Model Predictive Control

Solving Nonlinear Model Predictive Control (NMPC) Problems

Colin Jones

Laboratoire d'Automatique

NMPC Theory - Quick and Dirty

Nonlinear Model Predictive Control - NMPC

$$
\begin{aligned}
u^{\star}(x_0) = \operatorname{argmin} \; & \sum_{i=0}^{N-1} l(x_i, u_i) + V_f(x_N) \\
\text{s.t.} \quad & x_{i+1} = f(x_i, u_i) \quad \forall i = 0, \ldots, N-1 \\
& g(x_i, u_i) \leq 0 \qquad\;\; \forall i = 0, \ldots, N-1 \\
& h(x_N) \leq 0
\end{aligned}
$$

where f, g and h are continuous.

Theory is the same as linear MPC

Feasibility Same assumptions on terminal constraint

Stability Same assumptions on stage cost and terminal cost

Nonlinear Model Predictive Control - NMPC

$$
\begin{aligned}
u^\star(x_0) = \text{argmin} \ \sum_{i=0}^{N-1} l(x_i, u_i) + V_f(x_N) & \\
\text{s.t.} \qquad x_{i+1} = f(x_i, u_i) & \quad \forall i = 0, \ldots, N-1 \\
g(x_i, u_i) \leq 0 & \quad \forall i = 0, \ldots, N-1 \\
h(x_N) \leq 0 &
\end{aligned}
$$

where f, g and h are continuous.

Theory is the same as linear MPC

Feasibility Same assumptions on terminal constraint

Stability Same assumptions on stage cost and terminal cost

What is much harder

Invariance Sets are harder to calculate... so we often drop terminal constraints (or take $x_N = 0$)

Optimality May only obtain a local minimum, or there may be multiple optimal solutions. This leads to many difficulties.

Today: Forming and Solving NMPC Problems

$$
\begin{aligned}
\min \quad & \sum_{i=0}^{N-1} l(x_i, u_i) + V_f(x_N) \\
\text{s.t.} \quad & x_{i+1} = f(x_i, u_i) \quad \forall i = 0, \dots, N-1 \\
& g(x_i, u_i) \leq 0 \quad \forall i = 0, \dots, N-1 \\
& h(x_N) \leq 0
\end{aligned}
\quad \Rightarrow \quad
\begin{aligned}
\min \; & F(z) \\
\text{s.t.} \; & G(z) \leq 0 \\
& H(z) = 0
\end{aligned}
$$

Two challenges:

Discretization The world is continuous - where do we get f from?

Gradients Optimization is based on gradients - how to compute?

Nonlinear Programming

Recall: Descent Methods

$$\min \ f(z)$$

$$z^{(k+1)} = z^{(k)} + t^{(k)} \Delta z^{(k)} \quad \text{with } f(z^{(k+1)}) < f(z^{(k)})$$

- Δz is the **step** or **search direction**
- t is the **step size** or **step length**
- $f(z^{(k+1)}) < f(z^{(k)})$, i.e., Δz is a **descent direction**
- There exists a $t > 0$ such that $f(z^{(k+1)}) < f(z^{(k)})$ if $\nabla f(z)^T \Delta z < 0$

Newton's Method

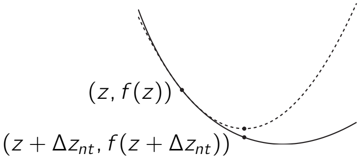
$$\Delta z_{nt} = -\nabla^2 f(z)^{-1} \nabla f(z)$$

- Interpretation: $z + \Delta z_{nt}$ minimizes second order approximation

$$\hat{f}(z+v) = f(z) + \nabla f(z)^T v + \frac{1}{2} v^T \nabla^2 f(z) v$$

Optimality condition: $\nabla \hat{f}(z + v^*) = 0$

$$\nabla f(z) + \nabla^2 f(z) v^* = 0$$
$$\Rightarrow \nabla^2 f(z) v^* = -\nabla f(z)$$



- Decent direction:

$$\nabla f(z)^T \Delta z_{nt} = -\nabla f(z)^T \nabla^2 f(z)^{-1} \nabla f(z) < 0$$

f convex implies that $\nabla^2 f(z) \succeq 0$

- If z is close to optimum, $\|\nabla f(z)\|_2$ converges to zero quadratically (**extremely** quickly)

Gauss-Newton Method for NLPs

$$\begin{aligned} \min\ & F(z) = \|R(z)\|^2 \\ \text{s.t.}\ & G(z) \leq 0 \\ & H(z) = 0 \end{aligned}$$

[1]Note that this is a little more complex than in the convex case

Gauss-Newton Method for NLPs

$$
\begin{aligned}
\min \; & F(z) = \|R(z)\|^2 \\
\text{s.t. } & G(z) \leq 0 \\
& H(z) = 0
\end{aligned}
$$

Compute quadratic approximation

$$
\begin{aligned}
\min \; & \|R(z^k) + \nabla R(z^k)^T \Delta z\|^2 \\
\text{s.t. } & G(z^k) + \nabla G(z^k)^T \Delta z \leq 0 \\
& H(z^k) + \nabla H(z^k)^T \Delta z = 0
\end{aligned}
$$

We solve this **quadratic program** to get the search direction Δz, and then compute a step size via line search[1]

[1] Note that this is a little more complex than in the convex case

Newton's Method for NLPs - Sequential Quadratic Programming

This can also be done for general NLPs

$$\begin{aligned} \min\ & F(z) \\ \text{s.t. } & G(z) \leq 0 \\ & H(z) = 0 \end{aligned}$$

Compute quadratic approximation

$$\begin{aligned} \min\ & \nabla F(z^k)^T \Delta z + \frac{1}{2} \Delta z^T A^k \Delta z \\ \text{s.t. } & G(z^k) + \nabla G(z^k)^T \Delta z \leq 0 \\ & H(z^k) + \nabla H(z^k)^T \Delta z = 0 \end{aligned}$$

where A^k is the Hessian of the Lagrangian function.

Dual optimal solution of the QP also gives a search direction for the dual variables.

Many Methods to solve NLPs...

Interior-point Form the KKT optimality conditions and apply Newton's method to the set of equations.

Sequential Quadratic Programming Linearize the KKT conditions and solve. Equivalent to (sort of) computing quadratic approximation of the original problem repeatedly.

Operator Splitting methods Divide the optimization problem into the sum of two "simple" parts

$$\min \; f(x) + g(z) \text{ s.t. } x = z$$

Solve by alternating between minimizing f and minimizing g. Useful when solving f and g alone is very easy.

All of these methods require **gradient** calculations!

Discretization

Discretization of Nonlinear Systems

The world is continuous

$$\dot{x} = f(x, u)$$

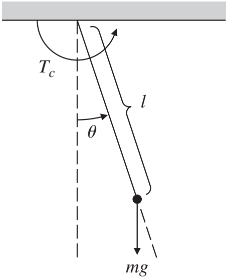
How do we discretize?

$$x_{k+1} = \hat{f}(x_k, u_k)$$

For linear systems, this is easy and closed-form

For nonlinear is has to be done online

Example: Pendulum



Moment of inertia wrt. rotational axis: $m\, l^2$
Torque caused by external force: T_c
Torque caused by gravity: $m\, g\, l \sin(\theta)$

System equation: $m\, l^2\, \ddot{\theta} = T_c - m\, g\, l \sin(\theta)$

Using $x_1 := \theta, x_2 := \dot{\theta} = \dot{x}_1$, $u := T_c / m\, l^2$ and $g/l = 10$

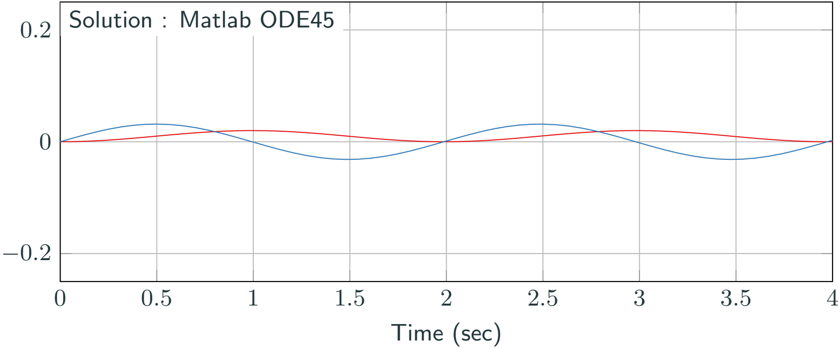
$$\dot{x} = \begin{pmatrix} \dot{x}_1 \\ \dot{x}_2 \end{pmatrix} = \begin{pmatrix} x_2 \\ -\frac{g}{l} \sin(x_1) + u \end{pmatrix} = \begin{pmatrix} x_2 \\ -10 \sin(x_1) + u \end{pmatrix} = f(x, u)$$

Try the most obvious thing - Euler approximation

$$x^+ = x + hf(x,u)$$

where h is the sample period.

$$x_{k+1} = x_k + h \begin{bmatrix} x_{2,k} \\ -10\sin(x_{1,k}) + u_k \end{bmatrix}$$



Orange: Velocity, Blue: Angle

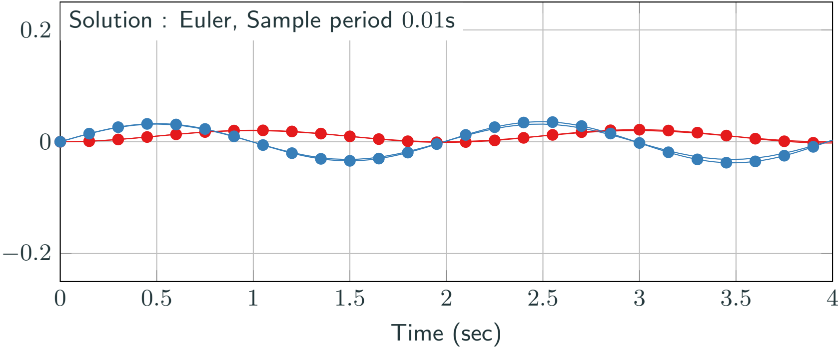
Integration - The Simple Way

Try the most obvious thing - Euler approximation

$$x^+ = x + hf(x, u)$$

where h is the sample period.

$$x_{k+1} = x_k + h \begin{bmatrix} x_{2,k} \\ -10\sin(x_{1,k}) + u_k \end{bmatrix}$$



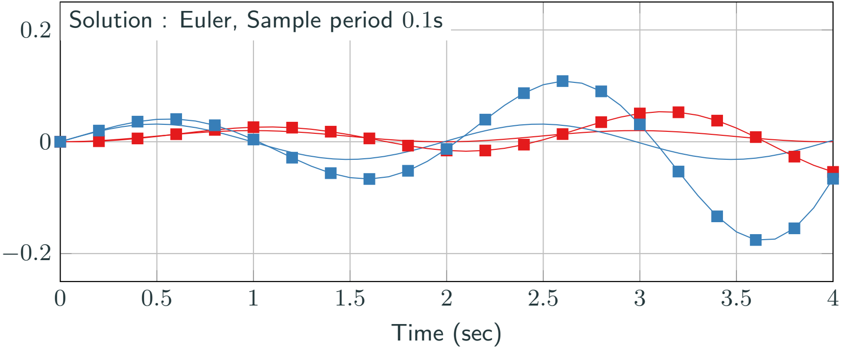
Orange: Velocity, Blue: Angle

Try the most obvious thing - Euler approximation

$$x^+ = x + hf(x, u)$$

where h is the sample period.

$$x_{k+1} = x_k + h \begin{bmatrix} x_{2,k} \\ -10\sin(x_{1,k}) + u_k \end{bmatrix}$$



Orange: Velocity, Blue: Angle

Two Methods of Integration

1. Direct integration
   - Use a integration algorithm to compute $x(k+1) = x(k) + \int_{t=T_s k}^{T_s(k+1)} f(x,u)dt$
2. Collocation
   - Define the trajectory in terms of basis functions

$$x(t) = \sum_{i=0}^{q} w_i \beta_i(t)$$

   - Enforce that the dynamic equations are met at the **collocation points**

$$\dot{x}(t_k) = f(x_k, u_k) = \sum_{i=0}^{q} w_i \dot{\beta}_i(t_k)$$

Runge-Kutta - The Basic Idea

Consider the ODE

$$\dot{x} = f(x)$$

Given $x = x(t)$, we want to compute $x^+ = x(t+h)$

Consider the ODE

$$\dot{x} = f(x)$$

Given $x = x(t)$, we want to compute $x^+ = x(t+h)$

Compute a second-order Taylor series expansion

$$x^+ = x + h\dot{x} + \frac{h^2}{2}\ddot{x} + \mathcal{O}(h^3)$$

Runge-Kutta - The Basic Idea

Consider the ODE

$$\dot{x} = f(x)$$

Given $x = x(t)$, we want to compute $x^+ = x(t+h)$

Compute a second-order Taylor series expansion

$$x^+ = x + h\dot{x} + \frac{h^2}{2}\ddot{x} + \mathcal{O}(h^3)$$

Take Jacobian of f to compute $\ddot{x}$

$$\ddot{x} = J_f(x)\dot{x} = J_f(x)f(x)$$

Runge-Kutta - The Basic Idea

Consider the ODE

$$\dot{x} = f(x)$$

Given $x = x(t)$, we want to compute $x^+ = x(t+h)$

Compute a second-order Taylor series expansion

$$x^+ = x + h\dot{x} + \frac{h^2}{2}\ddot{x} + \mathcal{O}(h^3)$$

Take Jacobian of f to compute $\ddot{x}$

$$\ddot{x} = J_f(x)\dot{x} = J_f(x)f(x)$$

The Taylor series expansion is now

$$\begin{aligned} x^+ &= x + hf(x) + \frac{h^2}{2}J_f(x)f(x) + \mathcal{O}(h^3) \\ &= x + \frac{h}{2}f(x) + \frac{h}{2}\left(f(x) + hJ_f(x)f(x)\right) + \mathcal{O}(h^3) \end{aligned}$$

Runge-Kutta - The Basic Idea

The Taylor series expansion is now

$$x^+ = x + \frac{h}{2} f(x) + \frac{h}{2} \left(f(x) + h J_f(x) f(x) \right) + \mathcal{O}(h^3)$$

Consider the Taylor series expansion of the expression

$$f(x + h f(x)) = f(x) + h J_f(x) f(x) + \mathcal{O}(h^2)$$

Therefore, we get

$$\begin{aligned} x^+ &\approx x + \frac{h}{2} f(x) + \frac{h}{2} f(x + h f(x)) \\ &= x + h \left(\frac{1}{2} k_1 + \frac{1}{2} k_2 \right) \end{aligned}$$

where

$$\begin{aligned} k_1 &= f(x) \\ k_2 &= f(x + h k_1) \end{aligned}$$

Runge-Kutta 4 - The Most Common Version

Consider the time dependent ODE

$$\dot{x} = f(t, x)$$

$$x_{k+1} = x_k + h\left(\frac{k_1}{6} + \frac{k_2}{3} + \frac{k_3}{3} + \frac{k_4}{6}\right)$$

where

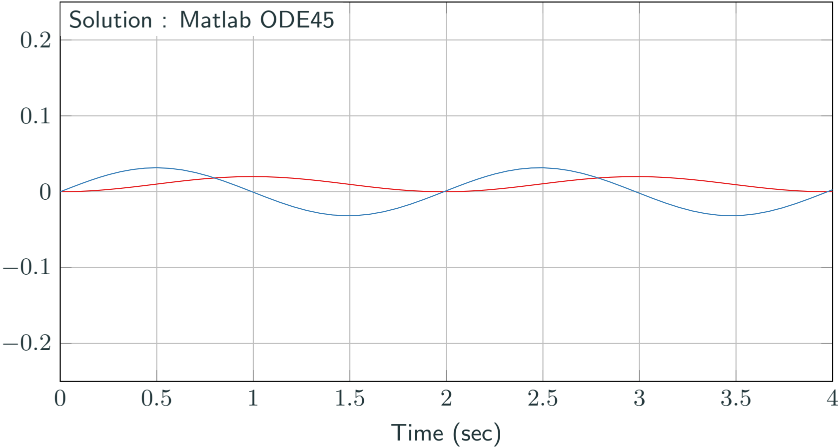
$$\begin{aligned}
k_1 &= f(t_k, x_k) \\
k_2 &= f(t_k + \frac{h}{2}, x_k + \frac{h}{2}k_1) \\
k_3 &= f(t_k + \frac{h}{2}, x_k + \frac{h}{2}k_2) \\
k_4 &= f(t_k + h, x_k + hk_3)
\end{aligned}$$

Note: There are **many** more ways to integrate, and different methods are appropriate depending on the properties of your system, and requirements of the optimization.

Example: Discretization of the pendulum

Pendulum equations are given by

$$\dot{x} = \begin{bmatrix} x_2 \\ -10\sin(x_1) + u \end{bmatrix}$$

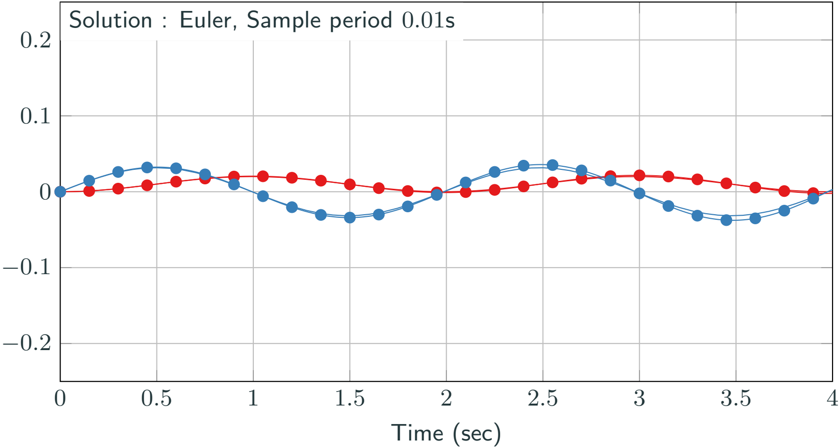


Orange: Velocity, Blue: Angle

Example: Discretization of the pendulum

Pendulum equations are given by

$$\dot{x} = \begin{bmatrix} x_2 \\ -10\sin(x_1) + u \end{bmatrix}$$

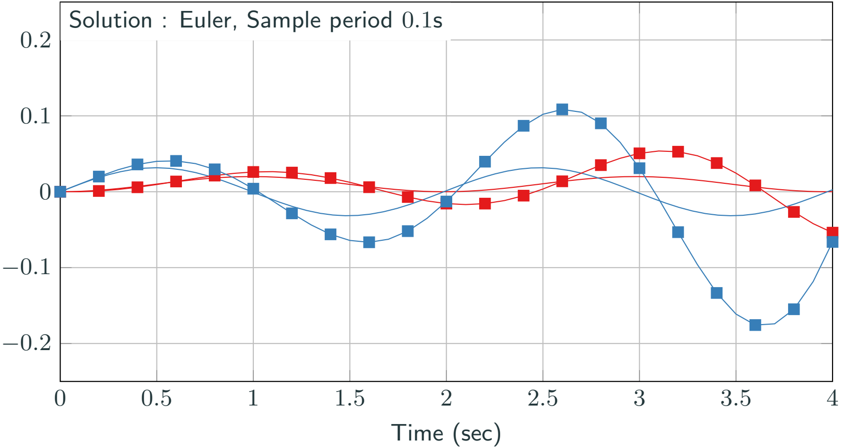


Orange: Velocity, Blue: Angle

Example: Discretization of the pendulum

Pendulum equations are given by

$$\dot{x} = \begin{bmatrix} x_2 \\ -10\sin(x_1) + u \end{bmatrix}$$

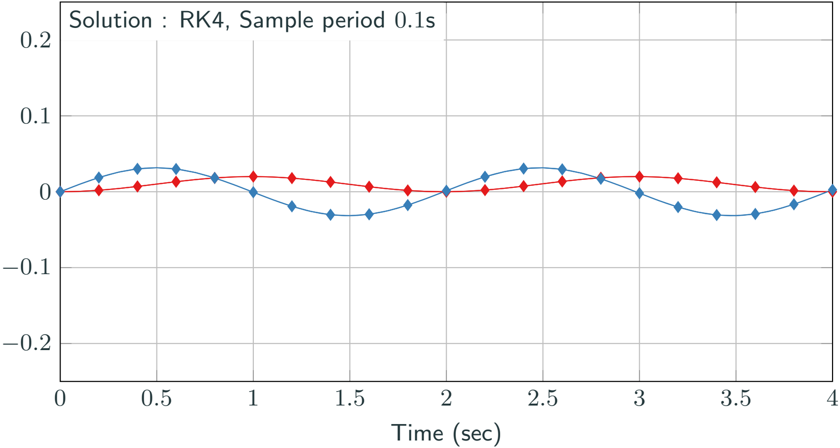


Orange: Velocity, Blue: Angle

Example: Discretization of the pendulum

Pendulum equations are given by

$$\dot{x} = \begin{bmatrix} x_2 \\ -10\sin(x_1) + u \end{bmatrix}$$



Orange: Velocity, Blue: Angle

Two Methods of Integration

1. Direct integration
   - Use a integration algorithm to compute $x(k+1) = x(k) + \int_{t=T_s k}^{T_s(k+1)} f(x,u)dt$
2. Collocation
   - Define the trajectory in terms of basis functions

$$x(t) = \sum_{i=0}^{q} w_i \beta_i(t)$$

   - Enforce that the dynamic equations are met at the **collocation points**

$$\dot{x}(t_k) = f(x_k, u_k) = \sum_{i=0}^{q} w_i \dot{\beta}_i(t_k)$$

Polynomial Interpolation

Given the state $x_k = x(t_k)$ and the constant input $u(t) = u_k$ we want to compute the state $x_{k+1} = x(t_{k+1})$ for the system $\dot{x} = f(x, u)$.

Polynomial Interpolation

Given the state $x_k = x(t_k)$ and the constant input $u(t) = u_k$ we want to compute the state $x_{k+1} = x(t_{k+1})$ for the system $\dot{x} = f(x, u)$.

Time grid

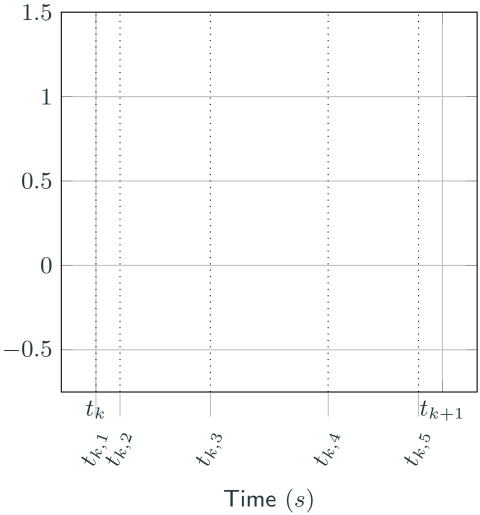
$$\{t_{k,0}, \dots, t_{k,K}\} \in [t_k, t_{k+1}]$$

Lagrange Polynomials

$$P_{k,i}(t) = \prod_{j=0, j\neq i}^{K} \frac{t - t_{k,j}}{t_{k,i} - t_{k,j}} \in \mathbb{R}$$

We have the property

$$P_{k,i}(t_{k,l}) = \begin{cases} 1 & \text{if } l = i \\ 0 & \text{if } l \neq i \end{cases}$$



Polynomial Interpolation

Given the state $x_k = x(t_k)$ and the constant input $u(t) = u_k$ we want to compute the state $x_{k+1} = x(t_{k+1})$ for the system $\dot{x} = f(x, u)$.

Time grid

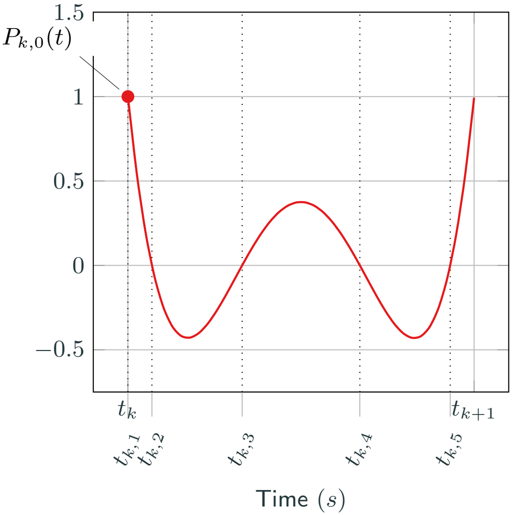
$$\{t_{k,0}, \dots, t_{k,K}\} \in [t_k, t_{k+1}]$$

Lagrange Polynomials

$$P_{k,i}(t) = \prod_{j=0, j\neq i}^{K} \frac{t - t_{k,j}}{t_{k,i} - t_{k,j}} \in \mathbb{R}$$

We have the property

$$P_{k,i}(t_{k,l}) = \begin{cases} 1 & \text{if } l = i \\ 0 & \text{if } l \neq i \end{cases}$$



Polynomial Interpolation

Given the state $x_k = x(t_k)$ and the constant input $u(t) = u_k$ we want to compute the state $x_{k+1} = x(t_{k+1})$ for the system $\dot{x} = f(x, u)$.

Time grid

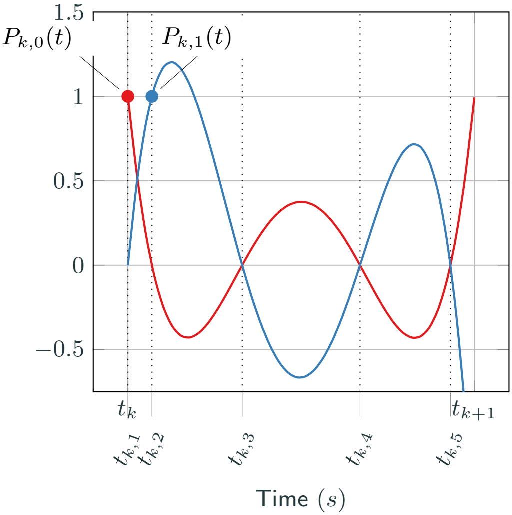
$$\{t_{k,0}, \dots, t_{k,K}\} \in [t_k, t_{k+1}]$$

Lagrange Polynomials

$$P_{k,i}(t) = \prod_{j=0, j\neq i}^{K} \frac{t - t_{k,j}}{t_{k,i} - t_{k,j}} \in \mathbb{R}$$

We have the property

$$P_{k,i}(t_{k,l}) = \begin{cases} 1 & \text{if } l = i \\ 0 & \text{if } l \neq i \end{cases}$$



Polynomial Interpolation

Given the state $x_k = x(t_k)$ and the constant input $u(t) = u_k$ we want to compute the state $x_{k+1} = x(t_{k+1})$ for the system $\dot{x} = f(x, u)$.

Time grid

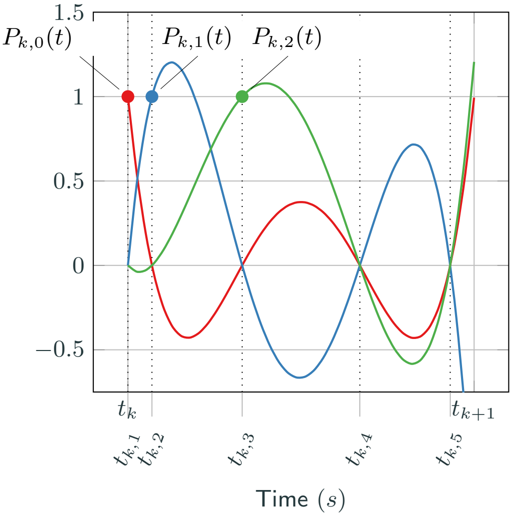
$$\{t_{k,0}, \dots, t_{k,K}\} \in [t_k, t_{k+1}]$$

Lagrange Polynomials

$$P_{k,i}(t) = \prod_{j=0, j\neq i}^{K} \frac{t - t_{k,j}}{t_{k,i} - t_{k,j}} \in \mathbb{R}$$

We have the property

$$P_{k,i}(t_{k,l}) = \begin{cases} 1 & \text{if } l = i \\ 0 & \text{if } l \neq i \end{cases}$$



Polynomial Interpolation

Given the state $x_k = x(t_k)$ and the constant input $u(t) = u_k$ we want to compute the state $x_{k+1} = x(t_{k+1})$ for the system $\dot{x} = f(x, u)$.

Time grid

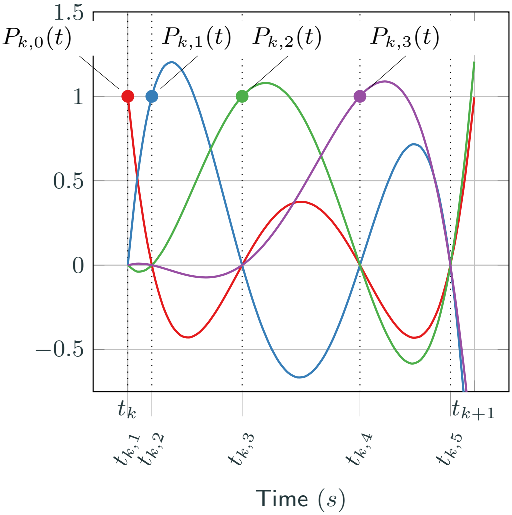
$$\{t_{k,0}, \ldots, t_{k,K}\} \in [t_k, t_{k+1}]$$

Lagrange Polynomials

$$P_{k,i}(t) = \prod_{j=0, j\neq i}^{K} \frac{t - t_{k,j}}{t_{k,i} - t_{k,j}} \in \mathbb{R}$$

We have the property

$$P_{k,i}(t_{k,l}) = \begin{cases} 1 & \text{if } l = i \\ 0 & \text{if } l \neq i \end{cases}$$



Polynomial Interpolation

Given the state $x_k = x(t_k)$ and the constant input $u(t) = u_k$ we want to compute the state $x_{k+1} = x(t_{k+1})$ for the system $\dot{x} = f(x, u)$.

Time grid

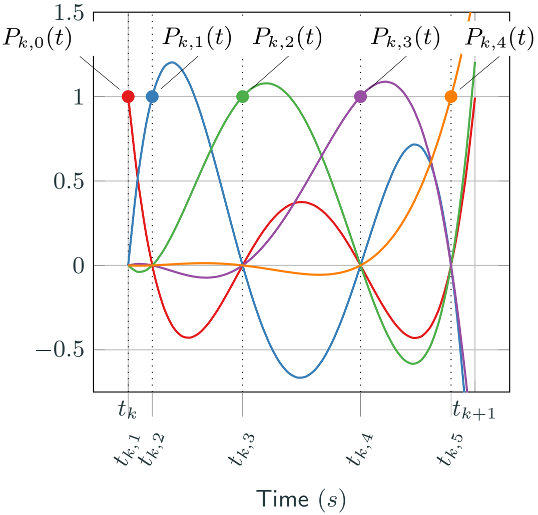
$$\{t_{k,0}, \dots, t_{k,K}\} \in [t_k, t_{k+1}]$$

Lagrange Polynomials

$$P_{k,i}(t) = \prod_{j=0, j \neq i}^{K} \frac{t - t_{k,j}}{t_{k,i} - t_{k,j}} \in \mathbb{R}$$

We have the property

$$P_{k,i}(t_{k,l}) = \begin{cases} 1 & \text{if } l = i \\ 0 & \text{if } l \neq i \end{cases}$$

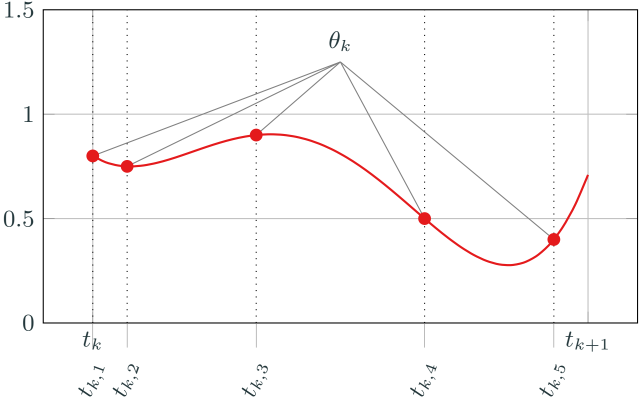


Polynomial Interpolation

Define the interpolating function

$$x(\theta_k, t) = \sum_{i=0}^{K} \underbrace{\theta_{k,i}}_{\text{parameters}} \cdot \underbrace{P_{k,i}(t)}_{\text{polynomials}}$$

Where we note that $x(\theta_k, t_{k,j}) = \theta_{k,j}$

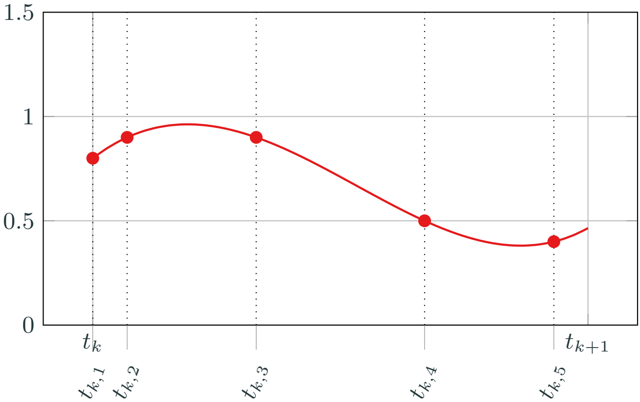


Polynomial Interpolation

Define the interpolating function

$$x(\theta_k, t) = \sum_{i=0}^{K} \underbrace{\theta_{k,i}}_{\text{parameters}} \cdot \underbrace{P_{k,i}(t)}_{\text{polynomials}}$$

Where we note that $x(\theta_k, t_{k,j}) = \theta_{k,j}$

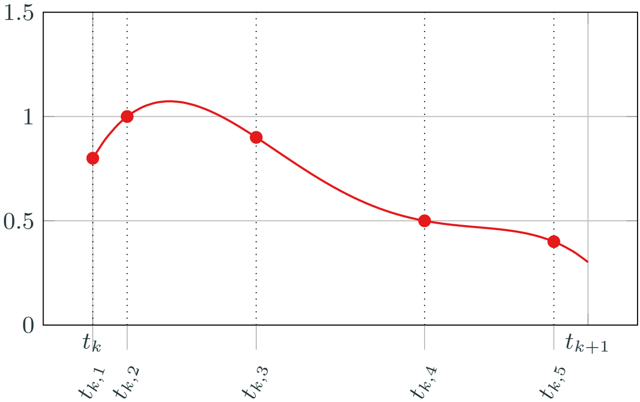


Polynomial Interpolation

Define the interpolating function

$$x(\theta_k, t) = \sum_{i=0}^{K} \underbrace{\theta_{k,i}}_{\text{parameters}} \cdot \underbrace{P_{k,i}(t)}_{\text{polynomials}}$$

Where we note that $x(\theta_k, t_{k,j}) = \theta_{k,j}$

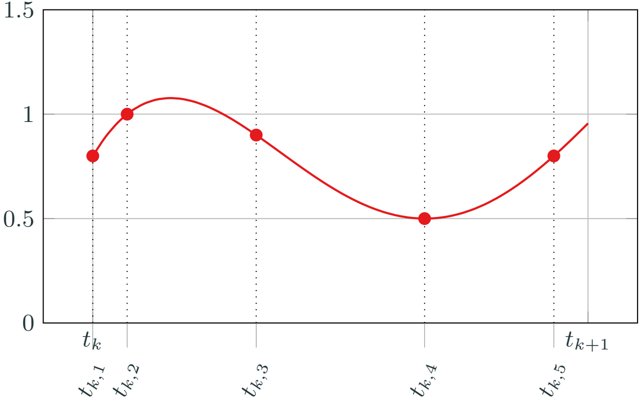


Polynomial Interpolation

Define the interpolating function

$$x(\theta_k, t) = \sum_{i=0}^{K} \underbrace{\theta_{k,i}}_{\text{parameters}} \cdot \underbrace{P_{k,i}(t)}_{\text{polynomials}}$$

Where we note that $x(\theta_k, t_{k,j}) = \theta_{k,j}$



Collocation

What we have:

- State x_k at time t_k
- Constant input $u(t) = u_k$ over the time interval $[t_k, t_{k+1}]$
- Gradient of the state trajectory $\dot{x} = f(x, u)$

Collocation

What we have:

- State x_k at time t_k
- Constant input $u(t) = u_k$ over the time interval $[t_k, t_{k+1}]$
- Gradient of the state trajectory $\dot{x} = f(x, u)$

Define a set of grid points $\{t_{k,0}, \ldots, t_{k,K}\} \in [t_k, t_{k+1}]$ and a polynomial interpolation

$$x(\theta_k, t) := \sum_{j=0}^{K} \theta_{k,j} P_{k,j}(t) \qquad\qquad x(\theta_k, t_{k,j}) = \theta_{k,j}$$

Collocation

What we have:

- State x_k at time t_k
- Constant input $u(t) = u_k$ over the time interval $[t_k, t_{k+1}]$
- Gradient of the state trajectory $\dot{x} = f(x, u)$

Define a set of grid points $\{t_{k,0}, \ldots, t_{k,K}\} \in [t_k, t_{k+1}]$ and a polynomial interpolation

$$x(\theta_k, t) := \sum_{j=0}^{K} \theta_{k,j} P_{k,j}(t) \qquad\qquad x(\theta_k, t_{k,j}) = \theta_{k,j}$$

Enforce the dynamics at the **collocation points**

$$x(\theta_k, t_k) = x_k \qquad \text{Initial condition}$$

$$\frac{\partial}{\partial t} x(\theta_k, t_{k,j}) = f(x(\theta_k, t_{k,j}), u_k) \qquad \text{Dynamics}$$

Collocation

What we have:

- State x_k at time t_k
- Constant input $u(t) = u_k$ over the time interval $[t_k, t_{k+1}]$
- Gradient of the state trajectory $\dot{x} = f(x, u)$

Define a set of grid points $\{t_{k,0}, \ldots, t_{k,K}\} \in [t_k, t_{k+1}]$ and a polynomial interpolation

$$x(\theta_k, t) := \sum_{j=0}^{K} \theta_{k,j} P_{k,j}(t) \qquad\qquad x(\theta_k, t_{k,j}) = \theta_{k,j}$$

Enforce the dynamics at the **collocation points**

$$\theta_{k,0} = x_k \qquad \text{Initial condition}$$

$$\frac{\partial}{\partial t} x(\theta_k, t_{k,j}) = f(x(\theta_k, t_{k,j}), u_k) \qquad \text{Dynamics}$$

Collocation

What we have:

- State x_k at time t_k
- Constant input $u(t) = u_k$ over the time interval $[t_k, t_{k+1}]$
- Gradient of the state trajectory $\dot{x} = f(x, u)$

Define a set of grid points $\{t_{k,0}, \ldots, t_{k,K}\} \in [t_k, t_{k+1}]$ and a polynomial interpolation

$$x(\theta_k, t) := \sum_{j=0}^{K} \theta_{k,j} P_{k,j}(t) \qquad\qquad x(\theta_k, t_{k,j}) = \theta_{k,j}$$

Enforce the dynamics at the **collocation points**

$$\theta_{k,0} = x_k \qquad \text{Initial condition}$$

$$\frac{\partial}{\partial t} \sum_{j=0}^{K} \theta_{k,j} P_{k,j}(t_{k,j}) = f(x(\theta_k, t_{k,j}), u_k) \qquad \text{Dynamics}$$

Collocation

What we have:

- State x_k at time t_k
- Constant input $u(t) = u_k$ over the time interval $[t_k, t_{k+1}]$
- Gradient of the state trajectory $\dot{x} = f(x, u)$

Define a set of grid points $\{t_{k,0}, \dots, t_{k,K}\} \in [t_k, t_{k+1}]$ and a polynomial interpolation

$$x(\theta_k, t) := \sum_{j=0}^{K} \theta_{k,j} P_{k,j}(t) \qquad\qquad x(\theta_k, t_{k,j}) = \theta_{k,j}$$

Enforce the dynamics at the **collocation points**

$$\theta_{k,0} = x_k \qquad \text{Initial condition}$$

$$\sum_{j=0}^{K} \theta_{k,j} \dot{P}_{k,j}(t_{k,j}) = f(x(\theta_k, t_{k,j}), u_k) \qquad \text{Dynamics}$$

Collocation

What we have:

- State x_k at time t_k
- Constant input $u(t) = u_k$ over the time interval $[t_k, t_{k+1}]$
- Gradient of the state trajectory $\dot{x} = f(x, u)$

Define a set of grid points $\{t_{k,0}, \ldots, t_{k,K}\} \in [t_k, t_{k+1}]$ and a polynomial interpolation

$$x(\theta_k, t) := \sum_{j=0}^{K} \theta_{k,j} P_{k,j}(t) \qquad\qquad x(\theta_k, t_{k,j}) = \theta_{k,j}$$

Enforce the dynamics at the **collocation points**

$$\theta_{k,0} = x_k \qquad\qquad \text{Initial condition}$$

$$\sum_{j=0}^{K} \theta_{k,j} \dot{P}_{k,j}(t_{k,j}) = f(\theta_{k,j}, u_k) \qquad\qquad \text{Dynamics}$$

Collocation Constraints

Collocation conditions:

$$\theta_{k,0} = x_k \qquad \text{Initial condition}$$

$$\sum_{j=0}^{K} \theta_{k,j} \dot{P}_{k,j}(t_{k,j}) = f(\theta_{k,j}, u_k) \qquad \text{Dynamics}$$

Re-write this as

$$\begin{aligned} \theta_{k,0} &= x_k \\ D\theta_k &= f(\theta_k, u_k) \end{aligned}$$

where the elements of the **derivative matrix** D are the constants $\dot{P}_{k,j}(t_{k,j})$ and $f(\theta_k, u_k) = \begin{bmatrix} f(\theta_{k,0}, u_k) & \cdots & f(\theta_{k,K}, u_k) \end{bmatrix}^T$

Quadrature Rules

How to compute the value function?

$$V = \int_0^T l(x, u) dt$$

Quadrature Rules

How to compute the value function?

$$V = \int_0^T l(x, u)dt$$

We note that this is equivalent to:

$$\dot{v} = l(x, u) \qquad\qquad V = v(T)$$

We can apply our discretization schemes to the ODE $l(x, u)$.

Quadrature Rules

How to compute the value function?

$$V = \int_0^T l(x, u) dt$$

We note that this is equivalent to:

$$\dot{v} = l(x, u) \qquad\qquad V = v(T)$$

We can apply our discretization schemes to the ODE $l(x, u)$. Consider the collocation method

$$D\theta^v = l(\theta^x, u)$$

where θ^v and θ^x are the values of $v(t)$ and $x(t)$ at the collocation points

$$\theta^v = D^{-1} l(\theta^x, u)$$

Note that we only want $v(T) = \theta^v(1) = w l(\theta^x, u)$, where w is the first row of D^{-1}.

Collocation - Optimization Problem

Putting it all together:

$$
\begin{aligned}
\min_{\{u_k\},\{\theta_k^x\}} \quad & \sum_{i=0}^{N} w^T l(\theta_i^x, u_i) \\
\text{s.t.} \quad & D\theta_i^x = f(\theta_i^x, u_i) \\
& \theta_i^x(end) = \theta_{i+1}^x(1) \\
& x_i = \theta_i^x(1) \\
& x_i \in X, u_i \in U
\end{aligned}
$$

The size of this problem is

$$\text{(horizion)} \times (\text{(num inputs)} + \text{(num states)} \times \text{(num collocation points)})$$

This is quite large, but also quite sparse and structured.

Note that I've been a bit loose with the notation here in the translation from 1D to nD and the differentiation matrix here would be the Kronecker product $D \otimes I_n$.

Gradients

What is the Gradient of an Integral?

We now have

$$x_{k+1} = \hat{f}(x_k, u_k) \leftarrow \hat{f} = RK4$$

To solve the optimization problem, we need $\nabla \hat{f}$

How to compute the derivative of an algorithm?!

Symbolic / Manual Differentiation <u>Sucks</u>

Consider the simple system

$$\dot{x} = f(x) = x^2$$

Discretize with a sample period of $h = 1s$ using RK4

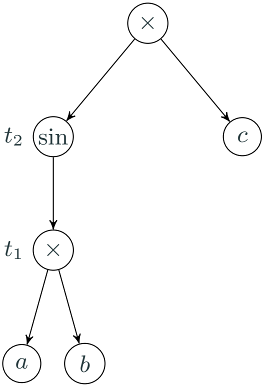
$$x^+ = \hat{f}(x)$$

The derivative of $\hat{f}$ is

$$\frac{x}{3} + \frac{\left(2\left(\left(\frac{x^2}{2} + x\right)(x+1) + 1\right)\left(x + \frac{\left(\frac{x^2}{2}+x\right)^2}{2}\right) + 1\right)\left(x + \left(x + \frac{\left(\frac{x^2}{2}+x\right)^2}{2}\right)^2\right)}{3} + \frac{2\left(\frac{x^2}{2} + x\right)(x+1)}{3} + \frac{2\left(\left(\frac{x^2}{2} + x\right)(x+1) + 1\right)\left(x + \frac{\left(\frac{x^2}{2}+x\right)^2}{2}\right)}{3} + 1$$

$$y = \sin(a \times b) \times c$$

can be written via the **computation graph** of elementary operations



Sequence of elementary operations

- Each intrinsic $v = \phi(w, u)$ has local partials $\frac{\partial \phi}{\partial w}, \frac{\partial \phi}{\partial u}$
- e.g., $\sin(t_1)$ yields $p_1 = \cos(t_1)$

$$t_1 = a \times b$$

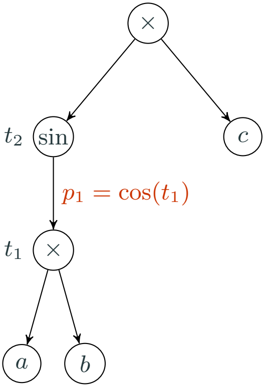
$$t_2 = \sin(t_1)$$

$$y = t_2 \times c$$

[1]Slides on AD based on "Introduction to Algorithmic Differentiation" by J. Utke, Argonne National Laboratory Mathematics and Computer Science Division, 2013

$$y = \sin(a \times b) \times c$$

can be written via the **computation graph** of elementary operations



Sequence of elementary operations

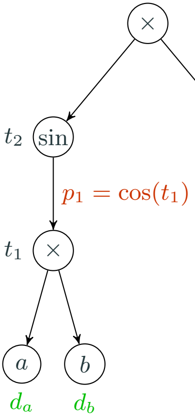
- Each intrinsic $v = \phi(w, u)$ has local partials $\frac{\partial \phi}{\partial w}, \frac{\partial \phi}{\partial u}$
- e.g., $\sin(t_1)$ yields $p_1 = \cos(t_1)$

$$t_1 = a \times b$$
$$p_1 = \cos(t_1)$$
$$t_2 = \sin(t_1)$$
$$y = t_2 \times c$$

[1] Slides on AD based on "Introduction to Algorithmic Differentiation" by J. Utke, Argonne National Laboratory Mathematics and Computer Science Division, 2013

Algorithmic Differentiation - The Rough Idea

- associate each variable v with a derivative $\dot{v}$
- take a point (a_0, b_0, c_0) and a direction $(\dot{a}, \dot{b}, \dot{c})$
- for each $v = \phi(w, u)$ propagate forward in order $\dot{v} = \frac{\partial \phi}{w}\dot{w} + \frac{\partial \phi}{w}\dot{u}$



- Associate a derivative with each variable $[a, d_a]$
- Interleave computations

$$t_1 = a \times b$$

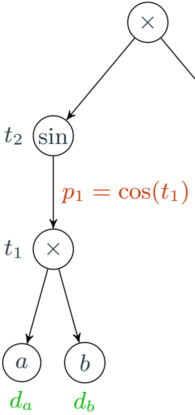
$$p_1 = \cos(t_1)$$

$$t_2 = \sin(t_1)$$

$$y = t_2 \times c$$

Algorithmic Differentiation - The Rough Idea

- associate each variable v with a derivative $\dot{v}$
- take a point (a_0, b_0, c_0) and a direction $(\dot{a}, \dot{b}, \dot{c})$
- for each $v = \phi(w, u)$ propagate forward in order $\dot{v} = \frac{\partial \phi}{w}\dot{w} + \frac{\partial \phi}{w}\dot{u}$



- Associate a derivative with each variable $[a, d_a]$
- Interleave computations

$$
\begin{aligned}
t_1 &= a \times b \\
d_{t_1} &= d_a \times b + d_b \times a \\
p_1 &= \cos(t_1) \\
t_2 &= \sin(t_1) \\
d_{t_2} &= d_{t_1} \times p_1 \\
y &= t_2 \times c \\
d_y &= d_{t_2} \times c + d_c \times t_2
\end{aligned}
$$

Algorithmic Differentiation - The Rough Idea

- What is returned: $\dot{y} = J\dot{x}$ computed at x_0
- Example: $(\dot{a}, \dot{b}, \dot{c}) = (1, 0, 0)$ will compute the first column of J
- Can compute J by evaluating the function $\mathrm{length}(x)$ times
- For optimization, we normally only need the product of J and a vector, which can be done in one computation

Example - Jacobian via Algorithmic Differentiation - CASADI

To MATLAB / CASADI : `sym_example.m`

NMPC

To MATLAB / CASADI : `pendulum.m`

Summary

Theory Very similar to linear MPC (although many important details that we haven't covered for more complex problems)

Computation A lot more complex

Practical Many, many challenges not mentioned here arising from local optima, slow computations, numerical issues, etc

Many are working on tools to make NMPC as simple and practical as (linear) MPC.