The observer tool is used to capture results from dynamic simulations, as well as parameter studies on steady-state systems. Includes basic x-y plotting for selected data columns.

Optimisation of compression ratio for a reheat-regenerative Brayton cycle gas turbine
External relations

IMPORT "freesteam";
MODEL steam_state;
  p IS_A pressure;
  h IS_A specific_enthalpy;
  T IS_A temperature;
  v IS_A specific_volume;
  s IS_A specific_entropy;
  x IS_A fraction;
  props: freesteam_Tvsx_ph(
    p, h : INPUT;
    T, v, s, x : OUTPUT
  );

Load external relations from a suitably-named file on ASCEND's search path
Reference to a named function loaded from the 'freesteam' external library
Each output variable results in an extra row in the incidence matrix.

So far we have only used C/C++ code by this route. Fortran easily wraps to C. Python is possible, but not trivial.

Part of the challenge here is to write external code that also returns partial derivatives wrt input variables (improve solver performance)
Custom scripting

# 'trivial example' from earlier: two roots.
from ascpy import *

L = Library()
L.load('example1.a4c')
T = L.findType('example1')
M = T.getSimulation('sim', True)
M.x.setRealValue(2.0)
M.solve(Solver('QRSlv'), SolverReporter())
print "first solution =", float(M.x)

M.x.setRealValue(4.0)
M.solve(Solver('QRSlv'), SolverReporter())
print "second solution =", float(M.x)

You can also dispense with the GUI completely, and control ASCEND directly from Python, C or C++. QRSlv is our default NLA solver.
Dynamic modelling

• Language constructs are still evolving
  – Ksenija Bestuzheva helped to implement DER(x,t) syntax

• We have single and multi-step solvers, implicit and explicit, and ODE and DAE.

• We have alpha support in our IDA solver for boundary-detection
  – events can be triggered, eg set velocity to the opposite direction
  – model structure can be changed, eg two-phase boiling instead of sensible heating

• Automatic index reduction not yet implemented, but possible. The needed diagnostics are all in place.
Canvas modeller – still very α

Visual programming and debugging on top of the ASCEND engine.

Cross-platform OSX, Win, Linux (Python, GTK3, Cairo)

So far, you can build a model, set component parameters, export and solver a flowsheet model. Definition of flow streams still needed.

Initial code by John Pye with input from Gaphor developer Arjan Molenaar. More recent GSOC contributions from Arijit, Grivan and Saheb
Architecture and development

Ben Allan
Sandia National Laboratories, Livermore CA
Today representing only himself.
ASCEND work is not supported by the Department of Energy.
Why ASCEND?

Enable *designers* and *implementers* of canned-model products to much more easily construct and deliver novel models of physical systems, whether sequential-modular or equation-based.

Critical chores enabled by or delegated to ASCEND:

- Separating the model “What” from the solution “How”
- Building up nested models from simpler ones by other modelers.
- Solving the model backward and evolving or devolving model fidelity.
- Detecting and correcting bad specifications, singular models.
- Deriving efficient solution processes.
- Tracking dimensions and units in equations & across interfaces.
- Keeping documentation (good names and formal text) accessible at all stages of model development.
Reusing or building on ASCEND?

Our main points today:
• Software architecture is mature (1990-2014)
• *Languages* are key
• Solver plugins for many problem types
• C module structure
• Model performance
• Chemical engineering capabilities
• Successful examples
• Licensing
Software Architecture

- Optional GUIs (Tcl/Tk, PyGTK)
- Scriptable (Python, Tcl) executive interfaces
- Engineering modeling language
- Foundation of modular C libraries
Modeling and executive languages

• We use scripting for software integration.
  – Python (15k lines) OR Tcl (50k lines) + Tk (35k lines)

• We drop down to C when performance matters.
  – Never a portability problem. (200k lines)

• We maintain a small, highly supportive model definition language.
  – Large languages (Fortran, C++) poorly support key modeling concepts.
  – We can't afford to maintain a large language.
  – Mapping models with a fixed solution process to large languages is trivial.
  – Hooking to external legacy model libraries is possible.
Solver engine plugins

• C APIs support DAE, MINLP, conditional programs, and any subtype found within these.
• John gave the current list
• Historical:
  – License bound: UMFPACK, MINOS, hsqp, cfsqp, CONOPT, CPLEX
  – Exporters: GAMS, Ipolve
• The quantity of successfully connected solvers indicates we probably have our APIs right for most solvers that are of interest.
• Multithreaded FOSS solvers are on the wish/todo lists.
C foundation

- Generic data structures
- Modeling language compiler
- Model data serialization
- Conversion of hierarchical models to flat for solvers.
- Equation structure analysis
- Equation gradients (AD,FD)
- Sparse linear solvers
- Nonlinear solvers

- External models and methods APIs
- Third-party solvers APIs
  - DAE
  - NLP (LP, CP, MINLP)
Model engine performance

• Function and gradient evaluation in C is optional (requires installed C compiler) and fast.

• Optimized interpreter (the default) is rarely slower than 50% of compiled C speed.

• Building models our way: No significant penalties for going from 5000 equations to 100,000.
Mature engineering capabilities

Design, steady, and dynamic flowsheeting
- Reaction, mixing, heat exchange, distillation, storage
- 1st order DAE BVP modeling
- Thermodynamics for VLE
  - 728 species from RPP*
  - Water and steam from FPROPS [Pye]
  - Ideal and Pitzer gases; Rackett, Wilson, UNIFAC liquids
  - Bring-your-own [Redlich-Kwong, Peng-Robinson, etc]

* Properties of Liquids and Gases, 4th Ed., Reid, Prausnitz, and Poling
Ease of rapid student contributions

• SQP solver integration
  – Dave Ternet (Biegler)

• Dynamic modeling
  – Duncan Coffey (Ydstie)

• Integer programming API and solvers
  – Craig Schmidt (Grossmann)

• GSOC students
ASCEND IV is

• A community with a history of welcoming newcomers and their contributions.
• One of the inspirations for Aspen Custom Modeler (alumni Piela, Epperly, others went to Aspen)
• Not a good tool for building PDE models on complex unstructured geometries.
  But for lumped systems and 1d grids and structured 2d and 3d grids it supports developing new solution methods and new modeling approaches unlike anything else freely available.
Mainly Open-source Licensing

• FOSS licensing was key in drawing students and support via Google Summer-of-Code
  – New GUIs, Hessians, Event handling,…
  – GPLv2 for modeling language, solvers, and GUI implementations developed by CMU, ANU.

• Unrestricted distribution of models (or exported code of models) developed by users.

• Third party solver licenses vary.
Examples: energy & chemical engineering
Chemical engineering

• Petrochemical complex [Abbott]
  – (100k+ eqns, RK thermo)

• NH3 plant dynamics [Monsen]
  – Proprietary thermo, unfortunately

• Pulp/paper processing [NTNU]

• Azeotropic distillation [numerous]

• Pareto optimization via executive scripting
Energy system modelling

• Supercritical CO$_2$ Brayton power cycle
• Molten salt receiver for CSP
• Parabolic trough plant
• Dynamic modelling of 'direct steam generation' in a solar collector
sCO2 power cycle

This power cycle has been proposed for use in solar-thermal power plants because of compact size and very high efficiency at moderate temperatures.

Turbine, compressor are isobaric with simple isentropic efficiency given as a fixed parameter.
Heat exchanger models

The `heatex_pinch` externally-coded component model allows minimum delta-T to be specified as *input*. 
sCO2 implementation

- Reusable turbine, pump/compressor, heat exchanger, tee/join components developed
- Support for high-accuracy Helmholtz fundamental equation of state for properties of CO2 provided through FPROPS library, developed in parallel (*more later*)
- Python scripts for visualisation of cycle, heat exchanger
- `heatex_pinch` external relation implemented to allow improved cycle optimisation by specification of heat exchanger pinch point temperature difference as input
Other cycles, same thermo/components
Molten salt receiver

One-dimensional flow model needed to forecast losses and examine effect of switching to different heat transfer fluids.

Pye et al, SolarPACES 2014, Beijing
receiver_flow_segment

\[ A_{\text{ext, slice}} = \frac{1}{2} \pi d_{\text{ext}} L_{\text{slice}} \]

\[ L_{\text{slice}} \]

\[ \begin{align*}
V_i & \quad h_i & \quad \rho_i \\
\mu & \quad k & \quad c_p & \quad T \\
\rho & \quad Q_i & \quad T_{\text{int}} \\
C & \quad G & \quad A_i & \quad Q_{\text{abs}}
\end{align*} \]

\[ \begin{align*}
V_{i+1} & \quad h_{i+1} \\
\rho_{i+1} & \quad d_{\text{int}} \\
\rho_{i+1} & \quad d_{\text{ext}}
\end{align*} \]
Exergy analysis

This Sankey/Grassman diagram describes how the usable exergy from the sun is lost as it passes from radiation through, and was drawn via a Python script hooked to an ASCEND method. The Python script fetches all the needed data from the model and draws the diagram.

\[
X_{\text{net}} = X_{\text{out}} - X_{\text{in}}
\]

\[
\eta_{II} = \frac{X_{\text{net}}}{X_{\text{sun}}}
\]

\[
X_{\text{in}} = \dot{m} \phi_{\text{in}}
\]

\[
X_{\text{out}} = \dot{m} \phi_{\text{out}}
\]
For this study, the FPROPS code was used for CO2, but ideal air and incompressible sodium and salt were modelled. This was done by superclassing the fluid properties model and allowing the different correlations to be switched within that superclass.

- **Molten salt** (NaNO3/KNO3): incompressible ($\rho, c_p, \mu, k$ functions of $T$) Zavoico SAND2001-2100
- **Sodium**: incompressible Fink and Liebowitz
- **CO$_2$**: Helmholtz equation of state + correlations for $\mu, k$. Span and Wagner, Fengour, Vesovic
- **Air**: ideal gas + correlations for $\mu, k$ Lo, Kadoya

**External air**: Holman tables for air at ambient pressure
Implementation

• Equation-based model in ASCEND
  – Mesh invariance: ~50-100 flow elements
  – ~2000 variables and relations
  – Converges quickly with standard solver; no coding needed for iterative loops, numerical routines etc.

• FPROPS (C code) extended to implement viscosity and thermal conductivity correlations for CO$_2$. 

http://ascend4.org/
Results for 'realistic' basis

- salt baseline
- sodium 500–850°C
- CO₂ 500–700°C
- air 700–890°C
- sodium cavity
- sodium high-flux

Exergy rate / [MW]

- absorption
- reflection
- convection
- thermal emission
- internal convection
- friction
- wall conduction
- net exergy to working fluid
Quasi-dynamic annual performance

A basic study on potential for integration of CSP with Birdsville geothermal plant (Emily Do, 2011)

Weather data reader (TMY, etc), together with NREL sunpos routine
(both implemented as external relations)

A range of reusable components eg parabolic_trough

Figure 4.8 Combined Geothermal and Solar Thermal Model Performance for total collector area of 4000m², January 2010
Dynamic modelling of two-phase flow


Simulation of a 100 m long direct steam generation solar boiler (linear Fresnel system), 60 mm internal diameter, 40 flow segments, 0.26 kg/s. Implemented as DAE equation system in ASCEND.

Model was first solved in steady-state with the QRSlv solver, for a solar irradiance set at 335 W/m per collector length.

Simulation results show temperatures and flow rates calculated from IDA solver for step change in solar flux up to 6000 W/m as solar boiling of the water takes places and is gradually 'squeezed' out the end.
Partial support for this work came from the Australian Renewable Energy Agency (ARENA), the Australian National University, and the Google Summer of Code program.
Comparison with other tools
Criteria

- open-source core code, GUI and model library
- all-platform support
- conditional/hybrid modelling
- reusable/hierarchical language
- user-friendly 'creep up on a solution' features
- solver nesting (e.g., optimisation of dynamical system, optimal control/model predictive control)
- user community
- implementation in well-known coding language(s)
- parallel processing features?

http://ascend4.org/Other_modelling_tools