Accurate Simulation for Numerical Optimal Control

Viktor Leek, Lars Eriksson

Division of Vehicular Systems, Linköping University, Sweden
viktor.leek@liu.se, lars.eriksson@liu.se

Abstract

Accurate simulation of the numerical optimal control in software environments where call to simulation routines is explicit, for instance Matlab and SciPy. A discussion on the simulation aspects of numerical optimal control, how it may fail, and how such erroneous results can be detected using accurate simulation. The key contribution is how to accurately include a piecewise constant control input in the simulations, which is discussed in detail, including code examples. The technique is demonstrated on an example problem which show how simulation can be used to analyze optimal control problems with uncertainty, but also demonstrates how erroneous simulation may lead to erroneous conclusions.

Keywords: Simulation, Optimal control, direct multiple shooting, direct collocation

1 Introduction

Numerical optimal control (NOC) is the field devoted to solving optimal control problems (OCPs) numerically. There exist several approaches that can be divided into three main categories: State-space methods, indirect methods, and direct methods. An overview is found in (Rao, 2009). Here, the focus is on *direct methods*, which are characterized by first discretizing the OCP, in a process known as *transcription*, into a parameter optimization problem, and then solved numerically. Often the parameter optimization problem is a nonlinear program (NLP), but may very well be a quadratic program, or some other suitable problem class.

Numerical solution to differential equations, the field underpinning simulation, has been actively researched for well over a century (Butcher, 1996). Too vast to enumerate all relevant publications, a good introduction to the subject can be found in the textbook (Ascher and Petzold, 1998).

Simulation enters NOC is many different ways. It is the key component in the transcription process but can also be used for other purposes. For instance, as a mean of providing an initial guess, as a mean to post-analyze the results, or the results are already from the beginning intended to be used in a simulation, for instance as input to a more complex representation of the system, or as a feedforward/reference signal. It also plays an important role in fine-tuning the transcription process, as it aids the user in selecting a suitable integration method that balance accuracy and execution time.

Perhaps the most important enabler of modern NOC was the release of software package Casadi (Andersson, 2013; Andersson et al., 2019). It provides the building blocks necessary for the user to implement a custom transcription method. The benefit is that the user can formulate and solve a large class of relevant OCPs. The drawback is that the user needs to implement the method themselves from essential building blocks, which is not trivial and therefore opens up for potential pitfalls.

The most vulnerable part of simulating the numerical optimal control is inclusion of a piecewise constant control input. It is well established that discontinuities make simulation more difficult (Shampine et al., 1976), and various solutions can be found (Gear and Osterby, 1984; Majer et al., 1995; Mao and Petzold, 2002). In the field of NOC, much attention is given to how simulation and numerical integration enters NOC (von Stryk and Bulirsch, 1992; Diehl et al., 2006; Betts, 2010; Biegler, 2010; Rawlings et al., 2017), but little attention is given to how NOC enters simulation. An important enabler of NOC in simulation is therefore an explicit investigation of how to handle the incurred difficulties, in particular, an explicit description of handling piecewise constant control inputs, which is the subject here.

The main contribution is how to structure simulations that involve NOC results based on a piecewise constant control input model, and why they should be structured in that way. Secondary contributions include demonstrating how integration in NOC can produce erroneous results and how accurate simulation can help detect that.

The outline is as follows. Section 2 introduces the subject by an example, Section 3 develops a user's model of the transcription process, Section 4 shows how to accurately simulate the optimal control, Section 5 gives an advanced use case in which Monte-Carlo simulation and optimal control is used to test a method for finding robust optimal trajectories, and the conclusions are presented in Section 6.

2 An introductory example

To introduce the subject of simulation in NOC, a simple example has been devised. The purpose is to show the importance of adequately selecting the integration method in the OCP transcription, but also the importance of an accurate simulation of the results.

Begin by denoting the state variable by x, the control

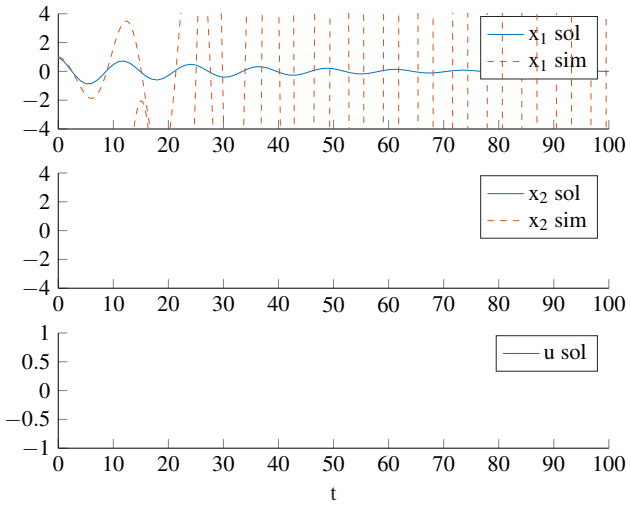

Figure 1. Numerical solution to (2) using implicit euler to discretize the continuous time dynamics. Solid lines (blue) show the numerical solution as obtained from the optimization problem solver. Dashed lines (red) show a validation of the results when the system is driven with the optimal control as input. Y-axis is limited.

input by u, the output by y, and consider the linear system

$$\dot{x} = \begin{bmatrix} \frac{1}{5} & -\frac{13}{50} \\ 1 & 0 \end{bmatrix} x + \begin{bmatrix} 1 \\ 0 \end{bmatrix} u \tag{1a}$$

$$y = \begin{bmatrix} 0 & 1 \end{bmatrix} x \tag{1b}$$

Omitting the control input ($u = 0$), the system is unstable and has poles $1/10 \pm 1/2i$.

Assume it is desired to solve the following optimal control problem, in which the above system dynamics is denoted $\dot{x} = f(x,u)$:

$$\begin{aligned} \min_{u} \quad & \int_0^{t_f} x^{\mathrm{T}} Q x + u^{\mathrm{T}} R u \, dt \\ \text{s.t.} \quad & \dot{x} = f(x,u), \\ & x(0) = x_0, \\ & x(t_f) = x_f \end{aligned} \tag{2}$$

The aim is to drive the state from the initial value x_0, to the final value x_f, while minimizing the quadratic criterion in the objective function, and doing so within the fixed time horizon $t \in [0, t_f]$. To solve the above problem using numerical optimal control a transcription method is needed. Here, the direct collocation method (Hargraves and Paris, 1987) using *implicit Euler* for numerical integration is used.

For the parametrization

$$Q = \begin{bmatrix} 0.1 & 0 \\ 0 & 0.1 \end{bmatrix}, \; R = 1000, \; t_f = 100,$$
$$x_0 = [1,1]^{\mathrm{T}}, \; x_f = [0,0]^{\mathrm{T}}$$

Figure 1 shows, drawn using solid lines (blue), a numerical solution to the problem.

The results are counter intuitive. The system dynamics are unstable, and yet the control input is close to zero. To confirm the results, the system is simulated using the Matlab routine `ode45` (default settings), which is an implementation of Dorman-Prince method (Dormand and Prince, 1980). The results are drawn using dashed lines (red) in Figure 1. It shows that the system is diverging, and not at all driven to the origin, as indicated by the numerical solution to the optimal control problem.

2.1 Analysis

The problem in the above example is that the stability properties of system (1) is not preserved by the implicit Euler method. To analyze the situation, and better appreciate the importance of simulation in numerical optimal control, it is demonstrated with which ease the problem can be constructed.

Consider the system, $\dot{x} = Ax$, where A is a diagonalizable, constant coefficient, $m \times m$ matrix. Take T as a nonsingular matrix consisting of eigenvectors of A. A is then diagonalizable as $A = TDT^{-1}$, where $D = \mathrm{diag}(\lambda_1, \lambda_2, \ldots, \lambda_m)$, is a diagonal matrix with diagonal entries $\lambda_i, i = 1,2,\ldots,m$, being the eigenvalues of A. By introducing the change of variable $z = T^{-1}x$, the decoupled dynamics is obtained as $\dot{z} = Dz$. Applying implicit Euler over the fixed grid

$$0 = t_0 < t_1 < \cdots < t_{N-1} < t_N = t_f \tag{3a}$$

$$h = t_{n+1} - t_n, \; n = 0,1,\ldots,N-1 \tag{3b}$$

for some number of steps N, the discretized dynamics is obtained as $z_{n+1} = (I - hD)^{-1} z_n$. Every component $z^{(i)}$ is of the form

$$z_{n+1}^{(i)} = \frac{1}{1 - h\lambda_i} z_n^{(i)}, \; i = 1,2,\ldots,m$$

For $|1 - h\lambda_i| > 1, i = 1,2,\ldots,m$, a converging sequence $|z_0^{(i)}| > |z_1^{(i)}| > \cdots > |z_n^{(i)}|$ is obtained, regardless of the stability properties of the underlying ODE. So, by construction the matrix from the desired eigenvalues and step size, an unstable ODE, whose stability property is not preserved by the implicit Euler method, can be constructed.

In the example, the poles are $1/10 \pm 1/2i$ and the step length $h = 1$, which gives $|1 - h\lambda_i| = 1 + 37/1250 > 1$.

These types of traps and pitfalls are to be found in simulation, especially when working with simple integration methods, as is typically done in NOC. It shows the importance of properly understanding the type of problem one is simulating, but perhaps even more, the importance of confirming the results by use of simulation.

2.2 Instability

Unstable systems are difficult to simulate, even when control is applied. To demonstrate that the above problem is

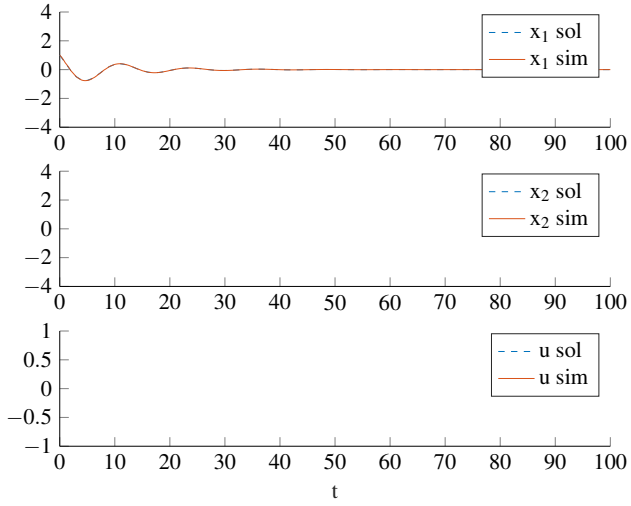

Figure 2. Numerical solution to (2). Drawn using dashed lines (blue) is the results from the OCP solver, using a high order integration method. Drawn in solid lines is the simulation of the results using a different high order integration method.

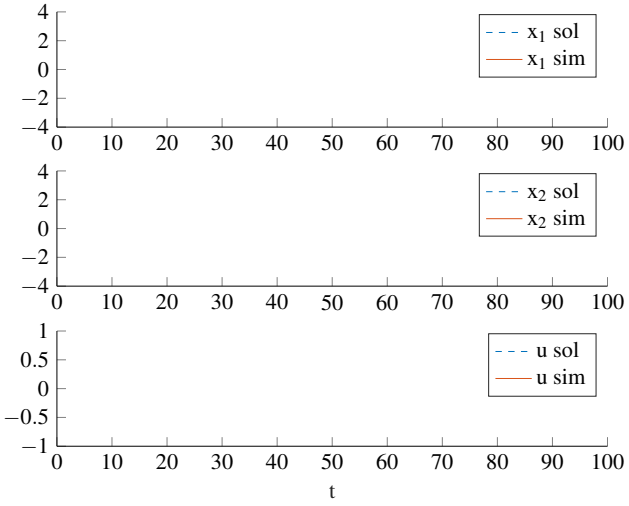

Figure 3. Dashed lines show the actual trajectory, obtained using a correct simulation setup. Solid lines show a faulty confirmation, obtained using the same integration method, with the same default settings as the dashed line, but with a faulty setup. Notice that the control signals are *almost* identical. The perturbation is still large enough to put the system on a completely different trajectory.

simulatable, the OCP (2) is solved using a fifth order Legendre collocation integration method and simulated using `ode45` in Matlab (it can be noted that this is overkill for a linear system). The results are found in Figure 2. Contrary to the first case, the results are consistent with intuition and control is applied forcefully in the beginning to prevent the system from diverging.

2.3 Erroneous simulation

Simulating the numerical results of an optimal control problem is not as trivial as it first appears. Consider Figure 3. Drawn in dashed (blue) is the same solution presented in Figure 2. Drawn in solid (red) is an erroneous simulation of the optimal control. The integration method used to obtain the simulation results presented in Figure 2 and Figure 3 is the same, the Matlab command `ode45`, using default settings in both cases. One trajectory is diverging, the other converging. The difference is in how the simulation is structured, and how to do that accurately is shown in Section 4.

3 Numerical optimal control

In order to structure the simulations correctly, it is necessary to develop a user's model of the transcription process. This section briefly introduces the subject of optimal control and outlines a sufficiently detailed model of the transcription process to structure the simulations.

3.1 Optimal control

A solution to an optimal control problem is seeking the optimal control, u^*, and the optimal state trajectory, x^*, that minimize the cost function and does not violate the constraints. The problem's characteristic feature is the differential constraint $\dot{x} = f(t,x,u)$. The objective function consists of two parts. An integral cost $\int L(t,x,u)\,dt$, and a terminal cost $E(t_f,x(t_f))$. There is an allowable set for the initial value $x(0) \in \mathscr{X}_0$, path constraints $x(t) \in \mathscr{X}$ and $u(t) \in \mathscr{U}$, and an allowable set for the terminal state $x(t_f) \in \mathscr{X}_f$. The problem is formulated as

$$\begin{aligned} \min_{u} \quad & E(t_f,x(t_f)) + \int_0^{t_f} L(t,x,u)\,dt \\ \text{s.t.} \quad & \dot{x} = f(t,x,u),\ t \in [0,t_f], \\ & x(0) \in \mathscr{X}_0, \\ & x(t) \in \mathscr{X},\ t \in [0,t_f], \\ & u(t) \in \mathscr{U},\ t \in [0,t_f], \\ & x(t_f) \in \mathscr{X}_f \end{aligned}$$

3.2 Direct methods for optimal control

Any numerical method for optimal control needs to address the fact that the system of interest is represented by a differential equation. While there are many available methods, the focus here is on *direct methods*, which transcribes the OCP into an NLP, and solves that numerically.

To transcribe the continuous time OCP into an NLP, there are two major considerations to take into account. How to handle the control input, and how to handle the integration of the system dynamics. There is also a third one, the integration of the integral cost $I = \int L(t,x,u)\,dt$. However, by introducing the integration state x_l, $\dot{x}_l = L(t,x,u)$, the integral can be integrated with the system dynamics and rephrased as an terminal cost $I = \int L(t,x,u)\,dt = x_l(t_f)$, and therefore the two main concerns are the state and control trajectory.

Consider again the fixed grid (3). Over it, the control input is parameterized as a piecewise constant signal, with a constant input, u_n, for each step

$$u(t) = u_n,\ t \in [t_n, t_{n+1})$$

While there are several ways of parameterizing the control input (Andersson, 2013), this is the most popular (Diehl, 2011) and corresponds to a zero-order hold control system implementation. It is also more general than it first appears. Consider the following augmentation. Denote the augmented system state by $\tilde{x}$ and augmented control signal by $\tilde{u}$ and let them be defined by

$$\tilde{x} = \begin{bmatrix} x \\ u \end{bmatrix},\ \dot{\tilde{x}} = \begin{bmatrix} f(t,x,u) \\ \tilde{u} \end{bmatrix},\ \tilde{u} = \frac{du}{dt}$$

Given that $\tilde{u}$ is piecewise constant, it then follows that u is piecewise linear. Since this augmentation can be applied arbitrarily many times, it shows that u can be of arbitrarily high order, even if the augmented input is piecewise constant. It should be noted that in a numerical setting, the obtained degree is also influenced by the order of the integration method.

The two most popular direct methods (Diehl et al., 2006), *direct collocation* (Hargraves and Paris, 1987), and *direct multiple shooting* (Bock and Plitt, 1984), integrate the system dynamics over each segment of the grid separately, forming a discontinuous trajectory, consisting of a sequence of initial value problems

$$\dot{x} = f(t,x,u_n),\ t \in [t_n, t_{n+1}),\ n = 0,\cdots,N-1 \quad (4a)$$
$$x(t_n) = x_n \quad (4b)$$

which is parameterized by the initial condition x_n, and state at the terminal boundary by $x(t_N) = x_N$.

To obtain a continuous trajectory, continuity constraints are introduced that bind the trajectory together

$$F(t_n, x_n, u_n) = x_{n+1},\ n = 0,\ldots,N-1 \quad (5)$$

Here, F is the numerical integration of the continuous time dynamics over the segment

$$F(t_n, x_n, u_n) \approx \int_{t_n}^{t_{n+1}} f(t,x,u_n)\,dt$$

A sketch of the process is found in Figure 4.

Omitting the path constraints and only considering box constraints for the initial and terminal constraints, the transcription process results in the NLP

$$\begin{aligned} \min_{\substack{x_0,\ldots,x_N \\ u_0,\ldots,u_{N-1}}} \quad & E(t_f, x_N) + x_{l,N} \\ \text{s.t.} \quad & \begin{bmatrix} x_{0,\min} \\ x_{f,\min} \end{bmatrix} \le \begin{bmatrix} x_0 \\ x_N \end{bmatrix} \le \begin{bmatrix} x_{0,\max} \\ x_{N,\max} \end{bmatrix}, \\ & \begin{bmatrix} F(t_0,x_0,u_0) - x_1 \\ \vdots \\ F(t_{N-1},x_{N-1},u_{N-1}) - x_N \end{bmatrix} = 0 \end{aligned} \quad (6)$$

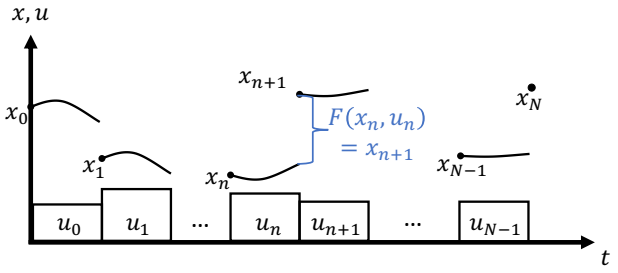

Figure 4. Sketch of a direct method for optimal control. The control input u is piecewise constant, and the state trajectory is discontinuous and parameterized with an initial value for every segment. The state trajectory is made constant by introducing the continuity constraint (5).

As a user's model, the two most important things to note is that the control input can be expected to be piecewise constant, and the state trajectory is integrated separately over the grid segments.

It should also be emphasized that this is an outline of the transcription process, and not a formal description, which can be found in (Biegler, 2010; Betts, 2010).

4 Simulation of the optimal control

The main difficulty in simulating a system with the optimal control as input is handling the piecewise constant control input. Consider the following basic IVP where the intention is to simulate the optimal control. $\dot{x} = h(t,x)$ is used to describe the simulated system, which is a combination of the controlled system and a look-up of the optimal control.

$$\dot{x} = h(t,x),\ t \in [0, t_f] \quad (7a)$$
$$x(0) = x_0 \quad (7b)$$

The controlled system, $\dot{x} = f(t,x,u)$, is contained in $\dot{x} = h(t,x)$. Since the simulated solution is expressed in the two variables t and x, then so is the simulated control, u^s. By introducing the definition $u^s = g(t,x)$, the simulated system can be described by $\dot{x} = h(t,x) = f(t,x,g(t,x))$. The transcription process gives that the optimal control is parameterized in terms of t, so the simulated control is a function in the independent variable only, $u^s = g(t)$. This means that formulation (7) include a look-up of u^* based on the independent variable t. If simulated correctly, $u^s = u^*$, but it is not necessarily so.

The following Matlab code is an example simulation of IVP (7). It uses interpolation based on the current value of the independent variable to look-up the control input. `t` and `x` are the variables the hold the simulation results, `h` is the function that is called by the integration routine `ode45`, `[t0, tf]` is the problem horizon, `x0` is the initial value, `interp1` does piecewise constant interpolation of the optimal control, (`t_sol`, `u_sol`), based on the function parameter `t` which represent the independent

variable, `f` is the function that holds the implementation of the controlled system $f(t,x,u)$, and `der` is the return value that represent $\dot{x}(t)$.

```
[t, x] = ode45(@h, [t0, tf], x0);
function der = h(t, x)
    u = interp1(...
        t_sol, u_sol, t, 'previous');
    der = f(t, x, u);
end
```

This is an example of a naive implementation, and was used to derive the erroneous confirmation in Figure 3 (solid lines). Since the control is interpolated based on the independent variable, the actual control input is dependent on the step length. Unless steps are taken at the exact grid points (3), the simulated control u^s does not equal the optimal control u^*. The problem is remedied by using a variable step-length solver and lowering the tolerance, but it does not solve it. Variable step-length solvers are built on the assumption of a smooth solution, differentiable up to the order conditions, which is why discontinuities are handled poorly (Ascher and Petzold, 1998). Also, the control input is not a dependent variable, so local error control does not have direct mean of estimating the error in that signal. A secondary effect of lowering the tolerance is that the number of function evaluations increases, which increase computational time, so both accuracy and time is lost when structuring the simulation according to (7).

From a user's perspective, an aggravating circumstance of simulating (7) is that it is easy to miss that the optimal control has not been properly reconstructed, since the simulated control, u^s, needs to be reconstructed from the simulation (it is not a state). One might think of using a global variable to store the control, but this approach does not work for variable step-length solvers as not all steps are accepted.

4.1 Event functions

(Ascher and Petzold, 1998; Gustafsson) suggest the use of event functions to tell the integration method that a discontinuous event has occurred. While being a good advice in general, caution needs to be taken. For instance, Matlab (Shampine and Reichelt, 1997) and SciPy (Virtanen et al., 2020) does not use event functions as a mean to direct step length. In other environments such as OpenModelica ans Assimulo (Lundvall et al., 2005; Andersson et al., 2015) events can be used to inform the solver of discontinuities.

4.2 Handling of the control input

To handle the problem of a discontinuous control input in the simulation, the simulation is restarted for every change in the control input, similar to what is done in the transcription process (4)

$$\dot{x} = f(t,x,u_n),\ t \in [t_n, t_{n+1}],\ n = 0,\cdots,N-1 \quad (8a)$$
$$x(t_n) = x_n \quad (8b)$$

This avoids the problem of having to localize the change in control input and ensures $u^s = u^*$, regardless of solver tolerance.

The following is an example Matlab simulation of (8). The code is structured based on the number of segments to simulate, `N`. The first loop iterate is peeled off in order to initialize the variables `t` and `x` which hold the solution. The local solution on each segment (`tt`, `xx`) is appended to the existing solution, without overlap. Notice how the control, `u_sol`, is fixed over each segment `[t_sol(i), t_sol(i+1)]`. The standard Matlab solver `ode45` is used, but it could be any suitable method.

```
f1 = @(t, x) f(t, x, u_sol(1));
[t, x] =  ode45(f1, [t0, tf], x0);
for i=2:N
    fi = @(t, x) f(t, x, u_sol(i));
    [tt, xx] = ode45(fi, [t_sol(i),...
        t_sol(i+1)], x(end,:));
    t = [t; tt(2:end)];
    x = [x; xx(2:end,:)];
end
```

This technique is used in obtaining the results presented in Figure 2 using solid lines (red). The relatively small difference between the code presented at the beginning of this section, and the code presented here, makes all the difference for accurate simulation of optimal control trajectories. The technique is also inline with general methods for handling discontinuities (Ascher and Petzold, 1998; Gustafsson).

5 Example application

In order to demonstrate the close connection between simulation and optimal control, an example application is demonstrated. The example is a variation on the classical OCP, Goddard's Rocket Problem, adapted from (Maurer, 1976; Rutquist and Edvall, 2010). The problem consists of launching a rocket as high up in the air as possible, given a finite amount of fuel.

The aim here is *not* to conduct rocket science so certain aspects are simplified. Instead, the aim is to demonstrate a case where simulation and numerical optimal control interact.

5.1 Rocket Model

The model has three states x: height h, speed v, and fuel mass m_f. The control input, u, is the fuel mass flow rate.

$$x = [h, v, m_f]^{\mathrm{T}},\ u = -\frac{dm_f}{dt}$$

The motion is governed by the ordinary differential equation

$$\dot{x} = f(x,u) = \begin{bmatrix} v \\ \frac{F_p(u) - F_D(v,h) - m(m_f)g(h)}{m(m_f)} \\ -u \end{bmatrix}$$

where F_p is the propulsion force, F_D the drag force, and g the gravitational acceleration. The rocket mass, $m = m_f + m_0$, consist of the fuel mass m_f, and ballast $m_0 = 68$ kg.

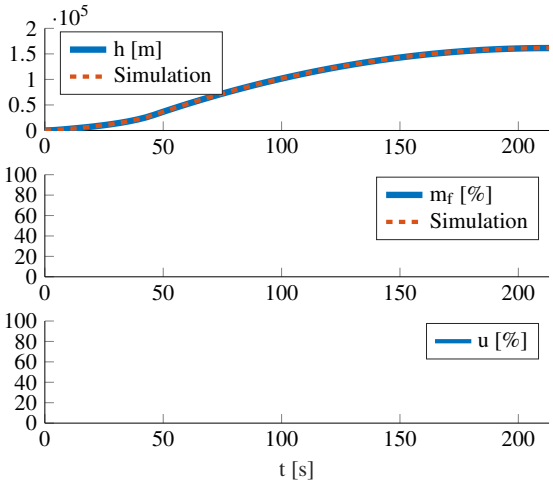

Figure 5. Numerical solution to the nominal formulation of the Goddard Rocket Problem (9), drawn in solid lines (blue). Fuel mass m_f and control (fuel mass flow) u, are presented in percentage of their maximum values. Confirmation by simulation drawn in dashed lines (red).

Propulsive force F_p is proportional to control effort

$$F_p = cu$$

and $c = 2069$ is the proportionality constant. Drag force is nonlinear and dependent on both height and speed

$$F_D = D_0 e^{-\gamma h} v^2$$

where $D_0 = 1.227 \cdot 10^{-2}$ and $\gamma = 1.450 \cdot 10^{-4}$ are model parameters. The gravitational acceleration accounts for how far above the Earth's surface the rocket is

$$g = g_0 \left(\frac{r_0}{r_0 + h} \right)^2$$

and $g_0 = 9.81$ m/s^2, and $r_0 = 6.371 \cdot 10^6$ m is Earth radius.

5.2 Nominal problem formulation

A nominal formulation of the problem is formulated as

$$\begin{aligned} \max_{t_f, u} \quad & h(t_f) \\ \text{s.t.} \quad & \dot{x} = f(x,u),\ t_f \geq 0, \\ & x(0) = [0,0,150]^{\mathrm{T}}, \\ & h \geq 0,\ v \geq 0,\ m_f \geq 0, \\ & 0 \leq u \leq 9.5 \end{aligned} \tag{9}$$

in which the final time, t_f, is a problem parameter. A solution to the formulation is found in Figure 5.

5.3 Problem variation

Assume γ is a parameter that is best described as a random variable on a launch-to-launch basis, but constant over a single launch

$$\gamma = \gamma_{nom} \mathcal{N}(\mu, \sigma^2) \tag{10}$$

with $\gamma_{nom} = 1.450 \cdot 10^{-4}$. Assume that the same reference trajectory is going to be used for every launch and that it therefore is desirable to find a balance between final height, $h(t_f)$, and deviation from the nominal trajectory when γ changes. To measure deviation the root mean square error (RMSE) is used

$$\text{RMSE} = \sqrt{\frac{1}{t_f} \int_0^{t_f} (h^* - h^s)^2 dt} \tag{11}$$

in which the optimal height trajectory is denoted by h^* and the simulated one by h^s for which γ changes.

To balance between height and deviation it is studied how the state trajectory change for a sufficiently small change, ϕ, in a parameter, p and is written as

$$x(t, p + \phi) = x(t,p) + \phi \frac{dx(t,p)}{dp} + \mathcal{O}(\phi^2)$$

Introducing the notation $P = \frac{dx(t;p)}{dp}$, the perturbation matrix function P is governed by the sensitivity equation

$$\dot{P} = \left(\frac{\partial f}{\partial x} \right) P + \frac{\partial f}{\partial p} \tag{12}$$

with the initial condition $P(0) = 0$. See (Ascher and Petzold, 1998) for a derivation. For $p = \gamma$ a cost for the sensitivity $dh/d\gamma$ is included in the objective function as a mean to balance the two objectives:

$$\max_{t_f, x, u} \quad h(t_f) - \beta \int_0^{t_f} \left(W \frac{dh}{d\gamma} \right)^2 dt$$

β is the trade-off parameter, and $W = 10^{-8}$ is a normalization factor that is used to avoid unreasonably small values of β. The formulation penalizes both height and final time t_f. To remedy this, an extra constraint is introduced, $v(t_f) = 0$, which ensures the trajectory reaches the apex.

The full formulation of the problem variation is:

$$\begin{aligned} \max_{t_f, x, u} \quad & h(t_f) - \beta \int_0^{t_f} \left(W \frac{dh}{d\gamma} \right)^2 dt \\ \text{s.t.} \quad & \dot{x} = f(x,u),\ t_f \geq 0, \\ & x(0) = [0,0,150]^{\mathrm{T}}, \\ & h \geq 0,\ v \geq 0,\ m_f \geq 0, \\ & 0 \leq u \leq 9.5, \\ & v(t_f) = 0 \end{aligned} \tag{13}$$

Note that $f(x,u)$ is used ambiguously and in this formulation includes the sensitivity equation (12), with $p = \gamma$.

For $\gamma = \gamma_{nom}$, Figure 6 presents the solution to the problem for a three different values of β, and Figure 7 shows the corresponding sensitivity state trajectory $dh/d\gamma$.

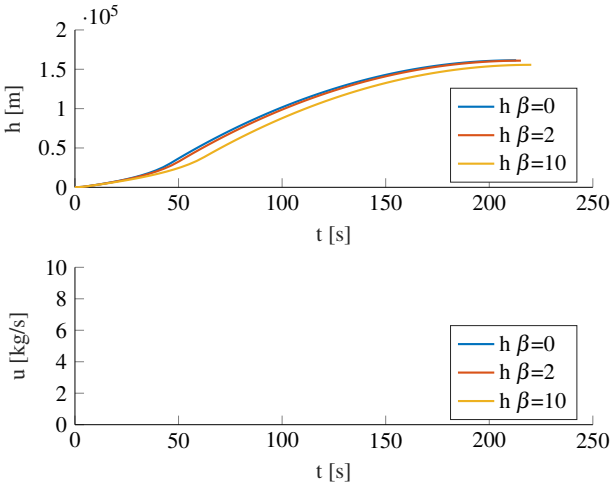

Figure 6. Solution to (13) for three different values of β.

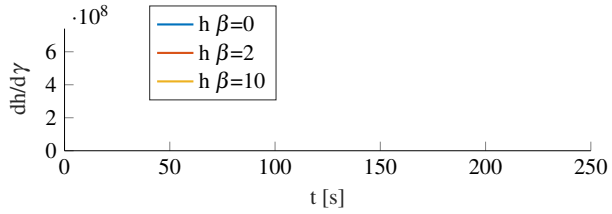

Figure 7. Optimal sensitivity trajectory $\frac{dh}{d\gamma}$ for different values of β.

5.4 Simulation

To quantify the trade-off between the two performance variables, final height $h(t_f)$ and RMSE, the problem (13) is solved for sequence of values of β, for $K = 12$

$$\beta_1 < \beta_2 < \cdots < \beta_K \tag{14}$$

For every solution corresponding to β_k, the optimal control, u_k^*, is simulated, using simulation setup (8), with the parameter γ drawn from the distribution (10). 10'000 simulations are run for every β_k. By completing the procedure for the full sequence of β (14), the results can be plotted, and a Pareto front is formed, see Figure 8. It clearly shows the trade-off between the performance variables.

For $\gamma = \gamma_{nom}\mathcal{N}(1, 0.15)$ Figure 8 shows the trade-off drawn in solid lines (blue), and for $\gamma = \gamma_{nom}\mathcal{N}(0.85, 0.15)$ in dashed lines (red). The height is normalized with maximum height for nominal parameter values, solution to problem (9), and deviation is normalized with the maximum one for the corresponding distribution of β. For the unbiased distribution it can be seen the final height is maximized for the second-most point from the right ($\beta = 1$), although only slightly higher than the nominal trajectory, but deviation is reduced by about 6 %, a free lunch. For the biased estimate, it can be seen that $\beta = 2$ maximizes height, but the most interesting point for both cases is perhaps $\beta = 5$ which gives a significant reduction in deviation while maintaining much of the height. An important aspect of the example is the use of simulation as a mean to gain insights into the optimal control, which shows that simulation too is an integral part of optimal control, not just optimization.

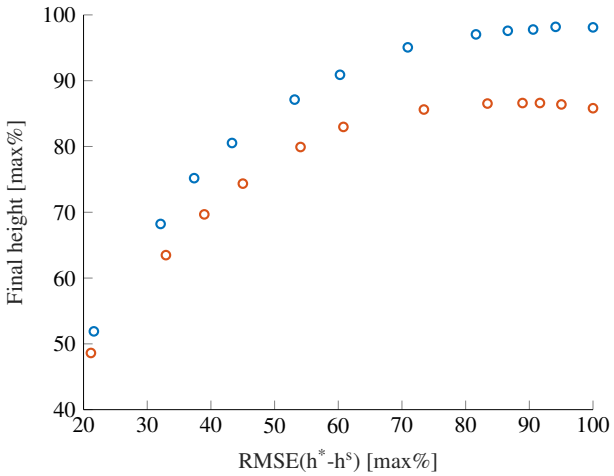

Figure 8. Pareto-optimal solution to (13), for $\beta = \{0, 1, 2, 3, 5, 10, 20, 30, 50, 70, 100, 200\}$. Values to the right correspond to lower values of β. Solid line (blue) correspond to the distribution $\gamma = \gamma_{nom}\mathcal{N}(1, 0.15)$, dashed lines (red), to the distribution $\gamma = \gamma_{nom}\mathcal{N}(0.85, 0.15)$.

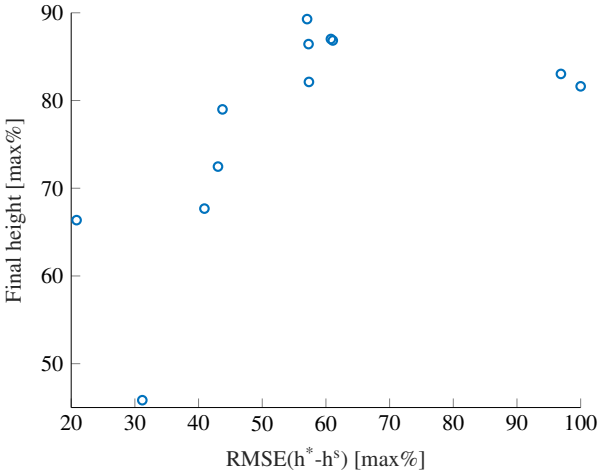

Figure 9. Erroneous Pareto-optimal solution to (13), for $\beta = \{0, 1, 2, 3, 5, 10, 20, 30, 50, 70, 100, 200\}$

As the example is primarily devoted to the simulation aspect, the obtained results are compared to the same analysis but using the look-up based simulation method (7). The simulations are otherwise conducted in the same way, solver and solver settings remains the same. The problem is only solved for the distribution $\gamma = \gamma_{nom}\mathcal{N}(1, 0.15)$ and Figure 9 shows the results. The results are erroneous, and any analysis of the results are therefore useless. In this case, it is obvious that something is wrong, but in a real case it does not have to be as easy to decide, and for those cases it is important to have confidence in the method.

6 Conclusions

It is demonstrated how simulation in NOC can fail and how it can be detected using accurate simulation. A user's model of the transcription process is developed, and based on it, a technique for structuring accurate simulations. The technique is based on the fact that the parameterized control input is discontinuous and to accurately handle that, the simulation is restarted at the discontinuity. MATLAB code, which is simple enough to act as pseudo code for other languages, is provided and is a practical guide for the user on how to apply the technique.The effectiveness and the importance of accurate simulation is demonstrated using an example. It shows how simulation can be used as a tool for getting the most out of optimal control, but also how an inappropriate simulation setup can lead to erroneous results.

References

Christian Andersson, Claus Führer, and Johan Åkesson. Assimulo: A unified framework for ode solvers. *Mathematics and Computers in Simulation*, 116:26–43, 2015.

Joel Andersson. *A general-purpose software framework for dynamic optimization*. PhD thesis, PhD thesis, Arenberg Doctoral School, KU Leuven, Department of Electrical . . . , 2013.

Joel AE Andersson, Joris Gillis, Greg Horn, James B Rawlings, and Moritz Diehl. Casadi: a software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11(1):1–36, 2019.

Uri M Ascher and Linda R Petzold. *Computer methods for ordinary differential equations and differential-algebraic equations*, volume 61. Siam, 1998.

John T Betts. *Practical methods for optimal control and estimation using nonlinear programming*. SIAM, 2010.

Lorenz T Biegler. *Nonlinear programming: concepts, algorithms, and applications to chemical processes*. SIAM, 2010.

Hans Georg Bock and Karl-Josef Plitt. A multiple shooting algorithm for direct solution of optimal control problems. *IFAC Proceedings Volumes*, 17(2):1603–1608, 1984.

John Charles Butcher. A history of runge-kutta methods. *Applied numerical mathematics*, 20(3):247–260, 1996.

M. Diehl, H. G. Bock, H. Diedam, and P. B. Wieber. Fast direct multiple shooting algorithms for optimal robot control. *Lecture Notes in Control and Information Sciences*, 340:65–93, 2006. ISSN 01708643. doi:10.1007/978-3-540-36119-0_4.

Moritz Diehl. Numerical optimal control. Technical report, KU Leuven, 2011.

John R Dormand and Peter J Prince. A family of embedded runge-kutta formulae. *Journal of computational and applied mathematics*, 6(1):19–26, 1980.

Charles William Gear and O Osterby. Solving ordinary differential equations with discontinuities. *ACM Transactions on Mathematical Software (TOMS)*, 10(1):23–44, 1984.

Kjell Gustafsson. Traps and pitfalls in simulation. Technical report, Ericsson Mobile Communications AB.

Charles R Hargraves and Stephen W Paris. Direct trajectory optimization using nonlinear programming and collocation. *Journal of guidance, control, and dynamics*, 10(4):338–342, 1987.

Håkan Lundvall, Peter Fritzson, and Bernhard Bachmann. *Event handling in the openmodelica compiler and runtime system*. Linköping University Electronic Press, 2005.

C Majer, W Marquardt, and Ernst Dieter Gilles. Reinitialization of dae's after discontinuities. *Computers & chemical engineering*, 19:507–512, 1995.

Guiyou Mao and Linda R Petzold. Efficient integration over discontinuities for differential-algebraic systems. *Computers & Mathematics with Applications*, 43(1-2):65–79, 2002.

H Maurer. Numerical solution of singular control problems using multiple shooting techniques. *Journal of Optimization Theory and Applications*, 18(2):235–257, 1976.

Anil V Rao. A survey of numerical methods for optimal control. *Advances in the Astronautical Sciences*, 135(1):497–528, 2009.

James Blake Rawlings, David Q Mayne, and Moritz Diehl. *Model predictive control: theory, computation, and design*, volume 2. Nob Hill Publishing Madison, WI, 2017.

Per E Rutquist and Marcus M Edvall. Propt-matlab optimal control software. *Tomlab Optimization Inc*, 260(1):12, 2010.

Lawrence F Shampine and Mark W Reichelt. The matlab ode suite. *SIAM journal on scientific computing*, 18(1):1–22, 1997.

LF Shampine, HA Watts, and SM Davenport. Solving nonstiff ordinary differential equations—the state of the art. *Siam Review*, 18(3):376–411, 1976.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi:10.1038/s41592-019-0686-2.

O. von Stryk and R. Bulirsch. Direct and indirect methods for trajectory optimization. *Annals of Operations Research*, 37(1):357–373, 1992. ISSN 02545330. doi:10.1007/BF02071065.