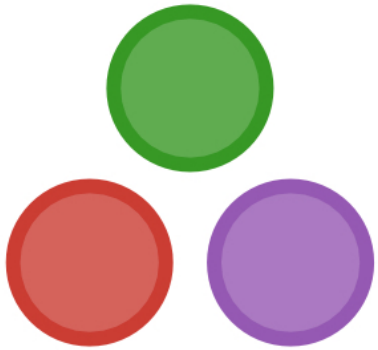
Solving optimal control problems with Julia

Jean-Baptiste Caillau, Olivier Cots, Joseph Gergaud, Pierre Martinon



Julia and Optimization Days 2023

Solving optimal control problems with Julia

Jean-Baptiste Caillau, Olivier Cots, Joseph Gergaud, Pierre Martinon, Sophia Sed

What it's about

- Nonlinear optimal control of ODEs:

$$g(x(t_0), x(t_f)) + \int_{t_0}^{t_f} f^0(x(t), u(t))\,\mathrm{d}t \to \min$$

subject to

$$\dot{x}(t) = f(x(t), u(t)), \quad t \in [t_0, t_f]$$

plus boundary, control and state constraints

- Our core interests: numerical & geometrical methods in control, applications

Where it comes from

- BOCOP: the optimal control solver
- HamPath: indirect and Hamiltonian pathfollowing

- Coupling direct and indirect solvers, examples

OptimalControl.jl

- Basic example: double integrator (1/3)
- Basic example: double integrator (2/3)
- Basic example: double integrator (3/3)
- Indirect simple shooting
- Advanced example: Goddard problem

Wrap up

- [X] High level modelling of optimal control problems
- [X] Efficient numerical resolution coupling direct and indirect methods
- [X] Collection of examples

Future

- ct_repl
- Additional solvers: direct shooting, collocation for BVP, Hamiltonian pathfollowing...
- ... and open to contributions!
- CTProblems.jl

control-toolbox.org

- Open toolbox
- Collection of Julia Packages rooted at OptimalControl.jl

control-toolbox

The control-toolbox ecosystem gathers `Julia` packages for mathematical control and applications. It is an outcome of a research initiative supported by the Centre Inria of Université Côte d'Azur and a sequel to previous developments, notably Bocop and Hampath. See also: ct gallery. The root package is `OptimalControl.jl` which aims to provide tools to solve optimal control problems by direct and indirect methods.

Documentation

doc OptimalControl.jl

Installation

See the installation page.

Partners

Université de Toulouse

UNIVERSITÉ CÔTE D'AZUR

Inria

cnrs

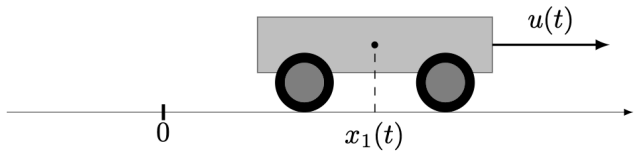
Credits (not exhaustive!)

- DifferentialEquations.jl
- JuMP, InfiniteOpt.jl, ADNLPModels.jl
- Ipopt
- JuliaDiff (FowardDiff.jl, Zygote.jl)
- MLStyle.jl
- REPLMaker.jl

« JuliaCon2023

Basic example

Let us consider a wagon moving along a rail, whom acceleration can be controlled by a force u. We denote by $x = (x_1, x_2)$ the state of the wagon, that is its position x_1 and its velocity x_2.



We assume that the mass is constant and unitary and that there is no friction. The dynamics we consider is given by

$$\dot{x}_1(t) = x_2(t), \quad \dot{x}_2(t) = u(t),, \quad u(t) \in \mathbb{R},$$

which is simply the double integrator system. Les us consider a transfer starting at time $t_0 = 0$ and ending at time $t_f = 1$, for which we want to minimise the transfer energy

$$\frac{1}{2} \int_0^1 u^2(t) \, \mathrm{d}t$$

starting from the condition $x(0) = (-1, 0)$ and with the goal to reach the target $x(1) = (0, 0)$.

Solution and details

See the page Double integrator: energy minimisation for the analytical solution and details about this problem.

First, we need to import the `OptimalControl.jl` package:

```
using OptimalControl
```

Then, we can define the problem

```
@def ocp begin
    t ∈ [ 0, 1 ], time
    x ∈ R², state
    u ∈ R, control
    x(0) == [ -1, 0 ]
    x(1) == [ 0, 0 ]
    ẋ(t) == [ x₂(t), u(t) ]
    ∫( 0.5u(t)^2 ) → min
end
```

Solve it

```
sol = solve(ocp)
```

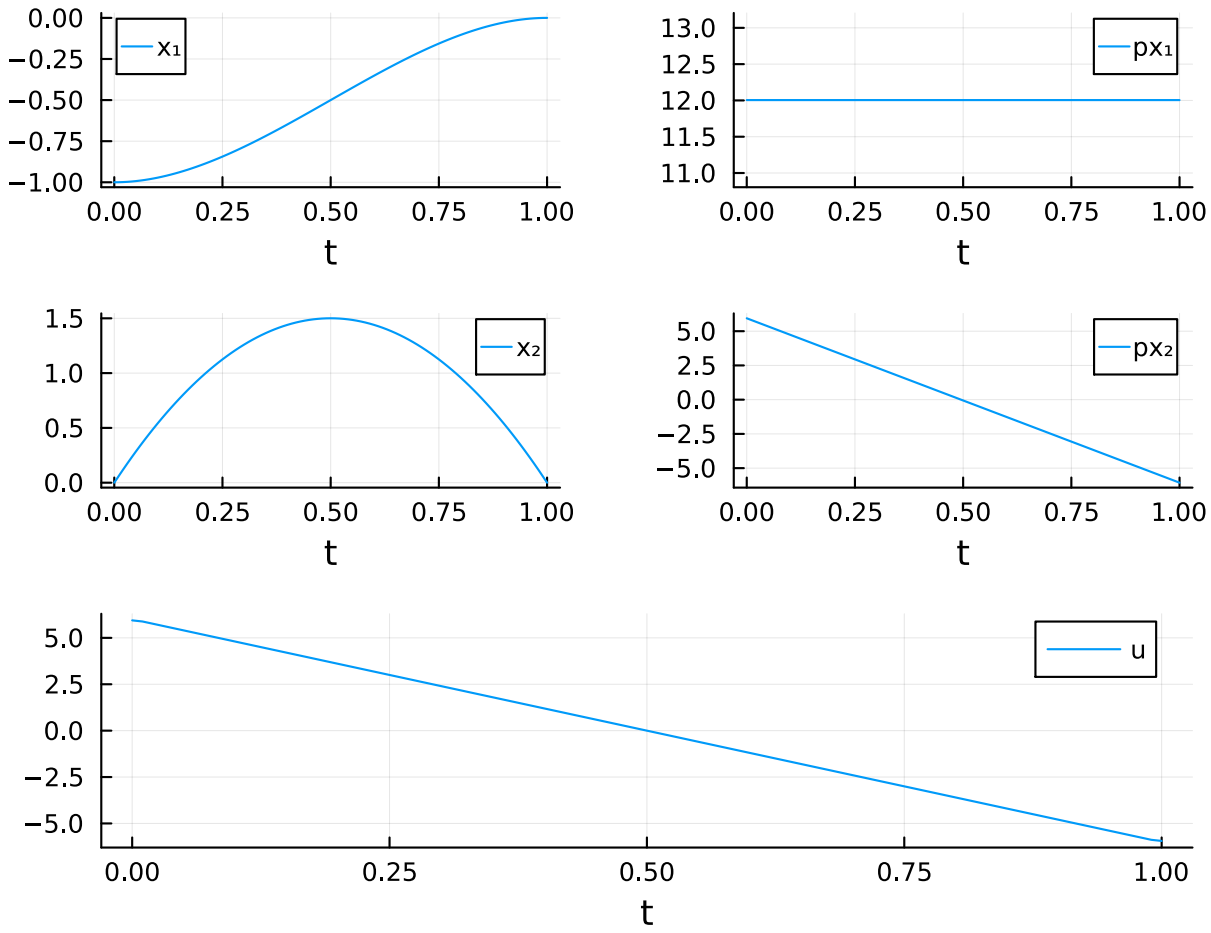
```
Method = (:direct, :adnlp, :ipopt)
This is Ipopt version 3.14.14, running with linear solver MUMPS 5.6.2.

Number of nonzeros in equality constraint Jacobian...:     1205
Number of nonzeros in inequality constraint Jacobian.:        0
```

```
Number of nonzeros in Lagrangian Hessian.............:      101

Total number of variables............................:      404
                     variables with only lower bounds:        0
                variables with lower and upper bounds:        0
                     variables with only upper bounds:        0
Total number of equality constraints.................:      305
Total number of inequality constraints...............:        0
        inequality constraints with only lower bounds:        0
   inequality constraints with lower and upper bounds:        0
        inequality constraints with only upper bounds:        0

iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
   0  1.0000000e-01 1.10e+00 1.92e-14   0.0 0.00e+00    -  0.00e+00 0.00e+00   0
   1 -5.0000000e-03 1.81e-01 1.78e-15 -11.0 6.04e+00    -  1.00e+00 1.00e+00h  1
   2  6.0023829e+00 8.88e-16 1.78e-15 -11.0 6.01e+00    -  1.00e+00 1.00e+00h  1

Number of Iterations....: 2

                                   (scaled)                 (unscaled)
Objective...............:   6.0023829460295719e+00    6.0023829460295719e+00
Dual infeasibility......:   1.7763568394002505e-15    1.7763568394002505e-15
Constraint violation....:   8.8817841970012523e-16    8.8817841970012523e-16
Variable bound violation:   0.0000000000000000e+00    0.0000000000000000e+00
Complementarity.........:   0.0000000000000000e+00    0.0000000000000000e+00
Overall NLP error.......:   1.7763568394002505e-15    1.7763568394002505e-15


Number of objective function evaluations             = 3
```

```
Number of objective gradient evaluations             = 3
Number of equality constraint evaluations            = 3
Number of inequality constraint evaluations          = 0
Number of equality constraint Jacobian evaluations   = 3
Number of inequality constraint Jacobian evaluations = 0
Number of Lagrangian Hessian evaluations             = 2
Total seconds in IPOPT                               = 0.022

EXIT: Optimal Solution Found.
```

and plot the solution

```
plot(sol, size=(600, 450))
```



« Introduction

Basic example (functional version) »

Goddard problem

Introduction

For this advanced example, we consider the well-known Goddard problem[1] [2] which models the ascent of a rocket through the atmosphere, and we restrict here ourselves to vertical (one dimensional) trajectories. The state variables are the altitude r, speed v and mass m of the rocket during the flight, for a total dimension of 3. The rocket is subject to gravity g, thrust u and drag force D (function of speed and altitude). The final time t_f is free, and the objective is to reach a maximal altitude with a bounded fuel consumption.

We thus want to solve the optimal control problem in Mayer form

$$r(t_f) \to \max$$

subject to the controlled dynamics

$$\dot{r} = v, \quad \dot{v} = \frac{T_{\max}\, u - D(r, v)}{m} - g, \quad \dot{m} = -u,$$

and subject to the control constraint $u(t) \in [0, 1]$ and the state constraint $v(t) \leq v_{\max}$. The initial state is fixed while only the final mass is prescribed.

Nota bene

The Hamiltonian is affine with respect to the control, so singular arcs may occur, as well as constrained arcs due to the path constraint on the velocity (see below).

Direct method

We import the `OptimalControl.jl` package:

```
using OptimalControl
```

We define the problem

```
t0 = 0      # initial time
r0 = 1      # initial altitude
v0 = 0      # initial speed
m0 = 1      # initial mass
vmax = 0.1  # maximal authorized speed
mf = 0.6    # final mass to target

@def ocp begin # definition of the optimal control problem

    tf ∈ R, variable
    t ∈ [ t0, tf ], time
    x ∈ R³, state
    u ∈ R, control
```

```
    r = x₁
    v = x₂
    m = x₃

    x(t0) == [ r0, v0, m0 ]
    m(tf) == mf,          (1)
    0 ≤ u(t) ≤ 1
    r(t) ≥ r0
    0 ≤ v(t) ≤ vmax

    ẋ(t) == F0(x(t)) + u(t) * F1(x(t))

    r(tf) → max

end;

# Dynamics
const Cd = 310
const Tmax = 3.5
const β = 500
const b = 2

F0(x) = begin
    r, v, m = x
    D = Cd * v^2 * exp(-β*(r - 1)) # Drag force
    return [ v, -D/m - 1/r^2, 0 ]
end

F1(x) = begin
```

```
    r, v, m = x
    return [ 0, Tmax/m, -b*Tmax ]
end
```

We then solve it

```
direct_sol = solve(ocp, grid_size=100)
```

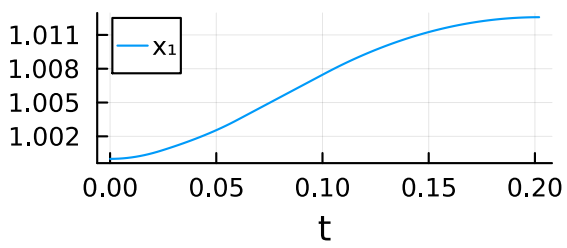
```
Method = (:direct, :adnlp, :ipopt)
This is Ipopt version 3.14.14, running with linear solver MUMPS 5.6.2.

Number of nonzeros in equality constraint Jacobian...:     1904
Number of nonzeros in inequality constraint Jacobian.:        0
Number of nonzeros in Lagrangian Hessian.............:     1111

Total number of variables............................:      405
                     variables with only lower bounds:      101
                variables with lower and upper bounds:      202
                     variables with only upper bounds:        0
Total number of equality constraints.................:      304
Total number of inequality constraints...............:        0
        inequality constraints with only lower bounds:        0
   inequality constraints with lower and upper bounds:        0
        inequality constraints with only upper bounds:        0

iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
   0 -1.0100000e+00 9.00e-01 2.00e+00   0.0 0.00e+00    -  0.00e+00 0.00e+00   0
   1 -1.0090670e+00 8.99e-01 6.67e+01   1.3 1.67e+02    -  3.64e-03 5.93e-04f  1
```

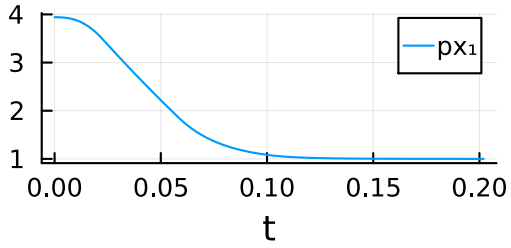
```
   2 -1.0000907e+00 8.74e-01 1.83e+02   1.0 6.64e+00    -  3.63e-02 2.83e-02h  1
   3 -1.0023670e+00 8.37e-01 1.34e+04   1.0 6.91e+00    -  2.11e-01 4.19e-02f  1
   4 -1.0025025e+00 7.70e-01 9.45e+03   1.5 4.04e+00    -  1.00e+00 8.09e-02f  1
   5 -1.0033626e+00 7.16e-01 1.48e+05   2.3 3.56e+00    -  3.73e-01 6.94e-02f  1
   6 -1.0142503e+00 9.62e-03 3.99e+04   2.3 7.16e-01    -  4.49e-01 9.90e-01h  1
   7 -1.0101264e+00 4.21e-03 4.24e+05   1.8 5.32e-01    -  5.03e-01 9.90e-01h  1
   8 -1.0068427e+00 3.20e-04 2.87e+06   0.9 2.44e-01    -  6.73e-01 9.91e-01h  1
   9 -1.0067336e+00 2.64e-06 2.30e+07   0.1 7.39e-02    -  7.07e-01 1.00e+00f  1
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
  10 -1.0067340e+00 1.13e-10 6.50e+05  -5.0 2.90e-04    -  9.89e-01 1.00e+00h  1
  11 -1.0067350e+00 2.81e-10 7.20e+03  -7.0 4.26e-04    -  9.89e-01 1.00e+00h  1
  12 -1.0078967e+00 9.07e-04 5.95e+03  -3.0 7.61e-01    -  6.55e-01 7.21e-01f  1
  13 -1.0081866e+00 3.67e-06 9.38e+03  -9.0 1.31e-02    -  8.60e-01 1.00e+00h  1
  14 -1.0091814e+00 1.57e-04 1.79e+02  -4.6 2.18e-01    -  1.00e+00 9.28e-01h  1
  15 -1.0105115e+00 2.55e-04 7.17e+02  -4.4 3.09e-01    -  1.00e+00 5.75e-01h  1
  16 -1.0114149e+00 2.34e-05 7.98e-04  -5.1 4.29e-02    -  1.00e+00 1.00e+00h  1
  17 -1.0122891e+00 7.90e-05 1.24e+02  -5.8 1.25e-01    -  9.98e-01 7.92e-01h  1
  18 -1.0125091e+00 2.83e-05 4.85e-04  -6.1 1.06e-01    -  1.00e+00 1.00e+00h  1
  19 -1.0125585e+00 1.21e-05 8.67e-05  -6.8 1.01e-01    -  1.00e+00 1.00e+00h  1
iter    objective    inf_pr   inf_du lg(mu)  ||d||  lg(rg) alpha_du alpha_pr  ls
  20 -1.0125675e+00 4.40e-06 3.01e-05  -7.3 9.80e-02    -  1.00e+00 1.00e+00h  1
  21 -1.0125707e+00 1.75e-06 3.70e-02  -8.0 1.14e-01    -  1.00e+00 9.91e-01h  1
  22 -1.0125714e+00 9.50e-07 4.98e-03  -8.6 1.37e-01    -  1.00e+00 9.94e-01h  1
  23 -1.0125716e+00 5.06e-08 6.91e-04  -9.3 2.57e-02    -  1.00e+00 9.96e-01h  1
  24 -1.0125716e+00 1.25e-10 1.08e-09 -11.0 2.15e-03    -  1.00e+00 1.00e+00h  1

Number of Iterations....: 24

                                   (scaled)                 (unscaled)
```

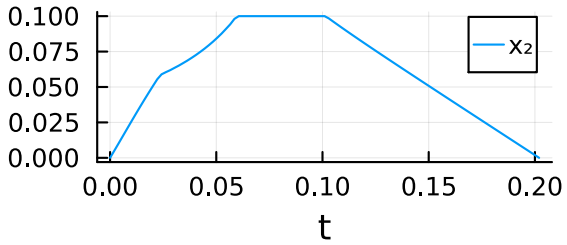
```
Objective...............:  -1.0125716188789131e+00   -1.0125716188789131e+00
Dual infeasibility......:   1.0759941386357690e-09    1.0759941386357690e-09
Constraint violation....:   1.5779155759787500e-11    1.2476877864209257e-10
Variable bound violation:   6.1594570555101313e-09    6.1594570555101313e-09
Complementarity.........:   1.5140175163848740e-11    1.5140175163848740e-11
Overall NLP error.......:   1.5779155759787500e-11    1.0759941386357690e-09


Number of objective function evaluations             = 25
Number of objective gradient evaluations             = 25
Number of equality constraint evaluations            = 25
Number of inequality constraint evaluations          = 0
Number of equality constraint Jacobian evaluations   = 25
Number of inequality constraint Jacobian evaluations = 0
Number of Lagrangian Hessian evaluations             = 24
Total seconds in IPOPT                               = 1.467

EXIT: Optimal Solution Found.
```

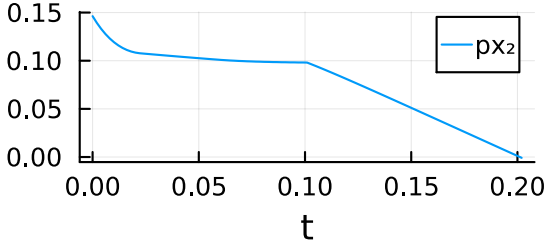
and plot the solution

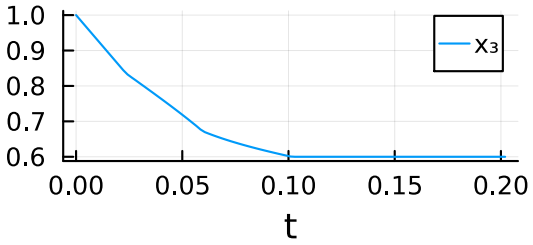
```
plt = plot(direct_sol, size=(600, 600))
```

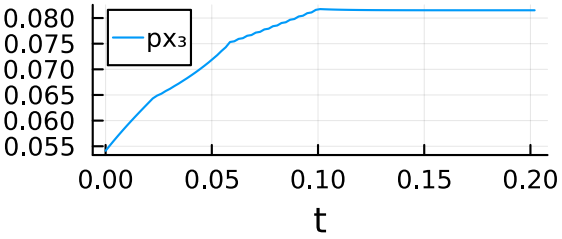

x₁
1.011
1.008
1.005
1.002
0.00
0.05
0.10
0.15
0.20
t

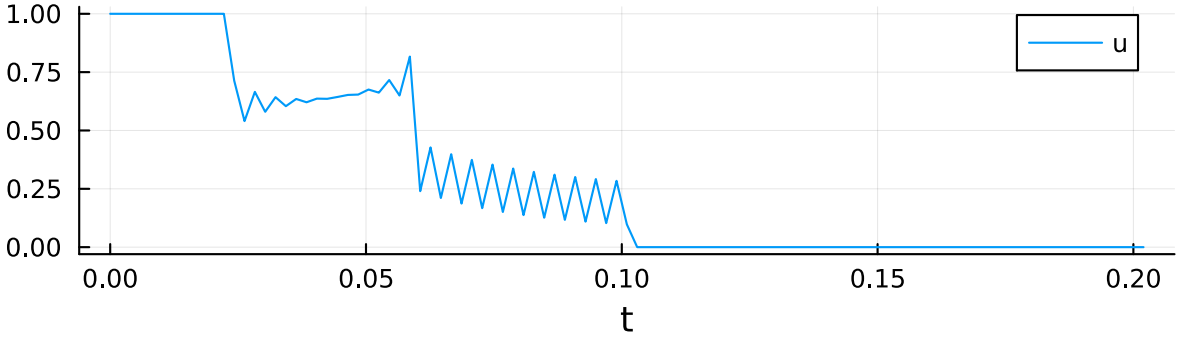

px₁
4
3
2
1
0.00
0.05
0.10
0.15
0.20
t


x₂
0.100
0.075
0.050
0.025
0.000
0.00
0.05
0.10
0.15
0.20
t


px₂
0.15
0.10
0.05
0.00
0.00
0.05
0.10
0.15
0.20
t


x₃
1.0
0.9
0.8
0.7
0.6
0.00
0.05
0.10
0.15
0.20
t

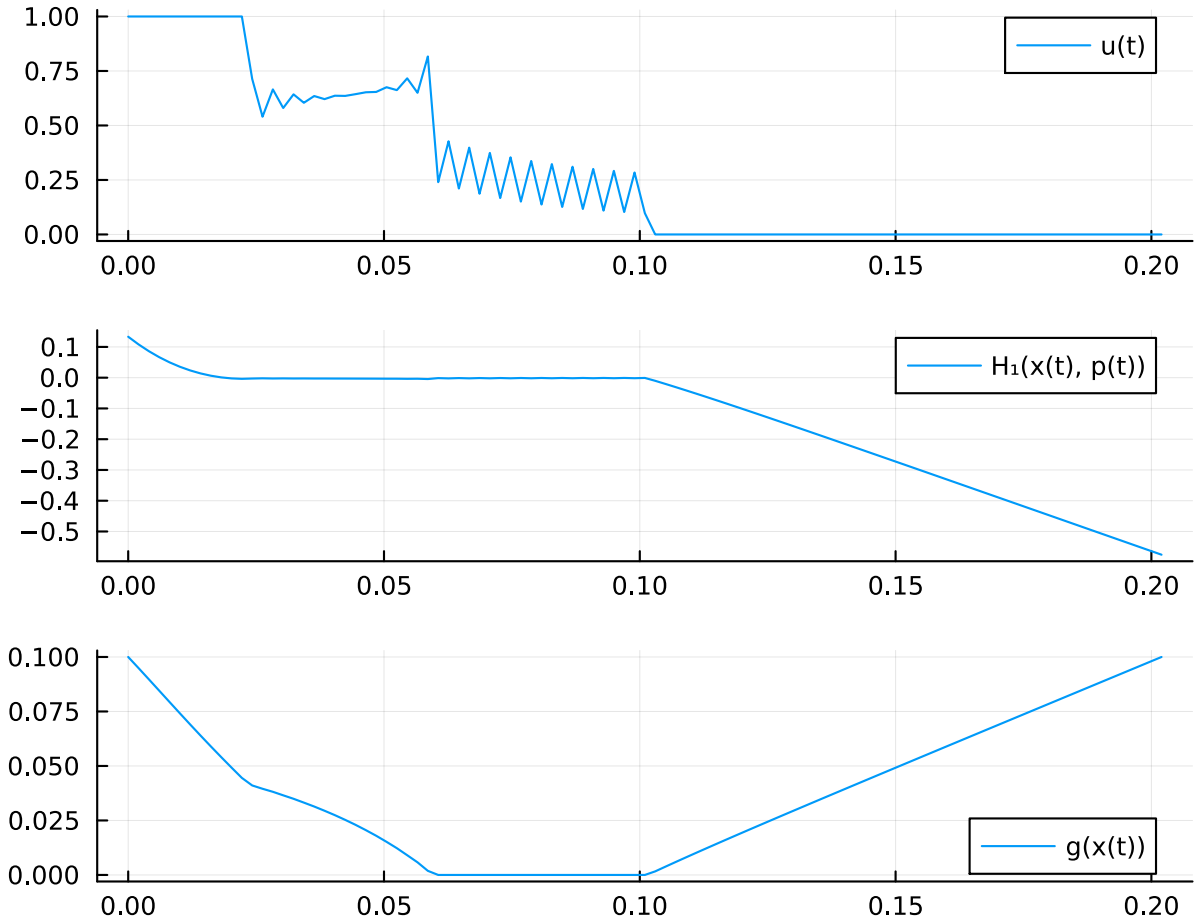

px₃
0.080
0.075
0.070
0.065
0.060
0.055
0.00
0.05
0.10
0.15
0.20
t


u
1.00
0.75
0.50
0.25
0.00
0.00
0.05
0.10
0.15
0.20
t

Indirect method

We first determine visually the structure of the optimal solution which is composed of a bang arc with maximal control, followed by a singular arc, then by a boundary arc and the final arc is with zero control. Note that the switching function vanishes along the singular and boundary arcs.

```
t = direct_sol.times
x = direct_sol.state
u = direct_sol.control
p = direct_sol.costate

H1 = Lift(F1)           # H1(x, p) = p' * F1(x)
φ(t) = H1(x(t), p(t))   # switching function
g(x) = vmax - x[2]      # state constraint v ≤ vmax

u_plot  = plot(t, u,     label = "u(t)")
H1_plot = plot(t, φ,     label = "H₁(x(t), p(t))")
g_plot  = plot(t, g ∘ x, label = "g(x(t))")

plot(u_plot, H1_plot, g_plot, layout=(3,1), size=(600,450))
```



We are now in position to solve the problem by an indirect shooting method. We first define the four control laws in feedback form and their associated flows. For this we need to compute some Lie derivatives, namely Poisson brackets of Hamiltonians (themselves obtained as lifts to the cotangent bundle of vector fields), or derivatives of functions along a vector field. For instance, the control along the *minimal order* singular arcs is obtained as the quotient

$$u_s = -\frac{H_{001}}{H_{101}}$$

of length three Poisson brackets:

$$H_{001} = \{H_0, \{H_0, H_1\}\}, \quad H_{101} = \{H_1, \{H_0, H_1\}\}$$

where, for two Hamiltonians H and G,

$$\{H, G\} := (\nabla_p H | \nabla_x G) - (\nabla_x H | \nabla_p G).$$

While the Lie derivative of a function f *wrt.* a vector field X is simply obtained as

$$(X \cdot f)(x) := f'(x) \cdot X(x),$$

and is used to the compute the control along the boundary arc,

$$u_b(x) = -(F_0 \cdot g)(x) / (F_1 \cdot g)(x),$$

as well as the associated multiplier for the *order one* state constraint on the velocity:

$$\mu(x, p) = H_{01}(x, p) / (F_1 \cdot g)(x).$$

Poisson bracket and Lie derivative

The Poisson bracket $\{H, G\}$ is also given by the Lie derivative of G along the Hamiltonian vector field $X_H = (\nabla_p H, -\nabla_x H)$ of H, that is

$$\{H, G\} = X_H \cdot G$$

which is the reason why we use the `@Lie` macro to compute Poisson brackets below.

With the help of the differential geometry primitives from CTBase.jl, these expressions are straightforwardly translated into Julia code:

```
# Controls
u0 = 0                                  # off control
u1 = 1                                  # bang control

H0 = Lift(F0)                           # H0(x, p) = p' * F0(x)
H01  = @Lie { H0, H1 }
H001 = @Lie { H0, H01 }
H101 = @Lie { H1, H01 }
us(x, p) = -H001(x, p) / H101(x, p)     # singular control

ub(x) = -(F0•g)(x) / (F1•g)(x)          # boundary control
μ(x, p) = H01(x, p) / (F1•g)(x)         # multiplier associated to the state constraint g

# Flows
f0 = Flow(ocp, (x, p, tf) -> u0)
f1 = Flow(ocp, (x, p, tf) -> u1)
fs = Flow(ocp, (x, p, tf) -> us(x, p))
fb = Flow(ocp, (x, p, tf) -> ub(x), (x, u, tf) -> g(x), (x, p, tf) -> μ(x, p))
```

Then, we define the shooting function according to the optimal structure we have determined, that is a concatenation of four arcs.

```
x0 = [ r0, v0, m0 ] # initial state
```

```
function shoot!(s, p0, t1, t2, t3, tf)

    x1, p1 = f1(t0, x0, p0, t1)
    x2, p2 = fs(t1, x1, p1, t2)
    x3, p3 = fb(t2, x2, p2, t3)
    xf, pf = f0(t3, x3, p3, tf)

    s[1] = constraint(ocp, :eq1)(x0, xf, tf) - mf # final mass constraint (1)
    s[2:3] = pf[1:2] - [ 1, 0 ]                   # transversality conditions
    s[4] = H1(x1, p1)                             # H1 = H01 = 0
    s[5] = H01(x1, p1)                            # at the entrance of the singular arc
    s[6] = g(x2)                                  # g = 0 when entering the boundary arc
    s[7] = H0(xf, pf)                             # since tf is free

end
```

To solve the problem by an indirect shooting method, we then need a good initial guess, that is a good approximation of the initial costate, the three switching times and the final time.

```
η = 1e-3
t13 = t[ abs.(φ.(t)) .≤ η ]
t23 = t[ 0 .≤ (g ∘ x).(t) .≤ η ]
p0 = p(t0)
t1 = min(t13...)
t2 = min(t23...)
t3 = max(t23...)
tf = t[end]

println("p0 = ", p0)
```

```
println("t1 = ", t1)
println("t2 = ", t2)
println("t3 = ", t3)
println("tf = ", tf)

# Norm of the shooting function at solution
using LinearAlgebra: norm
s = similar(p0, 7)
shoot!(s, p0, t1, t2, t3, tf)
println("Norm of the shooting function: ‖s‖ = ", norm(s), "\n")
```

```
p0 = [3.9420913558122708, 0.14627140398405294, 0.05411785715966011]
t1 = 0.01817850265880542
t2 = 0.06059500886268473
t3 = 0.07877351152149016
tf = 0.20198336287561577
Norm of the shooting function: ‖s‖ = 3.249855374403743
```

Finally, we can solve the shooting equations thanks to the MINPACK solver.

```
using MINPACK                                                  # NLE solver

nle = (s, ξ) -> shoot!(s, ξ[1:3], ξ[4], ξ[5], ξ[6], ξ[7])     # auxiliary function
                                                               # with aggregated inputs
ξ = [ p0 ; t1 ; t2 ; t3 ; tf ]                                 # initial guess
indirect_sol = fsolve(nle, ξ)                                  # resolution of S(ξ) = 0

# we retrieve the costate solution together with the times
```

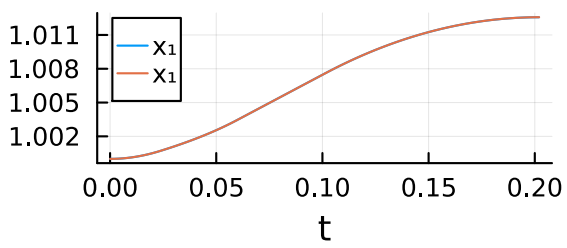
```
p0 = indirect_sol.x[1:3]
t1 = indirect_sol.x[4]
t2 = indirect_sol.x[5]
t3 = indirect_sol.x[6]
tf = indirect_sol.x[7]

println("p0 = ", p0)
println("t1 = ", t1)
println("t2 = ", t2)
println("t3 = ", t3)
println("tf = ", tf)

# Norm of the shooting function at solution
s = similar(p0, 7)
shoot!(s, p0, t1, t2, t3, tf)
println("Norm of the shooting function: ‖s‖ = ", norm(s), "\n")
```

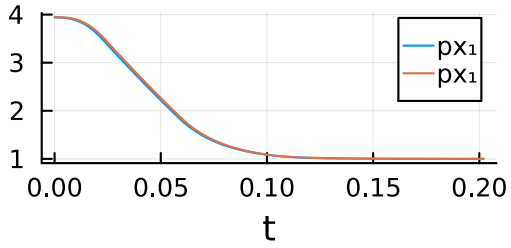
```
p0 = [3.945764658694955, 0.15039559623494775, 0.053712712941714945]
t1 = 0.023509684039930347
t2 = 0.059737380905787986
t3 = 0.1015713484236716
tf = 0.20204744057161464
Norm of the shooting function: ‖s‖ = 1.5775667383975246e-10
```

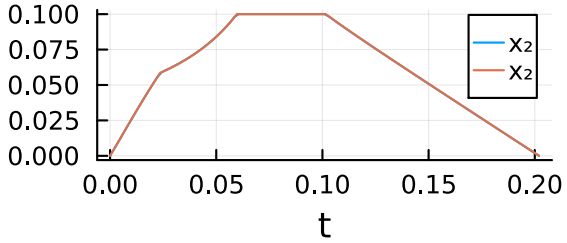
We plot the solution of the indirect solution (in red) over the solution of the direct method (in blue).

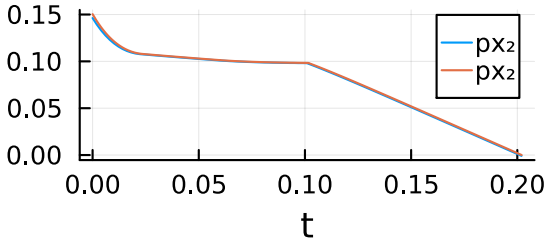
```
f = f1 * (t1, fs) * (t2, fb) * (t3, f0) # concatenation of the flows
flow_sol = f((t0, tf), x0, p0)          # compute the solution: state, costate, control...
```

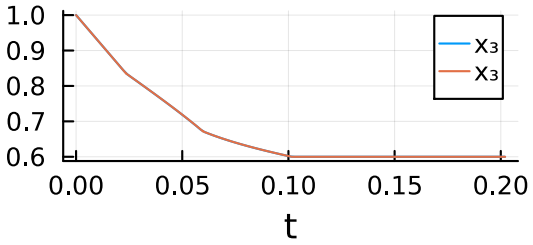
```
plot!(plt, flow_sol)
```

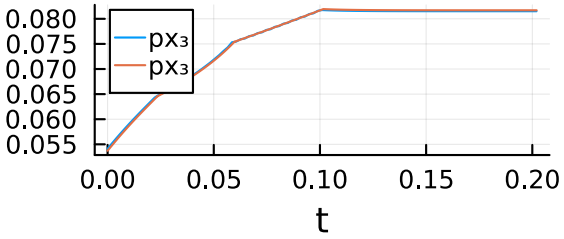

1.011
1.008
1.005
1.002
x₁
x₁
0.00
0.05
0.10
0.15
0.20
t

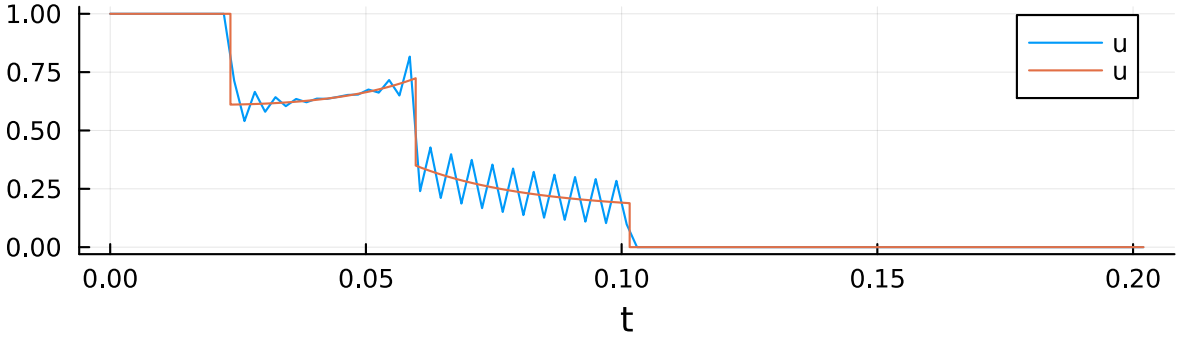

4
3
2
1
px₁
px₁
0.00
0.05
0.10
0.15
0.20
t


0.100
0.075
0.050
0.025
0.000
x₂
x₂
0.00
0.05
0.10
0.15
0.20
t


0.15
0.10
0.05
0.00
px₂
px₂
0.00
0.05
0.10
0.15
0.20
t


1.0
0.9
0.8
0.7
0.6
x₃
x₃
0.00
0.05
0.10
0.15
0.20
t


0.080
0.075
0.070
0.065
0.060
0.055
px₃
px₃
0.00
0.05
0.10
0.15
0.20
t


1.00
0.75
0.50
0.25
0.00
u
u
0.00
0.05
0.10
0.15
0.20
t

References

- [1] R.H. Goddard. A Method of Reaching Extreme Altitudes, volume 71(2) of Smithsonian Miscellaneous Collections. Smithsonian institution, City of Washington, 1919.
- [2] H. Seywald and E.M. Cliff. Goddard problem in presence of a dynamic pressure limit. Journal of Guidance, Control, and Dynamics, 16(4):776–781, 1993.

« Double integrator: time minimisation

Indirect simple shooting »