Implementing Model Predictive Control in the C#/.Net Environment

Jørgen K H Knudsen
2-control ApS
Model Based Control Conference
DTU, May 5, 2011

Microsoft C#/.NET environment

- C# and .NET is Microsoft's solution to development of client/server systems
  - Managed code
  - Very comprehensive sw, with rich fundamental building blocks
    - WPF Windows presentation foundation for Graphics
    - WCF Windows Communication Foundation
    - WF   Windows Work-flow
- Many of the next generation of DCS and SCADA systems will be developed in this environment.

Advanced Process Control, MPC

- Development and training in APC and MPC is often done in scripting environments as Matlab, Python or Octave.
  - High productivity, fast development, tool boxes available
  - Not designed for online servers
    - Must run 24/7
- There is a need for a C#/.NET based library for development and implementation of MPC

MPCMath

- Library for implementation of real time Advanced Process Control, Model Predictive Control and other Optimization tasks.
- Intelligent objects as building stones for implementing control systems
  - Matrix, vector objects
  - Transfer functions
  - ARX and State Space models
  - ....
- Written in C#

Linear Algebra

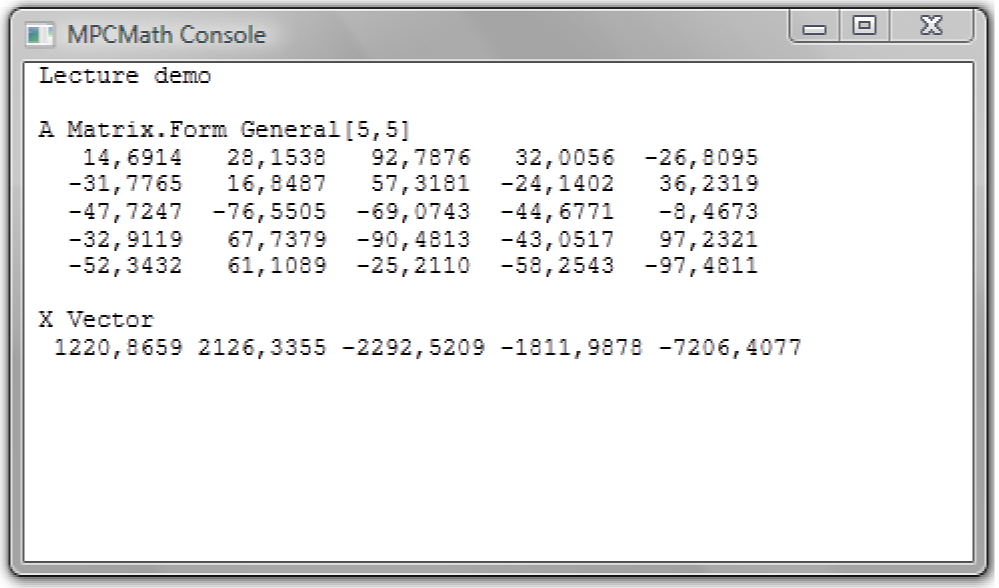
- Vector Matrix objects

```
int nx = 5;
int nu = 2;
int ny = 2;

Matrix A = Matrix.Random(nx, nx, -100.0, 100.0);
Matrix B = Matrix.Random(nx, nu, -100.0, 100.0);
Vector X = Vector.Random(nx, -100.0, 100.0);
Vector U = Vector.Random(nu, -100.0, 100.0);

X = A * X + B * U;

Matrix.Show("A", A);
Vector.Show("X", X);
```

Linear Algebra

- Vector Matrix objects

```
int nx = 5;
int nu = 2;
int ny = 2;

Matrix A = Matrix.Random(nx, nx, -100.0, 100.0);
Matrix B = Matrix.Random(nx, nu, -100.0, 100.0);
Vector X = Vector.Random(nx, -100.0, 100.0);
Vector U = Vector.Random(nu, -100.0, 100.0);

X = A * X + B * U;

Matrix.Show("A", A);
Vector.Show("X", X);
```

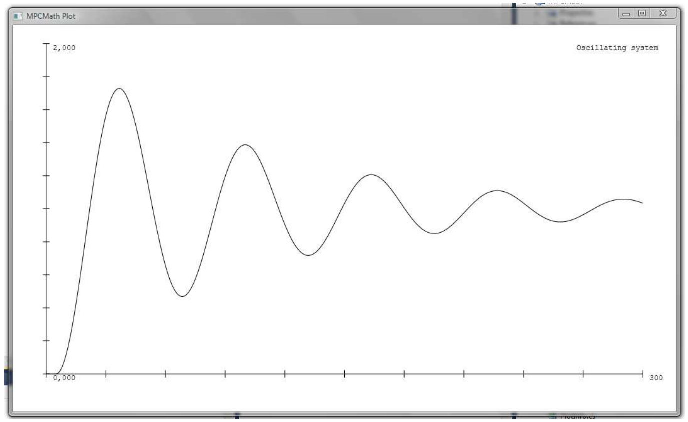


TransferFunctions

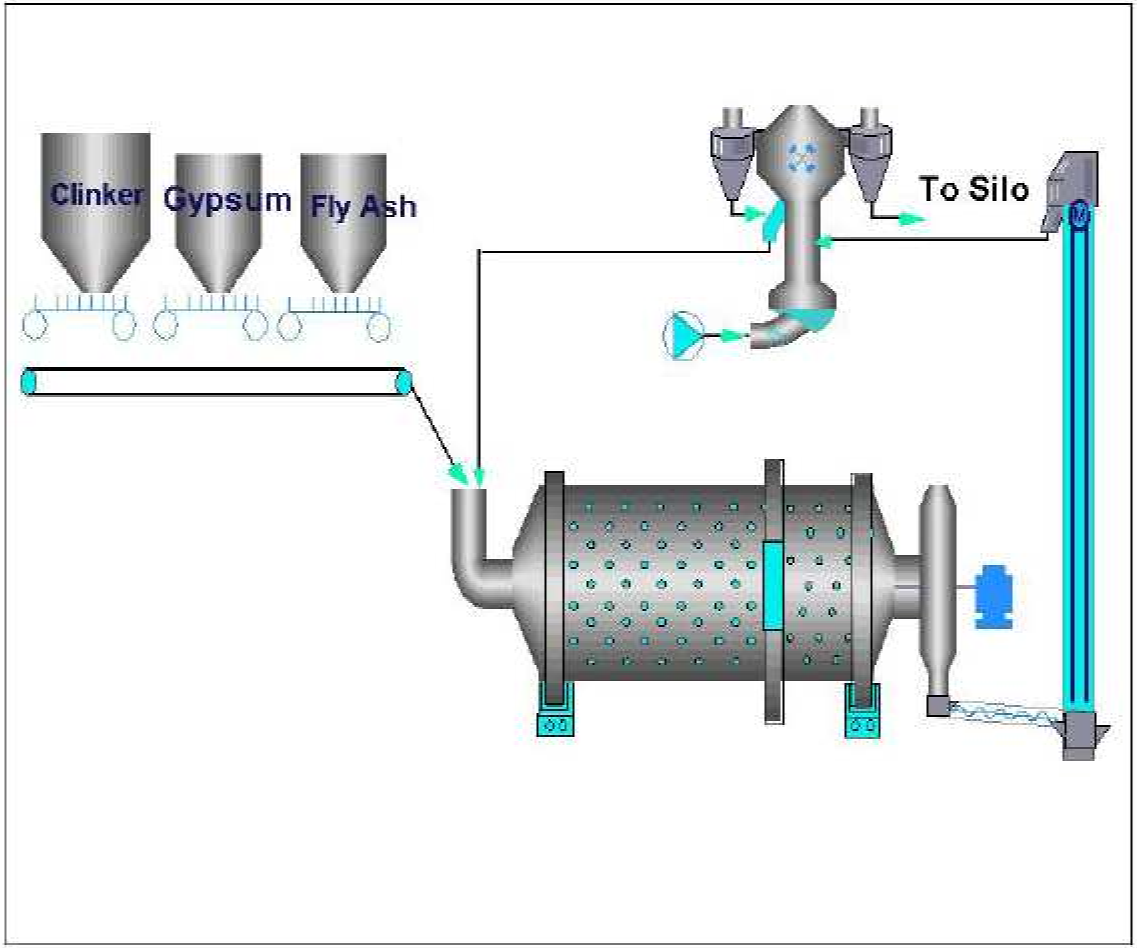
```
// step function
double gain = 1.0;
double delay = 5.0;
TransferFunction Step = new TransferFunction(gain,delay);

// Oscillating Second order system
double tau = 10.0;
double sigma = 0.1;
TransferFunction Oscillating = new TransferFunction(gain, 0.0, tau, sigma);

TransferFunction G = Oscillating * Step;

int steps = 300;
Vector resp = new Vector(steps);
for (int step = 0; step < steps; step++)
{
    double time = step;
    resp[step] = G.Value(time);
}

console.Plot(new PlotSeries("Oscillating system", 0.0, 2.0, resp));
```

TransferFunctions

```
// step function
double gain = 1.0;
double delay = 5.0;
TransferFunction Step = new TransferFunction(gain,delay);

// Oscillating Second order system
double tau = 10.0;
double sigma = 0.1;
TransferFunction Oscillating = new TransferFunction(gain, 0.0, tau, sigma);

TransferFunction G = Oscillating * Step;

int steps = 300;
Vector resp = new Vector(steps);
for (int step = 0; step < steps; step++)
{
    double time = step;
    resp[step] = G.Value(time);
}

console.Plot(new PlotSeries("Oscillating system", 0.0, 2.0, resp));
```



Cement Mill example



- Controlled variables
  - Elevator load
  - Fineness
- Manipulated variables
  - Feed
  - Separator speed

$$G(s) = \begin{bmatrix} \frac{0.62}{(45s+1)(8s+1)}e^{-5s} & \frac{0.29(8s+1)}{(2s+1)(38s+1)}e^{-1.5s} \\ \frac{(-15)}{(60s+1)}e^{-5s} & \frac{5}{(14s+1)(s+1)}e^{-0.1s} \end{bmatrix}$$

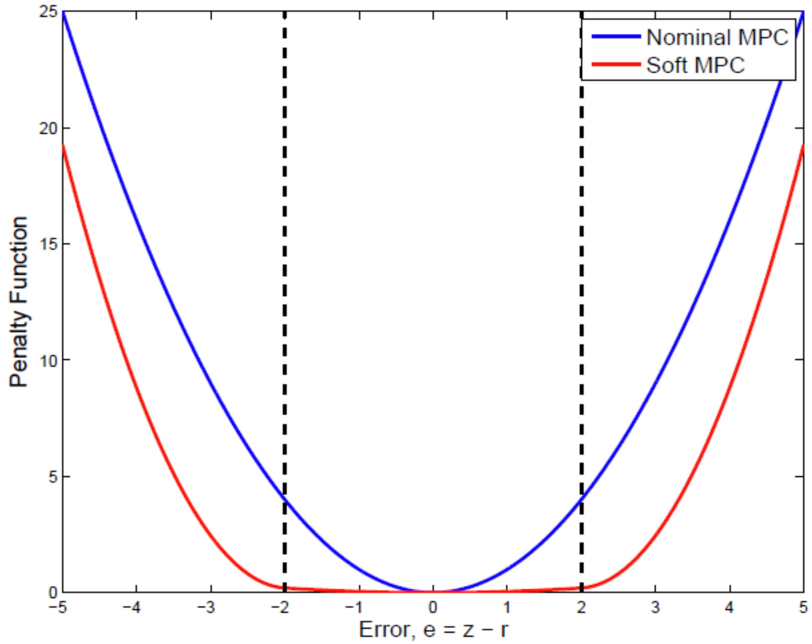
Prasath, G.; Recke, B.; Chidambaram, M.; Jørgensen, J.B.: Application of Soft Constrained MPC to a Cement Mill Circuit, 9th International Symposium on Dynamics and Control of Process Systems, DYCOPS 2010, 288-293 ,Leuven, Belgium, 2010

Objective

$$\min_{\{z,u,\eta\}} \phi = \frac{1}{2}\sum_{k=0}^{N-1} \|z_{k+1} - r_{k+1}\|_{Q_z}^2 + \|\Delta u_k\|_S^2 + \sum_{k=1}^{N} \frac{1}{2}\|\eta_k\|_{S_\eta}^2 + s_\eta' \eta_k$$

Subject to the constraints:

$$\begin{aligned}
z_k &= b_k + \sum_{i=1}^{n} H_i u_{k-i} && k = 1, \dots N \\
u_{\min} &\le u_k \le u_{\max} && k = 0, \dots N-1 \\
\Delta u_{\min} &\le \Delta u_k \le \Delta u_{\max} && k = 0, \dots N-1 \\
z_k &\le z_{\max,k} + \eta_k && k = 1, \dots N \\
z_k &\ge z_{\min,k} - \eta_k && k = 1, \dots N \\
\eta_k &\ge 0 && k = 1, \dots N
\end{aligned}$$



Defining StateSpace models

```
// Cement mill MIMO system
Double T = 1.0;
int ny = 2;
int nu = 2;

Matrix<TransferFunction> CMModel =
    new Matrix<TransferFunction>(MatrixForm.General, ny, nu);

//                                        gain   delay          taus
CMModel[0, 0] = TransferFunction.TauForm(0.62,  5.0, new Vector(45.0, 8.0));
CMModel[1, 0] = TransferFunction.TauForm(-15.0, 5.0, new Vector(60.0));
CMModel[0, 1] = TransferFunction.TauForm(0.29,  1.5, new Vector(2.0, 38.0)
    , new Vector(8.0));
CMModel[1, 1] = TransferFunction.TauForm(5.00,  0.1, new Vector(14.0, 1.0));



// generate state space models of Cement Mill
StateSpaceModel Plant = new StateSpaceModel(CMModel, T);
StateSpaceModel Model = new StateSpaceModel(CMModel, T, alfa);
```

"Plant" used to simulate cement mill
"Model" Extended Delta ARX model for MPC Controller

$$G(s) = \begin{bmatrix} \frac{0.62}{(45s+1)(8s+1)}e^{-5s} & \frac{0.29(8s+1)}{(2s+1)(38s+1)}e^{-1.5s} \\ \frac{(-15)}{(60s+1)}e^{-5s} & \frac{5}{(14s+1)(s+1)}e^{-0.1s} \end{bmatrix}$$

“Control Loop”

```
// Setup MiMo MPC
MiMoMPC mpc = new MiMoMPC(Model, history, horizon,
    theta, my, ymin, ymax, rho, umin, umax);


// MPC control
for (int step = 0; step < steps; step++)
{
    // Manipulate setpoints
    YRef[1] = SetVal(step, 0, 100, 100.0); // Fineness setpoint
    YRef[0] = SetVal(step, 150, 250, 10.0); // elevator load setpoint
    mpc.SetRef(YRef);

    // Measurement and process noise
    eps = MPCMathLib.WhiteGaussianNoise(epsilonVar, epsilonMean);
    xi = MPCMathLib.WhiteGaussianNoise(xiVar, xiMean);

    // Plant measurement with measurement noise
    YPlant = Plant.Observed(XPlant, eps);

    // MPC constroller
    UC = mpc.Next(YPlant);

    // send control signal and process noise to plant
    XPlant = Plant.NextState(XPlant, UC, xi);
}
```

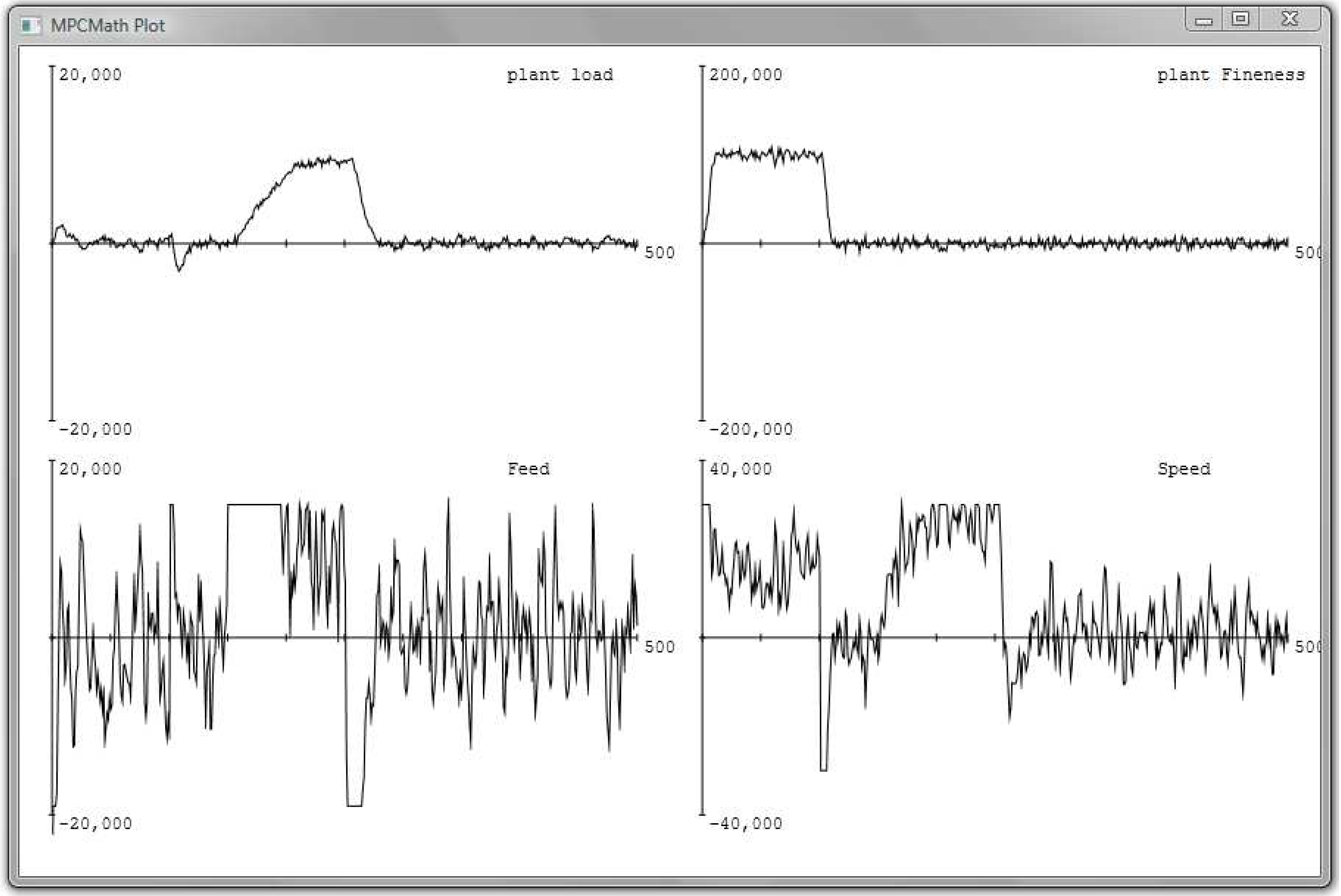
Define MPC controller

Read noisy Plant Controlled vars

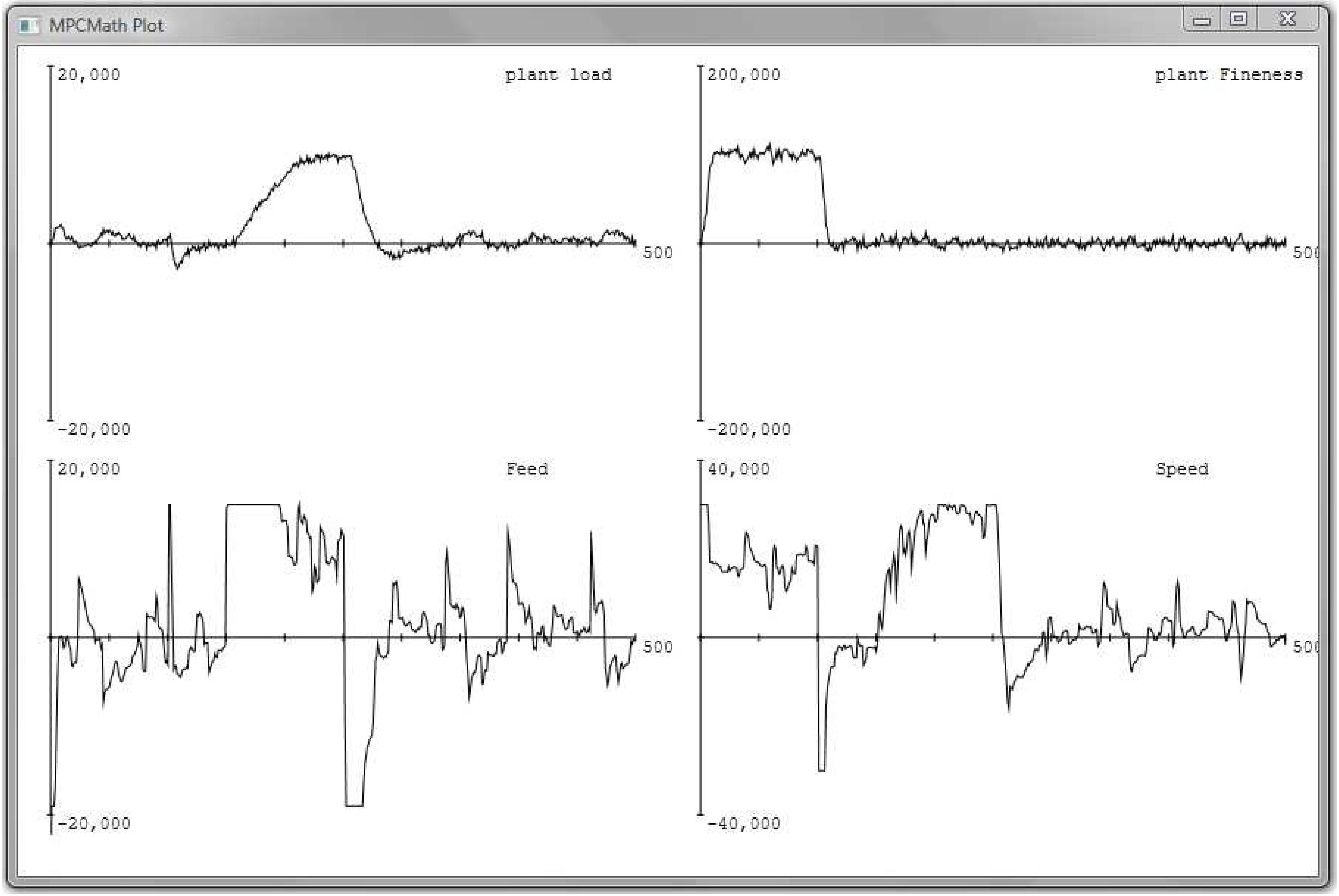
Calculate next action

Simulate Cement Mill

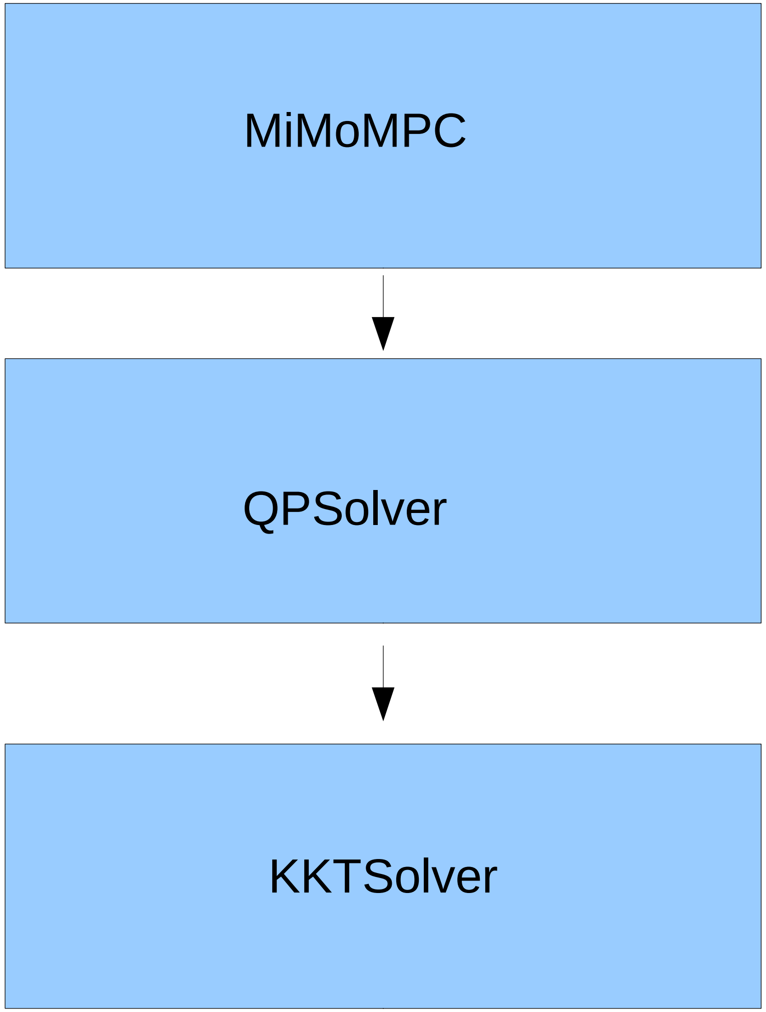
Conventional MPC



SoftConstrained MPC



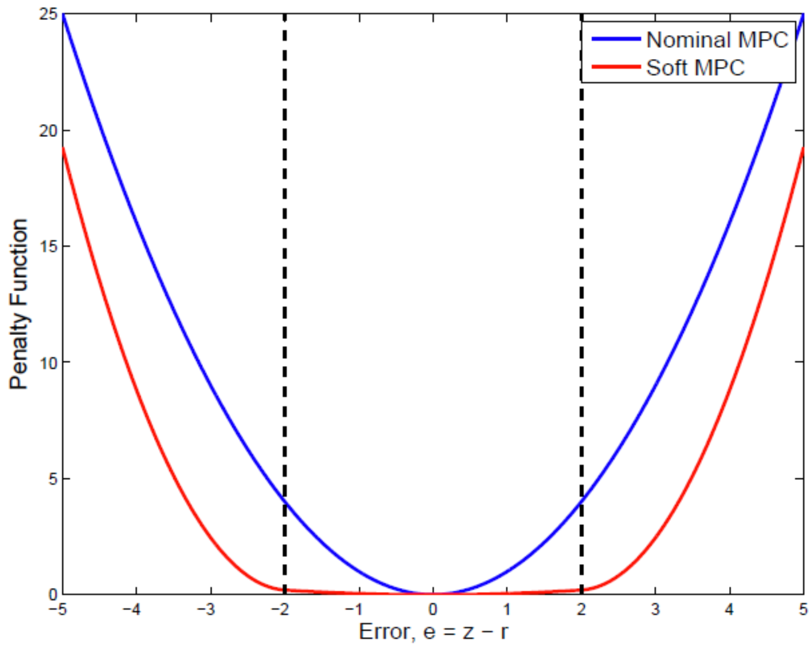
MPC controller layers



- Set up user MPC and call standard QPSolver
  - End user parameters
    - weigths and limits
- Primal-Dual Interior-Point Algorithm
  - Convex optimization
    - Linear, quadratic, conic problems
- KKT solver
  - Defines optimization type
  - CPU demanding calculations
  - Optimized for special case
    - Exploit matrix structures

```
// Setup MiMo MPC Constructor
MiMoMPC mpc = new MiMoMPC(Model, history, horizon,
           theta, my, ymin, ymax, rho, umin, umax);

// MPC constroller
UC = mpc.Next(YPlant);
```

- MiMoMPC object
  - How should the end user see the MPC controller
  - Set up tables for QPSolver
  - Data for operator plots



QPSolver

$$\min_{x \in R^n} \; \tfrac{1}{2} x'Gx + g'x$$

$$\text{s.t.} \quad Ax = b$$
$$Cx >= d$$

```
// Conventional MPC solver
solver = new QPSolver(G, g, A, b, C, d);
```

```
// SoftConstrained MPC Solver
QPSolver solver = new QPSolver(
new SoftConstraintKKTSolver(G, g, A, b, C, d));
```

```
// Solve QP problem
solver.Solve();
Vector ur = solver.X;
```

KKTSolver

$$\begin{vmatrix} G & -A' & -C' & 0 \\ -A & 0 & 0 & 0 \\ -C & 0 & 0 & I \\ 0 & 0 & S & Z \end{vmatrix} \begin{vmatrix} \Delta X \\ \Delta Y \\ \Delta Z \\ \Delta S \end{vmatrix} = - \begin{vmatrix} r_l \\ r_A \\ r_C \\ r_{SZ} \end{vmatrix} = - \begin{vmatrix} Gx + g - A'x - Cz \\ -Ax + b \\ -Cx + s + d \\ -SZe \end{vmatrix}$$

```
// Calculate residuals
solver.Residuals(x, y, z, s,
                 out rL, out rA, out rC, out rSZ);

// Factorize
solver.Factorize(SInvZ);


// find newton direction
solver.Solve(rL, rA, rCM,
             out deltaX, out deltaY,out deltaZ);
```

Speed issues

- 99% of CPU load is spend in the KKTSolver
  - The solver.Factorize routine
- Critical operations are
  - Matrix multiplications and Matrix factorizations
  - The first version of MPCMath was simply too slow !
- Solutions
  - Raw computing power using a high performance computing package
  - Exploit Matrix structures

Test Speed

- Test matrix multiplication and Factorization speeds

```
int dim = 32;
for (int test = 0; test < 7; test++)
{

    dim = 2 * dim;
    Matrix A = Matrix.Random(dim, dim, -100.0, 100.0);
    Matrix AT = Matrix.Transpose(A);
    Vector r = Vector.Random(dim, -1000.0, 1000.0);

    DateTime start = DateTime.Now;

    // Matrix multiplication
    A = AT * A;

    TimeSpan tm = DateTime.Now - start;

    // Cholesky factorization + solve
    start = DateTime.Now;
    CholeskyEquationSolver solver = new CholeskyEquationSolver(A);
    Vector x = solver.Solve(r);

    TimeSpan tc = DateTime.Now - start;

    console.WriteLine("dim " + dim.ToString() + "   matrix "
        + tm.TotalMilliseconds.ToString() +
        "   Cholesky " + tc.TotalMilliseconds.ToString() + " ms");
}
```

Speed improvement (Brute force)

- Intel Math Kernel Libray (BLAS & LAPACK)

Matrix multiplication time (msec)			
Dimension	C#	MKL	faktor
128	40	2	20,0
256	301	8	37,6
512	2.962	63	47,0
1024	29.266	456	64,2
2048	250.354	3.622	69,1
4096	2.177.916	28.294	77,0

Cholesky Factorization (msec)			
Dimension	C#	MKL	faktor
128	9	1	9,0
256	69	3	23,0
512	554	21	26,4
1024	4.782	113	42,3
2048	41.892	773	54,2
4096	375.996	5.786	65,0

- Intel MKL library is 50 to 60 times faster than the best possible C# code !

Speed, (Structure exploitation)

- Exploitation of Matrix structures
  - A very common operation:
    - Res = 0.0 * a + 0.0
- Block Matrices with matrix elements
  - Store information about matrix form in matrix object
    - Null, Constant Diagonal, Diagonal, General form
    - MPCMath exploits this information automatically

KKT Factorize

The critical operations are calculation of $C'(S^{-1}Z)C$ and LDL' factorization in

$$\begin{vmatrix} G + C'(S^{-1}Z)C & -A' \\ -A & 0 \end{vmatrix}$$

C Structure
Standard MPC

$$C = \begin{vmatrix} I \\ -I \end{vmatrix}$$

C Structure
SoftConstrained MPC

$$C = \begin{vmatrix} I & 0 \\ R & I \\ -I & 0 \\ -R & I \\ 0 & I \end{vmatrix}$$

Where:

0 Null matrix
I Identity matrix
R Lower block triangular

All matrices with dimension 100x100
(Horizon = 50)

This Null and Diagonal structures are automatically exploited,
the Lower triangular form requires a dedicated KKTSolver routine

Conclusions

- Intelligent objects written in C# enables the designer to implement MPC and other optimization tasks in the C#/.NET environment

Conclusions

- Intelligent objects written in C# enables the designer to implement MPC and other optimization tasks in the C#/.NET environment
- High Performance Computing libraries are necessary for fast linear algebra computations

Conclusions

- Intelligent objects written in C# enables the designer to implement MPC and other optimization tasks in the C#/.NET environment
- High Performance Computing libraries are necessary for fast linear algebra computations.
- Matrix structures should be exploited to increase speed

Conclusions

- Intelligent objects written in C# enables the designer to implement MPC and other optimization tasks in the C#/.NET environment
- High Performance Computing libraries are necessary for fast linear algebra computations.
- Matrix structures should be exploited to increase speed
- Future work
  - Identification sw for data driven Model development
- MPCMath documentation, sw and demo licenses can be retrieved from www.2-control.dk