Industrial implementation of nonlinear model predictive controllers
21st Nordic Process Control Workshop in Åbo, Finland

Jørgen K. H. Knudsen
2-control ApS, Denmark

January 18, 2018
One of the key players in a successful Advanced Process Control application is the plants process engineer. The process engineer

- has process knowhow
- is experienced with process operation
- knows chemistry, heat and mass balances
- normally don’t understand optimization mathematics (Matrix calculations, qpsolvers, convex constraints etc.)
- is not using programming languages as C# or Matlab in his daily work

We have to provide tools which can be used by the process engineer, requiring a minimum of programming skills and training, and which can be used occasionally, whenever maintenance of the control system is required.
The plant operators supervise the daily operation of the plant. The plant operator

- has experience with process
- is responsible for plant operation and security
- is in the drivers seat, determining whether an APC control loop is active or should be switched off

There must be good and understandable graphical user interfaces enabling the operator to understand and approve the actions of the APC loop. Before moving an APC application to the control room and plant, it’s important to use simulation tools to test and make preliminary tuning of the system.

- If the operators looses confidence to the APC system, it will be virtually impossiblye to achieve a successfull installation
Today there is a lot of good tools for model predictive controllers based on linear models.

- Linear models can be defined and entered into the APC control system using statespace models or transfer functions.

Non-Linear models are much more difficult. Without special tools:

- Models can be developed using high level programming languages as MATLAB or C#.
- Programming the Jacobians are tedious and error prone.
- Requires MATLAB or Compiler.
**ModelBuilder**

*ModelBuilder* is a tool for development and test of nonlinear models

- High abstraction level
- No compiler or Matlab required
- Generates Jacobians automatically
- Integral part of the *APCSuite* product

*APCSuite* has control algorithms able to use the nonlinear *ModelBuilder* models
**APCSuite** is an industrial environment for implementation of advanced process control applications based on linear and/or nonlinear models.

- **APCBox** is a windows based server holding the actual controller. **APCBox** runs 24/7.
- **APCBox** exchanges data and setpoints with the plants DCS/SCADA system using a standard OPC interface.
- **APCView** is the client program used to supervise and configure the **APCBox** functions.
The ULoop reactor

The *ModelBuilder* and non-linear controllers will be demonstrated on a ULoop reactor used to produce single cell protein.

The biomass consumes substrate (methanol) and oxygen and produces single cell proteins. The main reaction takes place in the U shaped plug flow reactor, where the gases and the biomass are mixed efficiently.
The growth rate of the biomass is a highly nonlinear function of the methanol concentration, making it a challenging task to control the U-Loop reactor.
ModelBuilder Source Code

ModelBuilder Source Code consist of a declaration section followed by an equations section

Model ODE (....)

declarations ...

Equations

equations ....

End

- The *Model* statement defines the model type (ODE or DAE) and defines some general information around the handling of the model.
- The *declarations* statements defines constants, parameters, variables, states, model inputs, and model outputs.
- The *equations* defines the relations between the entities defined under the declaration section.
Constants and parameters are defined as shown below:

```plaintext
// Yield Coefficients
Constant Yso = 0.439;
Constant Ysx = 0.732;
// Molar Weights
Constant Mws = 1.0079*4+12.011+15.9994; // [S = CH3OH][g/mol]
Constant Mwo = 2*15.9994; // [O = O2][g/mol]
Constant Mwx = 12.011+1.8*1.0079+0.5*15.9994+0.2*14.0067; // [X][g/mol]
// Kinetic Parameters
Parameter mumax = 0.37; // [[]][1/hr]
Parameter ks = 0.021; // [[]][kg/m3]
Parameter ki = 0.38; // [[]][kg/m3]
Parameter ko = 6.4e-5; // [[]][kg/m3]
```

Assignments can be expressions. Parameters can be changed from outside the model.
Declaration of states and inputs:

// CSTR states
State Cxcstr = 10.3; // [protein concentration][mol/m3]
State Cscstr = 0.01; // [substrate concentration][mol/m3]
State Cocstr = 0.123; // [oxygen concentration][mol/m3]

Input Fs = 0.00411; // [Substrate input][m3/hr]
Input Fw = 0.1736; // [Water input][m3/hr]
Input Fg = 0.7034; // [gas input][m3/hr]

The positions in the $x$ and $u$ vectors follows the declaration sequence. The states and inputs can be assigned initial values.
ModelBuilder Variables and outputs

Declaration of variables:

// Mixer
Var Fliq;   // [Liquid flow][m3/hr]
Var Ftot;   // [Total flow][m3/hr]
Var epsilon; // [gas/liquid ratio][
Var v;     // [PFR flux][m/hr];

The values of the variables must be a function of the states and inputs, as described in the equation section.
Finally the outputs $y$ can be defined as:

Output Cxcstr; // [protein concentration][mol/m3]
Output Cscstr; // [substrate concentration][mol/m3]
Output Cocstr; // [oxygen concentration][mol/m3]

The positions in the $y$ vector follows the declaration sequence.
Equations
F = Fs + Fw;  // Total flow
// Mixer
// Inlet boundary conditions for the PFR
Fliq = F + FR;
Ftot = Fliq + Fg;
epsilon = Fg / Ftot;
v = Ftot /Apfr;
.
Cxpfrin = (Fs * Cxin + FR * Cxcstr) / Fliq;
Cspfrin = (Fs * Cmet + FR * Cscstr) / Fliq;
Copfrin = (Fs * Coin + FR * Cocstr) / Fliq;
Cgopfrin = Cgoin;

// Differential equations for CSTR
rx[N] = Cxcstr * Mu(Cscstr,Cocstr);
time * der(Cxcstr) = (Fliq / Vcstr) * (Cx[N - 1] - Cxcstr) + rx[N];
time * der(Cscstr) = (Fliq / Vcstr) * (Cs[N - 1] - Cscstr - gammaS * rx[N];
time * der(Cocstr) = (Fliq / Vcstr) * (Co[N - 1] - Cocstr) - gammaO * rx[N];
.
End
The equation section is used to specify the relations between inputs, variables and states.

The operator \( \text{der}(C_{xcstr}) \) indicates the time derivative of a state variable \( C_{xcstr} \).

\textit{ModelBuilder} automatically sorts the equations in the correct sequence. The equations are of the type \( \text{expression} = \text{expression} \), not just simple assignments.

The number of equations must be equal to the number of states and variables.


Reaction rates and other functions can be defined in the declaration section.
Arrays of States and variables

The plugflow reactor is discretized into a number of equally sized cells. In order to handle discretized systems, States and variables can be declared as arrays in up to 3 dimensions

// Plug Flow reactor states
Constant N = 15;  // [Number of Discretizations]

// Plug Flow reactor states
State Cx[N];  // [protein concentration][mol/m3]
State Cs[N];  // [substrate concentration][mol/m3]
State Co[N];  // [oxygen concentration][mol/m3]
State Cgo[N]; // [oxygen concentration gas phase][mol/m3]

// Flux
Var Nx[N+1];  // [protein flux][mol/m]
Var Ns[N+1];  // [substrate flux][mol/m]
Var No[N+1];  // [oxygen flux][mol/m]
Var Ngo[N+1]; // [oxygen gas phase flux][mol/m]

The statement \( State Cx[N] \); is equivalent to declaring the States \( C[0], C[1].....C[15] \)
Discretizations for distributed systems

In the Dynamics section the loop operator can be used to specify equations for the lumped sections

\[
\begin{align*}
\text{No}[N] &= v \times \text{Co}[N - 1]; \\
\text{Ngo}[N] &= v \times \text{Cgo}[N - 1]; \\
\text{Repeat}(0 : N - 1) & \{ \\
    // \text{Gas-Liquid Transport} \\
    \text{Jglo}[ix] &= \text{kla0} \times (\text{RTdivMwoHo} \times \text{Cgo}[ix] - \text{Co}[ix]); \\
    \text{rx}[ix] &= \text{Cx}[ix] \times \text{Mu}(\text{Cs}[ix], \text{Co}[ix]); \\
    \text{time} \times \text{der}(\text{Cx}[ix]) &= -(\text{Nx}[ix + 1] - \text{Nx}[ix]) / \text{dz} + \text{rx}[ix]; \\
    \text{time} \times \text{der}(\text{Cs}[ix]) &= -(\text{Ns}[ix + 1] - \text{Ns}[ix]) / \text{dz} - \text{gammaS} \times \text{rx}[ix]; \\
    \text{time} \times \text{der}(\text{Co}[ix]) &= -(\text{No}[ix + 1] - \text{No}[ix]) / \text{dz} - \text{gammaO} \times \text{rx}[ix] \\
    &\quad + \text{Jglo}[ix] / (1.0 - \text{epsilon}); \\
    \text{time} \times \text{der}(\text{Cgo}[ix]) &= -(\text{Ngo}[ix + 1] - \text{Ngo}[ix]) / \text{dz} - \text{Jglo}[ix] / \text{epsilon}; \\
\} \\
// \text{Differential equations for CSTR} \\
\text{rx}[N] &= \text{Cxcstr} \times \text{Mu}(\text{Cscstr}, \text{Cocstr});
\end{align*}
\]

Where \textit{ix} is given the values 1, 2, ..... , 14 in the repeat section
The final ULoop model
From *ModelBuilder* to StateSpace Model

The *ModelBuilder* model is a DAE model with Jacobians

\[
F(x, \dot{x}, u) = 0 \quad (1)
\]

\[
\nabla_x F(x, \dot{x}, u) \quad (2)
\]

\[
\nabla_{\dot{x}} F(x, \dot{x}, u) \quad (3)
\]

\[
\nabla_u F(x, \dot{x}, u) \quad (4)
\]

The statespace model around an operating point \( x, u \) with step length \( DT \), can be calculated efficiently using orthogonal collocation, based on Radau/Lobatto polynomials:

\[
X_{k+1} = A_k X_k + B_k U_k + O f f_k
\]

The MPC controllers in *APCSuite* uses a set of statespace models to control the process
The control cycle

1. Initialize Models
2. Read Plant outputs
3. MPC Algorithm (Linear)
4. Write Plant setpoints
5. Update Models

ModelProvider
The control cycle

- The plant outputs are read via the OPC connection to the plant PLC, and the current state of the plant is estimated with a Kalmann filter.
- The MPC algorithm runs as a conventional linear MPC using state space models from the ModelProvider, and calculates setpoints for plant inputs together with the anticipated future trajectory of the plant.
- Setpoints are written to the plant PLC.
- Finally the idle time until next sample time is used to update the models in the ModelProvider according to the new future trajectory calculated by the MPC controller.
Model update strategies

The state space models in the ModelProvider are updated according to different strategies:

- **None**: Use one model based on the initial state *ModelBuilder* source code (Normal linear MPC control)
- **Initial**: Use one model based on the current state of the process (Partly nonlinear MPC)
- **All**: One model for each time step in the control horizon (Fully nonlinear MPC)
Control horizon = 300 steps, Update = All
300 models updated after each control cycle
- The curvature of the Cx curve has changed and the control signals are different
ULoop Response converged
Conclusions

- We must provide simple tools aimed at process engineers.
- *ModelBuilder* is a simple and efficient tool for development of non-linear models for MPC.
- The relatively simple strategy by using the linear algorithms combined with subsequent updating of the models, works for difficult cases as the U-Loop reactor.
- Using the linear algorithm provides a fast immediate response to the process.
- For unstable systems, a more complicated strategy based on ”multiple-shooting” algorithms might be required.
Questions and Comments

Jørgen K. H. Knudsen
joek@2-control.dk

2-control Aps
Frimodtsvej 11
DK-2900 Hellerup
Denmark