University of South-Eastern Norway

TMCC

Telemark Modeling and Control Center

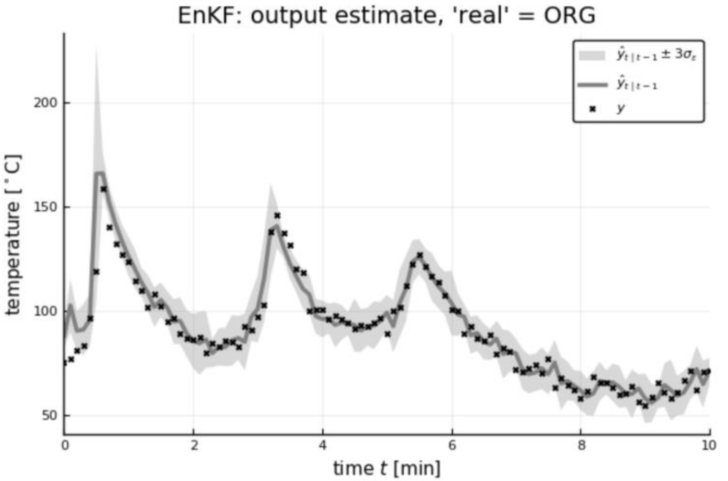
LINKÖPING UNIVERSITY

OpenModelica API for Accessing Modelica Models from Julia

Bernt Lie; USN

Arunkumar Palanisamy, Alachew Mengist, Lena Buffoni, Martin Sjölund, Adeel Asghar, **Adrian Pop**, Peter Fritzson; Linköping University


```
model ModSeborgCSTRorg
  // Model of ORiGinal Seborg CSTR in ode form
  // author:   Bernt Lie
  //           University of South-Eastern Norway
  //           November 7, 2017
  //
  // Parameters
  parameter Real V = 100 "Reactor volume, L";
  parameter Real rho = 1e3 "Liquid density, g/L";
  parameter Real a = 1 "Stoichiometric constant, -";
  parameter Real EdR = 8750 "Activation temperature, K";
  parameter Real k0 = exp(EdR/350) "Pre-exponential factor,
  parameter Real cph = 0.239 "Specific heat capacity of mi
  parameter Real DrHt = -5e4 "Molar enthalpy of reaction,
  parameter Real UA = 5e4 "Heat transfer parameter, J/(min
  //
  pa
  pa
  //
  //
  Re
  Re
  //
  Re
  Re
  Re
  //
  in
  in
  in
  in
  //
  ou
```



```
model ModSeborgCSTRorg
  // Model of ORiGinal Seborg CSTR in ode form
  // author:  Bernt Lie
  //          University of South-Eastern Norway
  //          November 7, 2017
  //
  // Parameters
  parameter Real V = 100 "Reactor volume, L";
  parameter Real rho = 1e3 "Liquid density, g/L";
  parameter Real a = 1 "Stoichiometric constant, -";
  parameter Real EdR = 8750 "Activation temperature, K";
  parameter Real k0 = exp(EdR/350) "Pre-exponential factor,
  parameter Real cph = 0.239 "Specific heat capacity of mix
  parameter Real DrHt = -5e4 "Molar enthalpy of reaction, J
  parameter Real UA = 5e4 "Heat transfer parameter, J/(min.

  // Initial state parameters
  parameter Real cA0 = 0.5 "Initial concentration of A, mol
  parameter Real T0 = 350 "Initial temperature, K";
  // Declaring variables
  // -- states
  Real cA(start = cA0, fixed = true) "Initializing concentr
  Real T(start = T0, fixed = true) "Initializing temperatur
  // -- auxiliary variables
  Real r "Rate of reaction, mol/(L.s)";
  Real k "Reaction 'constant', ...";
  Real Qd "Heat flow rate, J/min";
  // -- input variables
  input Real Vdi "Volumetric flow rate through reactor, L/m
  input Real cAi "Influent molar concentration of A, mol/L"
  input Real Ti "Influent temperature, K";
  input Real Tc "Cooling temperature', K";
  // -- output variables
  output Real y_T "Reactor temperature, K";
```

Overview

- Why scripting OpenModelica?
- Design choices
- OMJulia design
- Examples
- Conclusions

Why scripting OpenModelica?

Modelica

- Direct encoding of DAE models, equation based
- Excellent support for libraries

Modelica based tools

- Varying level of eco system
- Poor support for random numbers
- Varying support for control design, etc.
- Some support of plotting
- Some support for optimization (Optimica)

OpenModelica and scripting

- Basic scripting in OM
- Specialized scripting languages have rich eco system
  - MATLAB: large user base, good documentation, expensive
  - Python: large user base, medium documentation, little support for control community, free
  - Julia: small but growing user base, scant documentation, decent support for control community, free

Design choices

- Scripting strings
  - Send OM commands to OM via ZMQ
  - OMPython, originally
- Script language API
  - Commands native to script language
  - Glue tool to translate to-from string OM commands
  - E.g., OMPython, OMJulia

- Translate to script
  - OM code translates to C code, then compiles to .exe
  - Alternative: translate to native script code?
  - Advantage:
    - Use existing Modelica code
    - Continue to write Modelica code
  - Disadvantage:
    - Two-language solution

- Extend script language with Modelica structures
  - One-language solution
  - Best possible synergy
  - Script language may limit possibilities
  - Example:
    - Media for Julia
    - Full integration requires full use of type system
    - Utilize dispatch & use standard names (*solve*, not *simulate*, etc.)

OMJulia design

General

- Origin: Python API
- User demand:
  - (Python API)
  - MATLAB API
  - Julia API
- Ease of maintenance *essential*
- Tool developer needs to "own" API
- Ease of maintenance may require breaking best practice for languages

Compromises

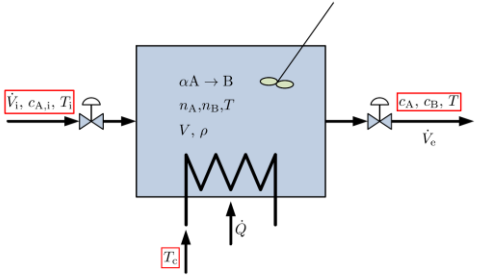
- Julia *type* based + *multiple dispatch* by design
- Desired Julia syntax:
  - `solve(model,…)`
- Python is object oriented
- Typical Python syntax
  - `model.solve()`
- Although not recommended, object syntax possible in Julia
- Compromise: object syntax to ease maintenance

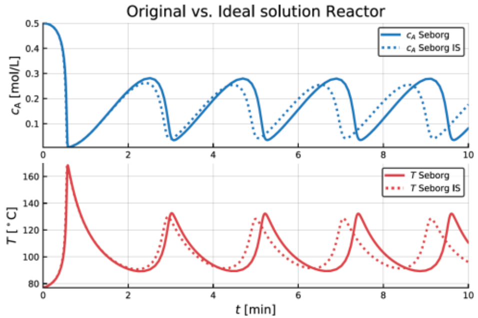
API examples

```
> m =
OMJulia.OMCSession()
> m.ModelicaSystem(
"SeborgCSTR.mo",
"SeborgCSTR")
>
m.setInputs(["Tc=300",
”Ti=350”])
> m.simulate()
> tm, T, Tc, cA =
m.getSolutions(["time"
,"T","Tc","cA"]);
```

Examples I: nonlinear reactor + PI tuning

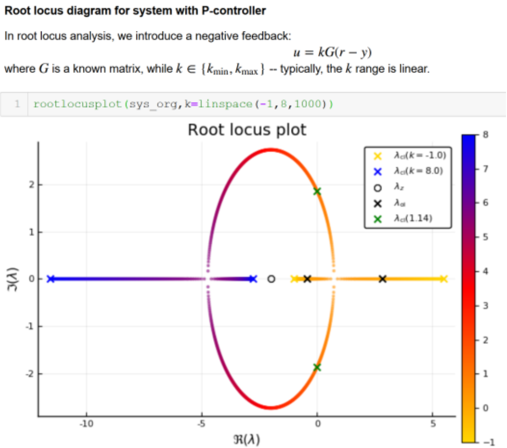
Seborg reactor:





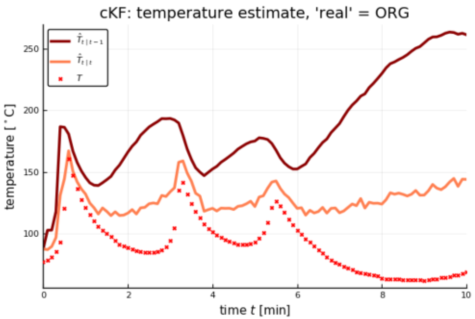
PI control design:

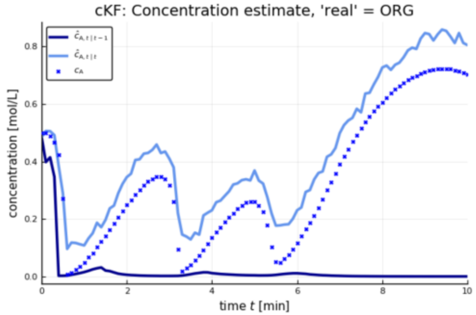
- OMJulia linearization
- Julia ControlSystems, LTI tools
- Root locus PI tuning

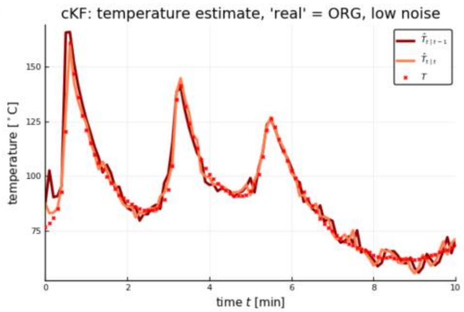


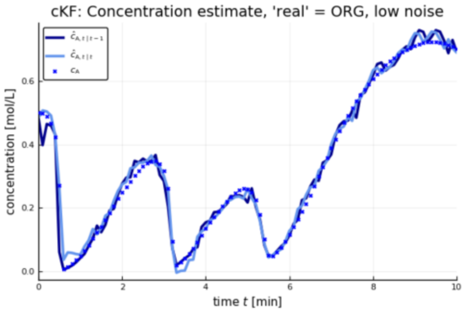
Examples II: state estimation of nonlinear reactor

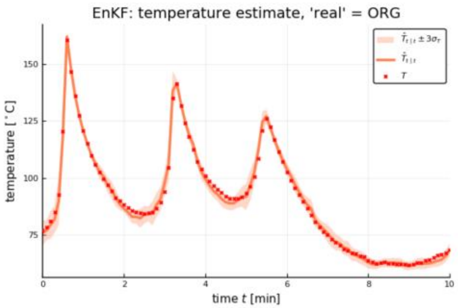
- cEKF (constant gain)
- cEKF, low noise
- EnKF (10 particles)

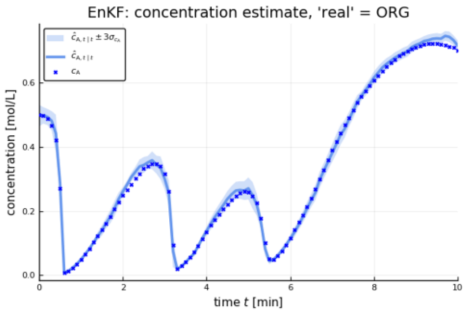






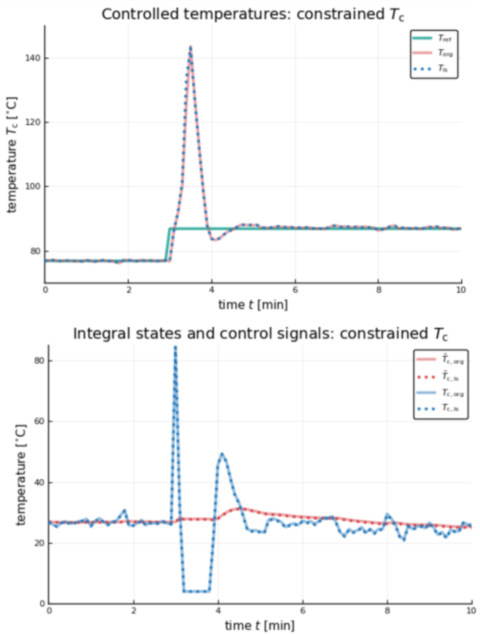




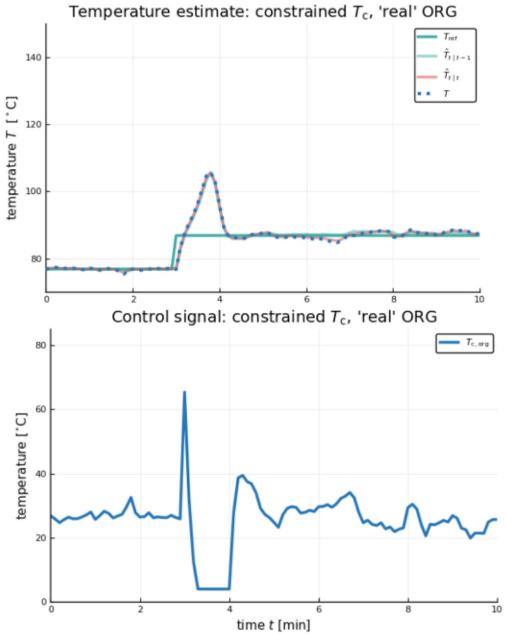


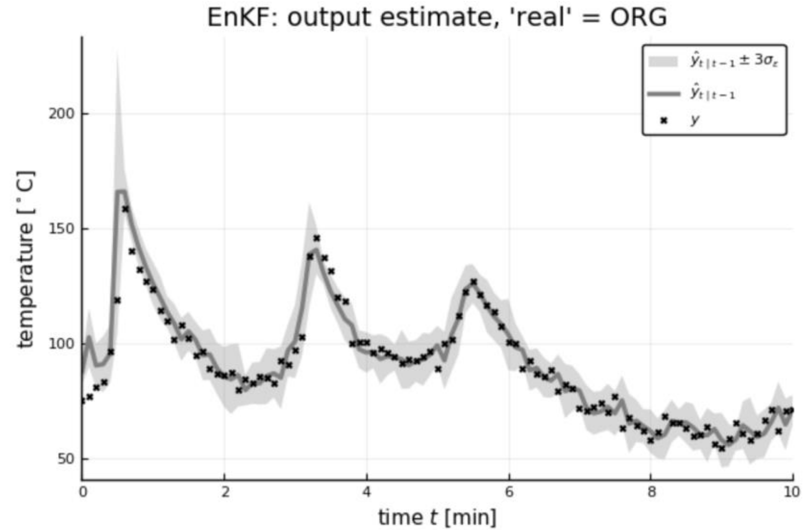
Examples III: linear control of nonlinear reactor

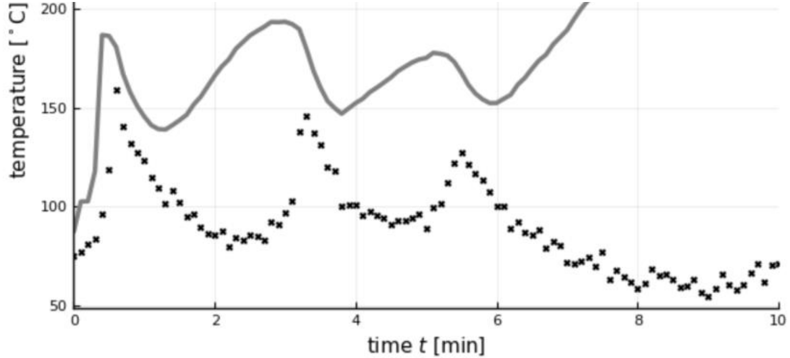
- PI control with constraint



- LQG+I control with constraint







Conclusions

- Scripting Modelica code expands possibilities for analysis/user base
- Ease of maintenance -> compromises when supporting several script languages
- Important to involve Modelica tool developer to ensure compatibility
- Julia is an exciting new language:
  - Control tools, fast execution, 2k> packages
- (Open)Modelica offers:
  - Rich language for model description
  - Good library support