KU LEUVEN

– H04U1C –
Optimization of Mechatronic Systems
Optimal Control

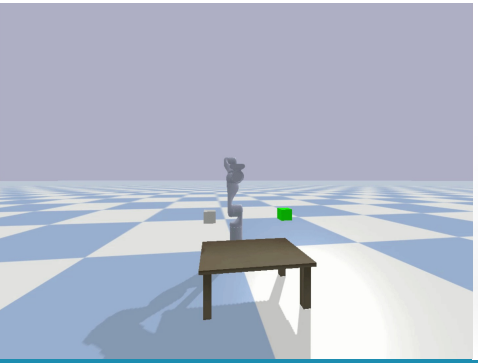
W. Decré
wilm.decre@kuleuven.be

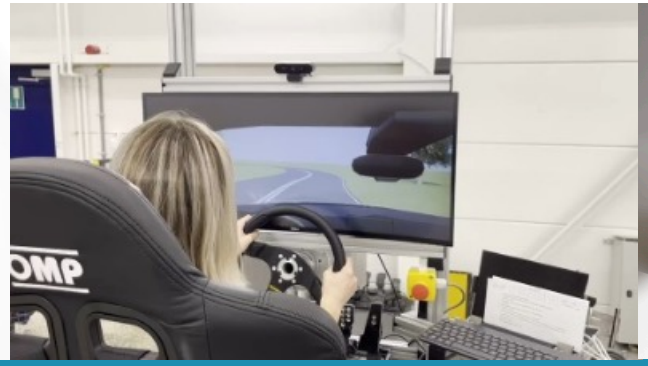
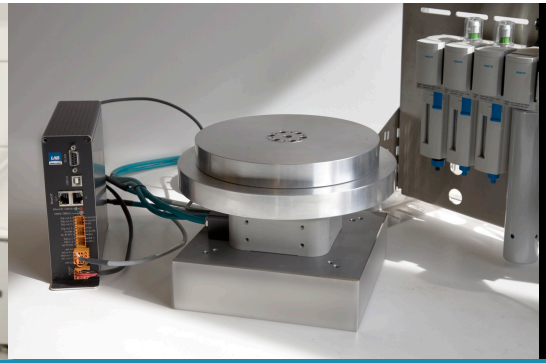
Overview

- **optimal control problems?**
- solution strategies
  - indirect methods
  - direct methods
    - shooting methods
    - collocation methods
- special cases and extensions
  - periodic problems
  - free-time problems
  - multi-stage problems

Optimal control

- find the “optimal” trajectory for a dynamical system, given
  - system dynamics, $\dot{x}(t) = f(x(t), u(t), t)$ (~ “state transition function” in the learning community) (here: deterministic)
  - initial state $x(0) = x_0$ (later we will relax this)
  - objective function, e.g., time, control effort, …
  - set of constraints, e.g., no-collision
- open-loop: find optimal control and state trajectory $u^*(t), x^*(t)$ ⟵ topic of today
  - direct methods, indirect methods
- closed-loop: find optimal control policy π^* such that $u^*(x(t), t) = \pi^*(x(t), t)$
  - dynamic programming, learning techniques (part of H02A4A Robotics next semester!)
- model predictive control
- for notational simplicity, in the remainder of the slides we will omit * and the function arguments if they are clear from the context




Why is optimal control challenging?

- challenges
  1. solve optimal control problems **fast** and **robustly** (global convergence)
     - in a few special cases we can solve optimal control problems directly, but mostly we employ iterative methods
     - high bandwidth/sampling rates, … in an MPC setting
     - expensive nonlinear functions, such as dynamics and no-collision constraints
  2. make optimal control **accessible** for users
- to address these challenges
  - implementations that exploit hardware to accelerate computations (⟶ cf. Sp. 1)

$$\text{CPU time} = \text{instruction count} \times \text{CPI} \times \text{clock cycle time}$$

  - tailored solvers that exploit the structure of optimal control problems
  - accessible software (⟶ cf. Tu. 2, Rockit tutorial)
  - efficient formulations of constrained dynamics
- additional challenges
  - taking uncertainty and disturbances into account
    - low-level feedback control
    - robust and stochastic optimal control
  - avoiding local infeasibility and spurious local minimizers

Optimal control problem

optimal control problem with state x, control u, on a horizon from 0 to T:

$$\underset{x(\cdot),u(\cdot)}{\text{minimize}}\; l_T\big(x(T)\big) + l_0\big(x(0)\big) + \int_0^T l(x(t),u(t),t)\mathrm{d}t \quad \longleftarrow \text{stagewise cost}$$

$$\text{subject to}$$

$$\begin{cases} \dot{x} = f(x,u,t) & \longleftarrow \text{system dynamics} \\ g(x,u,t) = 0 & \longleftarrow \text{stagewise equality constraints} \\ h(x,u,t) \leq 0 & \longleftarrow \text{stagewise inequality constraints} \end{cases}$$

with, unless noted otherwise, l, l_T, l_0, f, g and h twice continuously differentiable

this problem is infinite-dimensional… solution methods?

- indirect methods – "*optimize then discretize*"
- direct methods – "*discretize then optimize*" ⟵ topic of this lecture

Overview

- optimal control problems?
- **solution strategies**
  - indirect methods
  - **direct methods**
    - shooting methods
    - collocation methods
- special cases and extensions
  - periodic problems
  - free-time problems
  - multi-stage problems

Direct methods

- *"discretize then optimize"*
  1. discretize the problem using a transcription method
     - shooting
     - collocation
  2. solve the discretized problem

$$\underset{x(\cdot),u(\cdot)}{\text{minimize}}\; l_T\big(x(T)\big) + l_0\big(x(0)\big) + \int_0^T l(x(t),u(t),t)\mathrm{d}t$$
$$\text{subject to}$$
$$\begin{cases} \dot{x} = f(x,u,t) \\ g(x,u,t) = 0 \\ h(x,u,t) \leq 0 \end{cases}$$

Direct methods

- *"discretize then optimize"*
  1. discretize the problem using a transcription method
     - **shooting**
     - collocation
  2. solve the discretized problem

$$\underset{x(\cdot),u(\cdot)}{\text{minimize}}\; l_T\big(x(T)\big) + l_0\big(x(0)\big) + \int_0^T l(x(t),u(t),t)\mathrm{d}t$$
$$\text{subject to}$$
$$\begin{cases} \dot{x} = f(x,u,t) \\ g(x,u,t) = 0 \\ h(x,u,t) \leq 0 \end{cases}$$

Direct shooting

$$\underset{x(\cdot),u(\cdot)}{\text{minimize}}\; l_T\big(x(T)\big) + l_0\big(x(0)\big) + \int_0^T l(x(t),u(t),t)\mathrm{d}t$$

subject to

$$\begin{cases} \dot{x} = f(x,u,t) \\ g(x,u,t) = 0 \\ h(x,u,t) \leq 0 \end{cases}$$

we discretize the controls u on a grid of length K

- here we assume zero-order hold (zoh) controls
- we hence have controls $u[0], u[1], \ldots, u[K-1]$
- we use (numerical) integrators to compute the state and cost (cont. on next slides)

in its basic form, we use the same gridding, and obtain:

$$\underset{x[\cdot],u[\cdot]}{\text{minimize}}\; l_T(x[K]) + l_0(x[0]) + \sum_{k=0}^{K-1} l_d(x[k],u[k],k)$$

subject to

$$\begin{cases} x[k+1] = f_d(x[k],u[k],k) \\ g(x[k],u[k],k) = 0, g_K(x[K]) = 0 \text{ for } k = 0,1,\ldots,K-1 \\ h(x[k],u[k],k) \leq 0, h_K(x[K]) \leq 0 \end{cases}$$

Direct shooting

$$\underset{x[\cdot],u[\cdot]}{\text{minimize}}\; l_T(x[K]) + l_0(x[0]) + \sum_{k=0}^{K-1} l_d(x[k], u[k], k)$$

subject to

$$\begin{cases} x[k+1] = f_d(x[k], u[k], k) \\ g(x[k], u[k], k) = 0, g_K(x[K]) = 0 \text{ for } k = 0, 1, \dots, K-1 \\ h(x[k], u[k], k) \leq 0, h_K(x[K]) \leq 0 \end{cases}$$

- single shooting: we eliminate $x[k]$ for $k = 1, 2, \dots, K$ using the discrete-time state dynamics, and retain only $x[0]$ and the controls as decision variables
- multiple shooting: we keep the states as decision variables
- remark 1: if the initial state is given, it can be eliminated as well
- remark 2: the constraints are only enforced on a grid

Direct shooting

single versus multiple shooting?

single shooting	multiple shooting
smaller problem size	larger problem size, yet specific sparsity pattern that can be exploited
feasible initialization w.r.t. the system dynamics	allows infeasible initialization w.r.t. the system dynamics
	improved distribution of nonlinearities, leading to superior convergence
	some of the underlying computations can be parallelized (more) easily

$\Longrightarrow$in practice: in most cases that we will consider, multiple shooting is preferred

Rockit/CasADi, you can inspect the sparsity with the `spy` function

Direct shooting - integrators

$$\dot{x} = f(x, u, t)$$
$$\text{step size } h = \frac{T}{K}$$

forward (or explicit) Euler

- $x[k+1] = x[k] + hf(x[k], u[k], k) = f_d(x[k], u[k], k)$
- local truncation error (LTE) (assuming exact arithmetic)
  - local, we have: $x[k] = x(kh)$
  - Taylor expansion: $x\big((k+1)h\big) = x(kh) + h\dot{x}(kh) + \frac{1}{2}h^2\ddot{x}(kh) + O(h^3)$
  - LTE: $x\big((k+1)h\big) - x[k+1] = \frac{1}{2}h^2\ddot{x}(kh) + O(h^3)$
  - error is approximately proportional to h^2 (for small h) (assuming a bounded third derivative)
- global truncation error (error over time) is approximately proportional to h (under some mild assumptions)
  - intuition: number of steps for a given integration time is proportional to $\frac{1}{h}$, and error in each step to h^2
  - hence: explicit Euler is said to be “first order”

explicit Runge-Kutta 4 (RK4)

- $x[k+1] = x[k] + \frac{h}{6}(m_1 + 2m_2 + 2m_3 + m_4)$, with
  - $m_1 = f(x[k], u[k], kh)$
  - $m_2 = f\left(x[k] + \frac{hm_1}{2}, u[k], kh + \frac{h}{2}\right)$
  - $m_3 = f\left(x[k] + \frac{hm_2}{2}, u[k], kh + \frac{h}{2}\right)$
  - $m_4 = f(x[k] + hm_3, u[k], kh + h)$
- a single step is much more expensive to calculate
- but… much more accurate (for small h): global truncation error $O(h^4)$, hence “fourth-order method” (under some mild assumptions)

Direct shooting - integrators

can we do better?

for linear time-invariant systems we can integrate *exactly* (assuming zoh controls) …

- $\dot{x} = Ax + Bu$
- on a single integration interval we have a constant u, hence:
- $\begin{bmatrix} x[k+1] \\ - \end{bmatrix} = e^{\begin{bmatrix} A & B \\ 0 & 0 \end{bmatrix} h} \begin{bmatrix} x[k] \\ u[k] \end{bmatrix} \Rightarrow x[k+1] = A_d x[k] + B_d u[k]$
- routines available in, e.g., `scipy`: `scipy.signal.cont2discrete`

Direct methods

- *"discretize then optimize"*
  1. discretize the problem using a transcription method
     - shooting
     - **collocation**
  2. solve the discretized problem

$$\underset{x(\cdot),u(\cdot)}{\text{minimize}}\; l_T\big(x(T)\big) + l_0\big(x(0)\big) + \int_0^T l(x(t),u(t),t)\mathrm{d}t$$
$$\text{subject to}$$
$$\begin{cases} \dot{x} = f(x,u,t) \\ g(x,u,t) = 0 \\ h(x,u,t) \leq 0 \end{cases}$$

Direct collocation

- again, consider: $\dot{x} = f(x, u, t)$
- define n collocation points $0 \leq c_1 < c_2 < \ldots < c_n \leq 1$ per integration step h
- x_p is a polynomial approximation of x, of degree n, such that
  - $\begin{cases} x_p(t_k) = x[k] \\ x_p(t_k + h) = x[k+1] \end{cases}$ *no-gap* constraints to enforce continuity across steps
  - $\dot{x}_p(t_i) = f\left(x_p(t_i), u[k], t_i\right)$, with $t_i = t_k + c_i h$
- for the entire horizon we obtain a piecewise polynomial approximation – splines
- basic form: implicit (or backward) Euler, with $n = 1$ and $c_1 = 1$

Direct methods

- *"discretize then optimize"*
  1. discretize the problem using a transcription method
     - shooting
     - collocation
  2. **solve the discretized problem**

$$\underset{x(\cdot),u(\cdot)}{\text{minimize}}\; l_T\big(x(T)\big) + l_0\big(x(0)\big) + \int_0^T l(x(t),u(t),t)\mathrm{d}t$$

$$\text{subject to}$$

$$\begin{cases} \dot{x} = f(x,u,t) \\ g(x,u,t) = 0 \\ h(x,u,t) \leq 0 \end{cases}$$

NLP solvers

- after transcription (shooting or collocation) we obtain a nonlinear programming problem (NLP)

$$\begin{aligned} &\underset{x}{\text{minimize}}\ f(x) \\ &\text{subject to} \\ &\begin{cases} g(x) = 0 \\ h(x) \leq 0 \end{cases} \end{aligned}$$

- first-order necessary optimality (Karush-Kuhn-Tucker (KKT)) conditions

if x^* is a locally optimal solution, then there exist λ^*, ν^* such that

$$\begin{cases} \nabla_x \mathcal{L}(x^*, \lambda^*, \nu^*) = 0 \\ g(x^*) = 0 \\ h(x^*) \leq 0 \\ \nu^* \geq 0 \\ \nu^{*\mathsf{T}} h(x^*) = 0, \end{cases}$$

with Lagrangian $\mathcal{L}(x, \lambda, \nu) = f(x) + \lambda^{\mathsf{T}} g(x) + \nu^{\mathsf{T}} h(x)$
(under some mild assumptions)

Remark (to avoid confusion): we overloaded the symbols: x is not the state, but represents all (finite number of) decision variables

NLP solvers

- no inequality constraints ⟶ we obtain a system of nonlinear equations
  - solve iteratively, e.g., using (variant of) Newton's method
  - to improve (global) convergence, two strategies:
    - line search: 1) the search direction, and 2) the step size
    - trust region: 1) the maximum step size, and 2) the search direction + step size
- inequality constraints
  - active-set methods
    - iteratively find out which constraints are active and which ones are not
  - interior-point methods ⟵ focus of today

NLP solvers

$$\underset{x}{\text{minimize}}\ f(x) \quad \text{subject to} \quad \begin{cases} g(x) = 0 \\ h(x) \leq 0 \end{cases} \longrightarrow \underset{x,s}{\text{minimize}}\ f(x) - \mu \sum_i \log s_i \quad \text{subject to} \quad \begin{cases} g(x) = 0 \\ h(x) + s = 0 \end{cases}$$

for $\mu \to 0$

$$\longrightarrow \begin{cases} \nabla_x \mathcal{L}(x^*, \lambda^*, \nu^*) = 0 \\ g(x^*) = 0 \\ h(x^*) + s^* = 0 \\ \lceil s^* \rfloor \nu^* - \mu e = 0 \end{cases}$$

$$\begin{array}{c} \begin{matrix} \mathbf{x}_2 & \mathbf{v}_2 & \boldsymbol{\pi}_2 & \mathbf{u}_1 & \mathbf{x}_1 & \boldsymbol{\lambda}_1 & \boldsymbol{\pi}_1 & \mathbf{u}_0 & \mathbf{x}_0 & \boldsymbol{\lambda}_0 & \end{matrix} \\ \left[\begin{array}{cccccccccc:c} \mathbf{Q}_2 & \mathbf{H}_2' & -\boldsymbol{I} & & & & & & & & \mathbf{q}_2 \\ \mathbf{H}_2 & & & & & & & & & & \mathbf{h}_2 \\ -\boldsymbol{I} & & & \mathbf{B}_1 & \mathbf{A}_1 & & & & & & \mathbf{b}_1 \\ & & \mathbf{B}_1' & \mathbf{R}_1 & \mathbf{S}_1' & \mathbf{H}_{1,u}' & & & & & \mathbf{r}_1 \\ & & \mathbf{A}_1' & \mathbf{S}_1 & \mathbf{Q}_1 & \mathbf{H}_{1,x}' & -\boldsymbol{I} & & & & \mathbf{q}_1 \\ & & & \mathbf{H}_{1,u} & \mathbf{H}_{1,x} & & & & & & \mathbf{h}_1 \\ & & & & -\boldsymbol{I} & & & \mathbf{B}_0 & \mathbf{A}_0 & & \mathbf{b}_0 \\ & & & & & & \mathbf{B}_0' & \mathbf{R}_0 & \mathbf{S}_0' & \mathbf{H}_{0,u}' & \mathbf{r}_0 \\ & & & & & & \mathbf{A}_0' & \mathbf{S}_0 & \mathbf{Q}_0 & \mathbf{H}_{0,x}' & \mathbf{q}_0 \\ & & & & & & & \mathbf{H}_{0,u} & \mathbf{H}_{0,x} & & \mathbf{h}_0 \end{array}\right] \end{array}$$

structure of the primal-dual system for $K = 2$ [2]

- we again obtain a system of nonlinear equations
- μ is adjusted over iterations, in order to (hopefully) converge to a solution of the original problem
- in this course we will use interior-point solver Ipopt, with a general-purpose sparse linear solver
- alternative, in-house developed, solver Fatrop that uses tailored recursion for OCPs (https://github.com/meco-group/fatrop)

Ipopt (sketch)

$$\underset{x}{\text{minimize}}\ f(x) \quad \text{subject to} \begin{cases} g(x) = 0 \\ x \geq 0 \end{cases} \quad \rightarrow \quad \underset{x,s}{\text{minimize}}\ f(x) - \mu \sum_i \log s_i \quad \text{subject to} \begin{cases} g(x) = 0 \\ x - s = 0 \end{cases}$$

for $\mu \to 0$ $\rightarrow$

$$\begin{cases} \nabla_x \mathcal{L}(x^*, \lambda^*, \nu^*) = 0 \\ g(x^*) = 0 \\ \lceil x^* \rfloor \nu^* - \mu e = 0 \end{cases}$$

dual feasibility

primal feasibility

- consider problem formulation above
- generalizing to unconstrained x_i for some i, lower + upper bounds, and general inequalities is easy
- further reading, cf. [3]
  (some procedures, e.g., watchdog not covered here in the sketch)

Ipopt (sketch)

$$\underset{x,s}{\text{minimize}}\ f(x) - \mu \sum_i \log s_i$$

$$\text{subject to } \begin{cases} g(x) = 0 \\ x - s = 0 \end{cases}$$

for $\mu \to 0$ $\longrightarrow$

$$\begin{cases} \nabla_x \mathcal{L}(x^*, \lambda^*, \nu^*) = 0 \\ g(x^*) = 0 \\ \left[x^*\right] \nu^* - \mu e = 0 \end{cases}$$

- current iterate: $x^{(i)}, \lambda^{(i)}, \nu^{(i)}$
- linearize PD equations to compute PD system

$$\begin{bmatrix} \nabla_{xx}\mathcal{L}^{(i)} & \nabla g^{(i)} & -I \\ \nabla g^{(i)^{\mathrm{T}}} & 0 & 0 \\ \left[\nu^{(i)}\right] & 0 & \left[x^{(i)}\right] \end{bmatrix} \begin{pmatrix} p_x^{(i)} \\ p_\lambda^{(i)} \\ p_\nu^{(i)} \end{pmatrix} = - \begin{pmatrix} \nabla f^{(i)} + \nabla g^{(i)} \lambda^{(i)} - \nu^{(i)} \\ g^{(i)} \\ \left[x^{(i)}\right] \nu^{(i)} - \mu e \end{pmatrix}$$

- for many problems, this *function evaluation* is an expensive step!
  - $\nabla_{xx}\mathcal{L}^{(i)}, \nabla g^{(i)}, \nabla f^{(i)}, g^{(i)}$

Overall algorithm sketch

```
 0: while no convergence of full problem
 1:   while no convergence of barrier subproblem
 2:     evaluate PD system at current iterate
 3:     compute search direction and initial trial point
 4:     if PD not acceptable, perform inertia correction
          and regularization
 5:     while trial point not accepted
 6:       perform SOC(s)
 7:       if trial point not accepted
 8:         backtrack line search
 9:       if trial step size too small, switch to FRP
10:     compute next iterate
11:   decrease barrier parameter μ
```

```
[...]

EXIT: Optimal Solution Found.
      solver  :   t_proc      (avg)   t_wall      (avg)    n_eval
       nlp_f  | 874.00us ( 87.40us)  53.29us (  5.33us)        10
       nlp_g  |   1.12ms (112.30us)  63.05us (  6.30us)        10
  nlp_grad_f  | 831.00us ( 92.33us)  51.09us (  5.68us)         9
  nlp_hess_l  | 530.00us ( 75.71us)  32.77us (  4.68us)         7
   nlp_jac_g  | 688.00us ( 76.44us)  43.00us (  4.78us)         9
       total  | 124.76ms (124.76ms)   7.79ms (  7.79ms)         1
```

Ipopt (sketch)

$$\underset{x,s}{\text{minimize}}\ f(x) - \mu \sum_i \log s_i$$

subject to

$$\begin{cases} g(x) = 0 \\ x - s = 0 \end{cases}$$

for $\mu \to 0$ $\longrightarrow$

$$\begin{cases} \nabla_x \mathcal{L}(x^*, \lambda^*, \nu^*) = 0 \\ g(x^*) = 0 \\ \left[x^*\right] \nu^* - \mu e = 0 \end{cases}$$

- solve linear system

$$\begin{bmatrix} \nabla_{xx}\mathcal{L}^{(i)} & \nabla g^{(i)} & -I \\ \nabla g^{(i)^{\mathrm{T}}} & 0 & 0 \\ \left[\nu^{(i)}\right] & 0 & \left[x^{(i)}\right] \end{bmatrix} \begin{pmatrix} p_x^{(i)} \\ p_\lambda^{(i)} \\ p_\nu^{(i)} \end{pmatrix} = - \begin{pmatrix} \nabla f^{(i)} + \nabla g^{(i)} \lambda^{(i)} - \nu^{(i)} \\ g^{(i)} \\ \left[x^{(i)}\right] \nu^{(i)} - \mu e \end{pmatrix}$$

$$\begin{cases} \begin{bmatrix} \nabla_{xx}\mathcal{L}^{(i)} + \Sigma^{(i)} & \nabla g^{(i)} \\ \nabla g^{(i)^{\mathrm{T}}} & 0 \end{bmatrix} \begin{pmatrix} p_x^{(i)} \\ p_\lambda^{(i)} \end{pmatrix} = - \begin{pmatrix} \nabla f^{(i)} - \mu \left[x^{(i)}\right]^{-1} e + \nabla g^{(i)} \lambda^{(i)} \\ g^{(i)} \end{pmatrix} \\ p_\nu^{(i)} = \mu \left[x^{(i)}\right]^{-1} e - \nu^{(i)} - \Sigma^{(i)} p_x^{(i)} \end{cases}$$

with $\Sigma^{(i)} = \left[x^{(i)}\right]^{-1} \left[\nu^{(i)}\right]$

Overall algorithm sketch

```
0:  while no convergence of full problem
1:    while no convergence of barrier subproblem
2:      evaluate PD system at current iterate
3:      compute search direction and initial trial point
4:      if PD not acceptable, perform inertia correction
          and regularization
5:      while trial point not accepted
6:        perform SOC(s)
7:        if trial point not accepted
8:          backtrack line search
9:        if trial step size too small, switch to FRP
10:     compute next iterate
11:   decrease barrier parameter μ
```

- set initial values for $\alpha, \alpha_\nu \in (0,1]$
- initial trial point:
  - $x^{(i+1)} \leftarrow x^{(i)} + \alpha p_x^{(i)}$
  - $\lambda^{(i+1)} \leftarrow \lambda^{(i)} + \alpha p_\lambda^{(i)}$
  - $\nu^{(i+1)} \leftarrow \nu^{(i)} + \alpha_\nu p_\nu^{(i)}$

Ipopt (sketch)

$$\underset{x,s}{\text{minimize}}\ f(x) - \mu \sum_i \log s_i$$

subject to

$$\begin{cases} g(x) = 0 \\ x - s = 0 \end{cases}$$

for $\mu \to 0$ $\longrightarrow$

$$\begin{cases} \nabla_x \mathcal{L}(x^*, \lambda^*, \nu^*) = 0 \\ g(x^*) = 0 \\ \left[x^*\right] \nu^* - \mu e = 0 \end{cases}$$

- in order to have a unique solution and descent direction:
  - reduced Hessian must be positive definite
  - PD system must be nonsingular

- inertia correction and regularization

$$\begin{bmatrix} \nabla_{xx}\mathcal{L}^{(i)} + \Sigma^{(i)} + \delta_w I & \nabla g^{(i)} \\ \nabla g^{(i)^{\mathrm{T}}} & -\delta_c I \end{bmatrix} \begin{pmatrix} p_x^{(i)} \\ p_\lambda^{(i)} \end{pmatrix} = - \begin{pmatrix} \nabla f^{(i)} - \mu \left[x^{(i)}\right]^{-1} e + \nabla g^{(i)} \lambda^{(i)} \\ g^{(i)} \end{pmatrix}$$

for some $\delta_w, \delta_c \geq 0$

- compute search direction and initial trial point using this modified system

Overall algorithm sketch

```
0:  while no convergence of full problem
1:    while no convergence of barrier subproblem
2:      evaluate PD system at current iterate
3:      compute search direction and initial trial point
4:      if PD not acceptable, perform inertia correction
          and regularization
5:      while trial point not accepted
6:        perform SOC(s)
7:        if trial point not accepted
8:          backtrack line search
9:        if trial step size too small, switch to FRP
10:     compute next iterate
11:   decrease barrier parameter μ
```

Ipopt (sketch)

$$\underset{x,s}{\text{minimize}}\ f(x) - \mu \sum_i \log s_i$$

subject to

$$\begin{cases} g(x) = 0 \\ x - s = 0 \end{cases}$$

for $\mu \to 0$ $\rightarrow$

$$\begin{cases} \nabla_x \mathcal{L}(x^*, \lambda^*, \nu^*) = 0 \\ g(x^*) = 0 \\ [x^*]\nu^* - \mu e = 0 \end{cases}$$

- Ipopt uses a filter technique to check if a trial point is acceptable
- basic concept
  - filter contains points that do not *dominate* each other, meaning that not both the objective function and the constraint violation of one point are lower than any other one
  - a new iterate is acceptable to the filter if is not dominated by any point in the filter

Overall algorithm sketch

```
0:  while no convergence of full problem
1:    while no convergence of barrier subproblem
2:      evaluate PD system at current iterate
3:      compute search direction and initial trial point
4:      if PD not acceptable, perform inertia correction
          and regularization
5:      while trial point not accepted
6:        perform SOC(s)
7:        if trial point not accepted
8:          backtrack line search
9:        if trial step size too small, switch to FRP
10:     compute next iterate
11:   decrease barrier parameter μ
```

Ipopt (sketch)

$$\underset{x,s}{\text{minimize}}\ f(x) - \mu \sum_i \log s_i$$

$$\text{subject to } \begin{cases} g(x) = 0 \\ x - s = 0 \end{cases}$$

for $\mu \to 0$ $\longrightarrow$

$$\begin{cases} \nabla_x \mathcal{L}(x^*, \lambda^*, \nu^*) = 0 \\ g(x^*) = 0 \\ \left[x^*\right]\nu^* - \mu e = 0 \end{cases}$$

- second-order correction if trial step: $\tilde{p}_x^{(i)}$ has been rejected and constraint violation increases for this step
- goal: try to reduce infeasibility
- compute correction for the search direction that satisfies $\nabla g^{(i)}\, p_{x,\text{soc}}^{(i)} + g(x^{(i)} + \tilde{p}_x^{(i)}) = 0$
- corrected search direction then becomes

$$\hat{p}_x^{(i)} = \tilde{p}_x^{(i)} + p_{x,\text{soc}}^{(i)}$$

Overall algorithm sketch

```
0:  while no convergence of full problem
1:   while no convergence of barrier subproblem
2:    evaluate PD system at current iterate
3:    compute search direction and initial trial point
4:    if PD not acceptable, perform inertia correction
        and regularization
5:    while trial point not accepted
6:     perform SOC(s)
7:      if trial point not accepted
8:       backtrack line search
9:      if trial step size too small, switch to FRP
10:   compute next iterate
11:  decrease barrier parameter μ
```

Ipopt (sketch)

$$\underset{x,s}{\text{minimize}}\ f(x) - \mu \sum_i \log s_i$$

subject to

$$\begin{cases} g(x) = 0 \\ x - s = 0 \end{cases}$$

for $\mu \to 0$ $\rightarrow$

$$\begin{cases} \nabla_x \mathcal{L}(x^*, \lambda^*, \nu^*) = 0 \\ g(x^*) = 0 \\ \left[x^*\right]\nu^* - \mu e = 0 \end{cases}$$

- backtrack line search
  - $\alpha \leftarrow \frac{1}{2}\alpha$
  - $x^{(i+1)} \leftarrow x^{(i)} + \alpha p_x^{(i)}$
  - $\lambda^{(i+1)} \leftarrow \lambda^{(i)} + \alpha p_\lambda^{(i)}$
  - $\nu^{(i+1)} \leftarrow \nu^{(i)} + \alpha_\nu p_\nu^{(i)}$

Overall algorithm sketch

```
0:  while no convergence of full problem
1:    while no convergence of barrier subproblem
2:      evaluate PD system at current iterate
3:      compute search direction and initial trial point
4:      if PD not acceptable, perform inertia correction
            and regularization
5:      while trial point not accepted
6:        perform SOC(s)
7:        if trial point not accepted
8:          backtrack line search
9:        if trial step size too small, switch to FRP
10:     compute next iterate
11:   decrease barrier parameter μ
```

Ipopt (sketch)

$$\underset{x,s}{\text{minimize}}\ f(x) - \mu \sum_i \log s_i$$

subject to $\begin{cases} g(x) = 0 \\ x - s = 0 \end{cases}$

for $\mu \to 0$ $\longrightarrow$ $\begin{cases} \nabla_x \mathcal{L}(x^*, \lambda^*, \nu^*) = 0 \\ g(x^*) = 0 \\ \left[x^*\right]\nu^* - \mu e = 0 \end{cases}$

- feasibility restoration phase
  - restore feasibility or detect local infeasibility
- regular and alternative method
- main idea of regular method:

$$\underset{x}{\text{minimize}} \|g(x)\|_1 + \frac{\zeta}{2}\|D_R(x - x_R)\|_2^2$$

- main idea of alternative method: check that new trial point satisfies primal–dual equations sufficiently better than current iterate

Overall algorithm sketch

```
0:  while no convergence of full problem
1:    while no convergence of barrier subproblem
2:      evaluate PD system at current iterate
3:      compute search direction and initial trial point
4:      if PD not acceptable, perform inertia correction
            and regularization
5:      while trial point not accepted
6:        perform SOC(s)
7:        if trial point not accepted
8:          backtrack line search
9:        if trial step size too small, switch to FRP
10:     compute next iterate
11:   decrease barrier parameter μ
```

Ipopt (sketch)

$$\underset{x,y}{\text{minimize}}\ 2(x^2+y^2-1)-x$$
$$\text{subject to}\ \begin{cases} x^2+y^2=1 \\ x \geq 0 \end{cases}$$

now we can interpret the main parts of the Ipopt output

```
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
   0  4.8000000e-01 6.40e-01 7.58e-01  -1.0 0.00e+00    -  0.00e+00 0.00e+00   0
   1 -9.3019474e-02 5.47e-01 4.70e-01  -1.7 6.30e-01    -  6.78e-01 1.00e+00f  1
   2 -5.6304824e-01 1.96e-01 4.98e-01  -1.7 8.21e-01    -  1.00e+00 1.00e+00F  1
   3 -9.3469596e-01 4.28e-02 4.08e-02  -1.7 5.08e-01    -  1.00e+00 1.00e+00F  1
   4 -9.9696688e-01 2.02e-03 6.93e-05  -2.5 4.06e-02    -  1.00e+00 1.00e+00h  1
   5 -9.9999846e-01 1.02e-06 2.03e-06  -3.8 1.01e-03    -  1.00e+00 1.00e+00h  1
   6 -1.0000000e+00 2.82e-13 7.84e-11  -5.7 5.12e-07    -  1.00e+00 1.00e+00h  1
   7 -1.0000000e+00 0.00e+00 2.51e-14  -8.6 2.12e-11    -  1.00e+00 1.00e+00h  1

Number of Iterations....: 7
```

```python
import casadi as cs

opti = cs.Opti()
x = opti.variable(2)

opti.minimize(2*(x[0]**2+x[1]**2-1)-x[0])
opti.subject_to(x[0]**2+x[1]**2-1==0)
opti.subject_to(x[0]>= 0.0)

p_opts = {'expand': True}
s_opts = {"print_level": 5}
solver = opti.solver('ipopt', p_opts, s_opts)
opti.set_initial(x,[0.8, 1.0])

sol = opti.solve()
print(sol.value(x))
```

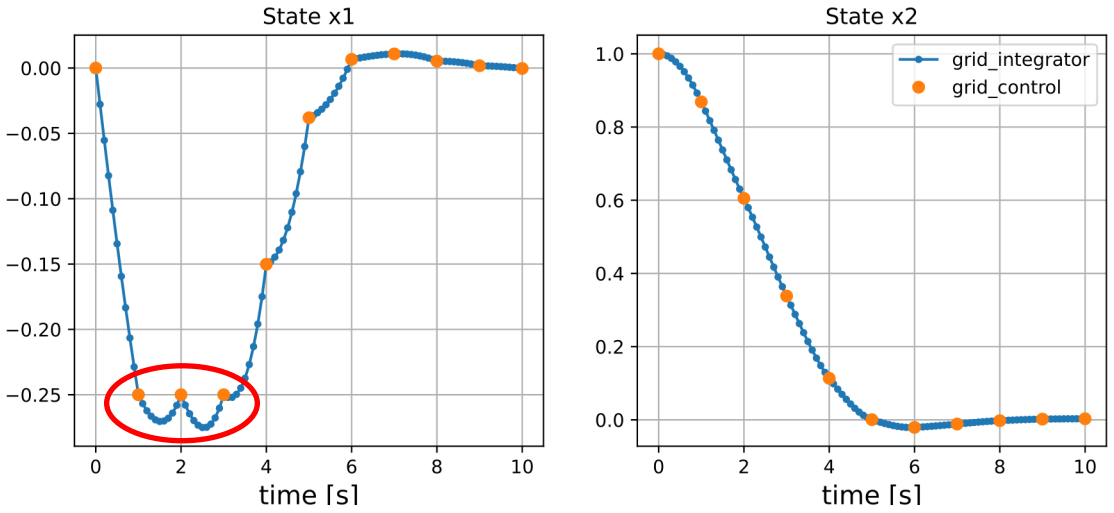
and the columns of output are defined as,

- iter: The current iteration count. This includes regular iterations and iterations during the restoration phase. If the algorithm is in the restoration phase, the letter "r" will be appended to the iteration number.
- objective: The unscaled objective value at the current point. During the restoration phase, this value remains the unscaled objective value for the original problem.
- inf_pr: The unscaled constraint violation at the current point. This quantity is the infinity-norm (max) of the (unscaled) constraints ($g^L \leq g(x) \leq g^U$ in (NLP)). During the restoration phase, this value remains the constraint violation of the original problem at the current point. The option inf_pr_output can be used to switch to the printing of a different quantity.
- inf_du: The scaled dual infeasibility at the current point. This quantity measure the infinity-norm (max) of the internal dual infeasibility, Eq. (4a) in the implementation paper [12], including inequality constraints reformulated using slack variables and problem scaling. During the restoration phase, this is the value of the dual infeasibility for the restoration phase problem.
- lg(mu): $\log_{10}$ of the value of the barrier parameter μ.
- ||d||: The infinity norm (max) of the primal step (for the original variables x and the internal slack variables s). During the restoration phase, this value includes the values of additional variables, p and n (see Eq. (30) in [12]).
- lg(rg): $\log_{10}$ of the value of the regularization term for the Hessian of the Lagrangian in the augmented system (δ_w in Eq. (26) and Section 3.1 in [12]). A dash ("-") indicates that no regularization was done.
- alpha_du: The stepsize for the dual variables (α_k^z in Eq. (14c) in [12]).
- alpha_pr: The stepsize for the primal variables (α_k in Eq. (14a) in [12]). The number is usually followed by a character for additional diagnostic information regarding the step acceptance criterion:
- ls: The number of backtracking line search steps (does not include second-order correction steps).

- more information available at https://coin-or.github.io/Ipopt/OUTPUT.html

Rockit example

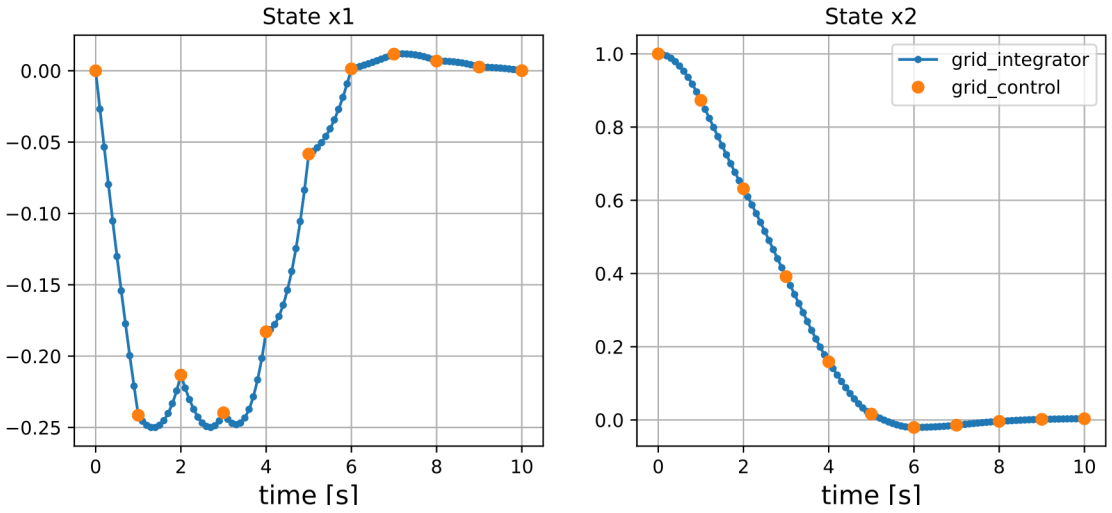
hello world example




```
from numpy import *
from rockit import *

ocp = Ocp(t0=0, T=10)

x1 = ocp.state()
x2 = ocp.state()
u = ocp.control()
e = 1 - x2**2

ocp.set_der(x1, e * x1 - x2 + u)
ocp.set_der(x2, x1)

ocp.add_objective(ocp.integral(x1**2 + x2**2 + u**2))
ocp.add_objective(ocp.at_tf(x1**2))

ocp.subject_to(x1 >= -0.25)
ocp.subject_to(-1 <= (u <= 1 ))

ocp.subject_to(ocp.at_t0(x1) == 0)
ocp.subject_to(ocp.at_t0(x2) == 1)

ocp.solver('ipopt')
method = MultipleShooting(N=10, M=10, intg='rk')
ocp.method(method)

ocp.set_initial(x2, 0)
ocp.set_initial(x1, ocp.t/10)
ocp.set_initial(u, linspace(0, 1, 10))

sol = ocp.solve()
```

Rockit example

hello world example




```
from numpy import *
from rockit import *

ocp = Ocp(t0=0, T=10)

x1 = ocp.state()
x2 = ocp.state()
u = ocp.control()
e = 1 - x2**2

ocp.set_der(x1, e * x1 - x2 + u)
ocp.set_der(x2, x1)

ocp.add_objective(ocp.integral(x1**2 + x2**2 + u**2))
ocp.add_objective(ocp.at_tf(x1**2))

ocp.subject_to(x1 >= -0.25, grid='integrator')
ocp.subject_to(-1 <= (u <= 1 ))

ocp.subject_to(ocp.at_t0(x1) == 0)
ocp.subject_to(ocp.at_t0(x2) == 1)

ocp.solver('ipopt')
method = MultipleShooting(N=10, M=10, intg='rk')
ocp.method(method)

ocp.set_initial(x2, 0)
ocp.set_initial(x1, ocp.t/10)
ocp.set_initial(u, linspace(0, 1, 10))

sol = ocp.solve()
```

Overview

- optimal control problems?
- solution strategies
  - indirect methods
  - direct methods
    - shooting methods
    - collocation methods
- **special cases and extensions**
  - periodic problems
  - free-time problems
  - multi-stage problems

Periodic problems

constraints to impose that the terminal state is equal to the initial state

- trivial if the initial state (and hence also the terminal state) is known
- otherwise it breaks our stagewise structure…

$$\underset{x(\cdot),u(\cdot)}{\text{minimize}}\; l_T\big(x(T)\big) + l_0\big(x(0)\big) + \int_0^T l(x(t),u(t),t)\mathrm{d}t$$

$$\text{subject to}\begin{cases}\dot{x} = f(x,u,t)\\ g(x,u,t) = 0\\ h(x,u,t) \leq 0\\ x(0) = x(T)\end{cases}$$

Periodic problems

how do we deal with this?

- when using a general-purpose NLP solver, the solver can directly deal with this problem
- when using an OCP solver, we can rewrite the problem by augmenting the state
  - $\hat{x} = \begin{bmatrix} x \\ \xi \end{bmatrix}, \hat{f}(\hat{x}, u, t) = \begin{bmatrix} f(x, u, t) \\ 0 \end{bmatrix}, \hat{g}(\hat{x}, u, t) = \begin{bmatrix} g(x, u, t) \\ \xi(0) = x(0) \\ \xi(T) = x(T) \end{bmatrix}$
  - $\hat{h}(\hat{x}, u, t) = h(x, u, t), \hat{l}_T\big(\hat{x}(T)\big) = l_T\big(x(T)\big), \hat{l}_0\big(\hat{x}(0)\big) = l_0\big(x(0)\big), \hat{l}(\hat{x}(t), u(t), t) = l(x(t), u(t), t)$
- this results in a standard OCP problem:

$$\underset{\hat{x}(\cdot), u(\cdot)}{\text{minimize}}\ \hat{l}_T\big(\hat{x}(T)\big) + \hat{l}_0\big(\hat{x}(0)\big) + \int_0^T \hat{l}(\hat{x}(t), u(t), t)\mathrm{d}t$$

$$\text{subject to}$$

$$\begin{cases} \dot{\hat{x}} = \hat{f}(\hat{x}, u, t) \\ \hat{g}(\hat{x}, u, t) = 0 \\ \hat{h}(\hat{x}, u, t) \leq 0 \end{cases}$$

- drawback: larger problem size (the number of state variables doubles!), yet there is structure that can be exploited

Free-time problems

the end time becomes an optimization variable

$$\underset{x(\cdot),u(\cdot),T}{\text{minimize}}\; l_T(x(T),T) + l_0\big(x(0)\big) + \int_0^T l(x(t),u(t),t)\mathrm{d}t$$

subject to

$$\begin{cases} \dot{x} = f(x,u,t) \\ g(x,u,t) = 0 \\ h(x,u,t) \leq 0 \\ T \geq 0 \end{cases}$$

we don't know the length of the horizon, so how to discretize…?

Free-time problems

how can we deal with this?

we introduce scaled time: $s = \frac{t}{T}$ and then rewrite the problem with s as independent variable, the horizon then runs from $s = 0$ to $s = 1$, and we can transform to an NLP!

$$\underset{x(\cdot),\, u(\cdot),\, T}{\text{minimize}}\; l_T(x(1), T) + l_0\big(x(0)\big) + \int_0^1 l(x(s), u(s), sT)\mathrm{d}s$$

subject to

$$\begin{cases} \dfrac{dx}{ds} = T f(x, u, sT) \\ g(x, u, sT) = 0 \\ h(x, u, sT) \leq 0 \\ T \geq 0 \end{cases}$$

a general NLP solver can directly solve this problem
for an OCP solver, we introduce an additional state variable
variants are possible with non-uniform grids

Multi-stage problems

- multiple optimal control problems “stitched together”
  - e.g., discontinuous change in dynamics or other constraints

$$\underset{x(\cdot),u(\cdot),T_1,T_2}{\text{minimize}} \; l_{T_2}\big(x(T_2)\big) + \int_{T_1}^{T_2} l_2(x(t),u(t),t)\mathrm{d}t + l_{T_1}\big(x(T_1)\big) + l_0\big(x(0)\big) + \int_0^{T_1} l_1(x(t),u(t),t)\mathrm{d}t$$

subject to

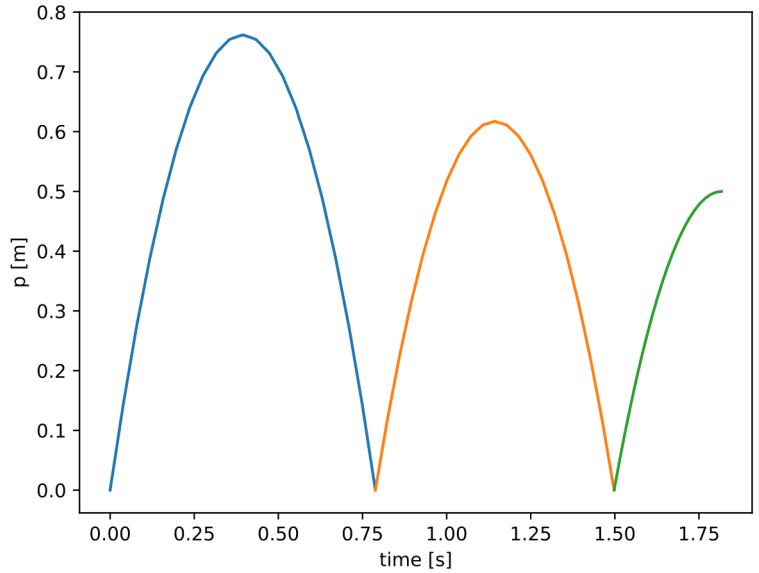
$$\begin{cases} \dot{x} = f_1(x,u,t) \\ g_1(x,u,t) = 0 \\ h_1(x,u,t) \leq 0 \\ T_1 \geq 0 \\ \dot{x} = f_2(x,u,t) \\ g_2(x,u,t) = 0 \\ h_2(x,u,t) \leq 0 \\ T_2 \geq T_1 \end{cases}$$

imposed during first stage from 0 to T_1

imposed during second stage from T_1 to T_2

Multi-stage problems

- Rockit example: shooting bouncing ball
  - $x = \begin{bmatrix} p \\ v \end{bmatrix}$
  - two bounces, at the end of the final bounce the ball should reach $0{,}5\ \mathrm{m}$
  - initial velocity is unknown




```
ocp = Ocp()

stage = ocp.stage(t0=FreeTime(0), T=FreeTime(1))
p = stage.state()
v = stage.state()

stage.set_der(p, v)
stage.set_der(v, -9.81)

stage.subject_to(stage.at_t0(v) >= 0)
stage.subject_to(p >= 0)
stage.method(MultipleShooting(N=1, M=20, intg='rk'))

stage.subject_to(stage.at_t0(p) == 0)
ocp.subject_to(stage.t0 == 0)

stage_prev = stage

n_bounce = 2
for i in range(n_bounce):
    ocp.subject_to(stage.at_tf(p) == 0)
    stage = ocp.stage(stage_prev)
    ocp.subject_to(stage.at_t0(v) == -0.9 * stage_prev.at_tf(v))
    ocp.subject_to(stage.t0 == stage_prev.tf)

    stage_prev = stage

# Final bounce should reach 0.5 exactly
ocp.subject_to(stage.at_tf(v) == 0)
ocp.subject_to(stage.at_tf(p) == 0.5)

ocp.solver('ipopt')

# Solve
sol = ocp.solve()
```

References and further reading

- [1] Rawlings, J. B., Mayne, D. Q., & Diehl, M. M. (2017). *Model predictive control: theory, computation, and design* (2nd ed.). Nob Hill Publishing.
- [2] Vanroye, L., Sathya, A., De Schutter, J., & Decré, W. (2023). FATROP : A Fast Constrained Optimal Control Problem Solver for Robot Trajectory Optimization and Control. *ArXiv.Org*. https://doi.org/10.48550/arxiv.2303.16746
- [3] Wächter, A., Biegler, L. On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Math. Program.* 106, 25–57 (2006). https://doi.org/10.1007/s10107-004-0559-y