Towards a Framework for Nonlinear Predictive Control using Derivative-Free Optimization ⋆

Ian McInerney * **Lucian Nita** * **Yuanbo Nie** **
Alberto Oliveri *** **Eric C. Kerrigan** *,**

* *Department of Electrical & Electronic Engineering, Imperial College London, SW7 2AZ London, UK, email: {i.mcinerney17,lucian.nita16,e.kerrigan,}@imperial.ac.uk*
** *Department of Aeronautics, Imperial College London, SW7 2AZ London, UK*
*** *Department of Electrical, Electronic, Telecommunications Engineering and Naval Architecture, University of Genoa, via Opera Pia 11a, 16145, Genova, IT, email: alberto.oliveri@unige.it*

Abstract: The use of derivative-based solvers to compute solutions to optimal control problems with non-differentiable cost or dynamics often requires reformulations or relaxations that complicate the implementation or increase computational complexity. We present an initial framework for using the derivative-free Mesh Adaptive Direct Search (MADS) algorithm to solve Nonlinear Model Predictive Control problems with non-differentiable features without the need for reformulation. The MADS algorithm performs a structured search of the input space by simulating selected system trajectories and computing the subsequent cost value. We propose handling the path constraints and the Lagrange cost term by augmenting the system dynamics with additional states to compute the violation and cost value alongside the state trajectories, eliminating the need for reconstructing the state trajectories in a separate phase. We demonstrate the practicality of this framework by solving a robust rocket control problem, where the objective is to reach a target altitude as close as possible, given a system with uncertain parameters. This example uses a non-differentiable cost function and simulates two different system trajectories simultaneously, with each system having its own free final time.

Keywords: optimal control, mesh adaptive direct search, derivative-free optimization

1. INTRODUCTION

Model Predictive Control (MPC) has grown in popularity, due to its explicit handling of system constraints and the availability of efficient optimization solvers to compute the control solution. With the rise of data-based control and economic MPC, more engineers are wanting to control systems described by higher-fidelity models that capture nonlinearities or systems that only have blackbox/data-based models. These systems may pose a challenge for solvers that use first or second-order methods, since these methods require derivative information for the system dynamics, which may not be available or easily attainable.

MPC can also be used with derivative-free optimization methods, where no knowledge of the derivatives for the optimization problem is required. To solve the Nonlinear MPC (NMPC) problem, these methods perform simulations of the dynamics to locate the future input trajectory that minimizes the cost function. These solvers can lead to embarrassingly parallel implementations, since all simulations in an iteration can be run in parallel. Simulation-based solvers have been readily used in Finite Control Set (FCS) algorithms in power electronics (Kouro et al., 2015). The derivative-free methods used to solve the FCS MPC problem are inefficient for long time horizons though, since they usually perform an exhaustive search of the possible future input sequences, which leads to a combinatorial explosion in the search space size as the horizon grows. Prior work has suggested several ways to work around this combinatorial explosion by randomly sampling a fixed number of points in the search space (Joos et al., 2012), or using methods such as pattern search (Gibson, 2015), the Nelder-Mead simplex algorithm (Sadrieh and Bahri, 2011), Trust Region methods (Dæhlen et al., 2014), or Particle Swarm Optimization (Xu et al., 2016).

In this work, we build on the pattern search method from Gibson (2015) and instead propose using the Mesh Adaptive Direct Search (MADS) algorithm from Audet and Dennis (2006). Using MADS, the number of points evaluated around the current iterate grows linearly with the input dimension, instead of combinatorially. We also propose an NMPC formulation for derivative-free optimization solvers that provides an easy way to handle the problem's path constraints and Lagrange cost term.

⋆ The support of the EPSRC Centre for Doctoral Training in High Performance Embedded and Distributed Systems (HiPEDS, Grant Reference EP/L016796/1) and a Royal Society International Exchanges Grant (IES/R3/170011) is gratefully acknowledged.

In Section 2 we introduce the NMPC optimization problem we use throughout the rest of the work, and in Section 3 we present an overview of the MADS algorithm. In Section 4, we show how to transform the NMPC problem into a form suitable for the MADS algorithm. We then present an example showing MADS solving an NMPC problem in Section 5, and conclude the paper with some future research directions in Section 6.

2. NONLINEAR MODEL PREDICTIVE CONTROL

NMPC can be formulated as the optimization problem

$$\min_{x,u,t_f} \Phi(x(t_f), u(t_f), t_f) + \int_{t_0}^{t_f} L(x(t), u(t), t)dt \quad (1a)$$

$$\text{s.t. } f(x(t), \dot{x}(t), u(t), t) = 0, \quad \forall t \in [t_0, t_f] \quad (1b)$$

$$g(x(t), u(t), t) \leq 0, \quad (1c)$$

$$h(x(0), u(0), t_0, x(t_f), u(t_f), t_f) = 0, \quad (1d)$$

where $x : [t_0, t_f] \to \mathbb{R}^{n_x}$ and $u : [t_0, t_f] \to \mathbb{R}^{n_u}$ are the continuous-time state and input trajectories, respectively. f is the continuous-time nonlinear system dynamics, g is the path constraints, and h is the boundary conditions. The cost functional (1a) is composed of two terms: the Mayer cost Φ, and the Lagrange cost L.

3. MESH ADAPTIVE DIRECT SEARCH

This section provides a tutorial on the Mesh Adaptive Direct Search (MADS) algorithm. We combine the ideas from several works into a single statement of the algorithm, and provide a thorough discussion on the two techniques for implementing constraints in MADS. The notation used in this section has been slightly modified from the original MADS papers to make it consistent and to allow for a clearer description of the NMPC framework in Section 4.

MADS is an extension of the Generalized Pattern Search (GPS) derivative-free method, and was first proposed in Audet and Dennis (2006). MADS has been extended to work with other types of variables (such as periodic, granular, integer or binary), model-based techniques, and multi-objective optimization (Audet and Hare, 2017). The optimization problem that we solve using MADS is

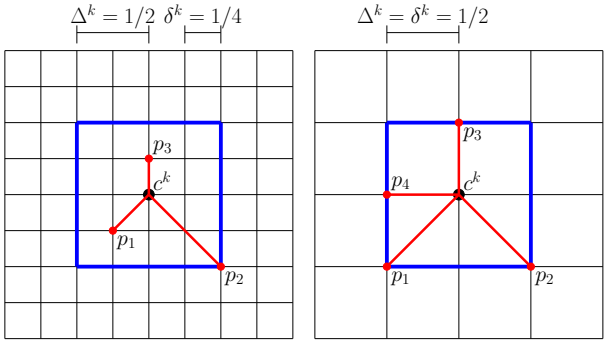
$$\min_{c,w} \mathcal{F}(c)$$

$$\text{s.t. } (c, w) \in \Omega := \{c \in \mathbb{R}^m, w \in \mathbb{R}^j : \omega_i(c, w) \leq 0, \forall i \in K\},$$

where $\mathcal{F} : \mathbb{R}^m \to \mathbb{R}$ is an arbitrary function that computes the cost of the optimization problem. The vector $c \in \mathbb{R}^m$ contains the optimization variables that form the *search space* and $w \in \mathbb{R}^j$ is a vector of internal variables computed by $\mathcal{F}$ and used only in the constraints. The constraint set Ω is defined by the functions $\omega_i(\cdot)$ spanning the search space and the internal variables of the cost function $\mathcal{F}$, with K the set of indices for the functions $\omega_i(\cdot)$ that define Ω.

3.1 MADS Algorithm

Pattern search methods sample the search space at a set of points called *poll points* in each iteration. The poll points are located on a *mesh* of size δ^k inside a *frame* of size Δ^k around the current iterate c^k, as in Fig. 1. The poll point with the lowest cost value is chosen as the next iterate.



(a) MADS after 1 iteration (b) GPS after 1 iteration

Fig. 1. The mesh (solid black lines spaced δ^k apart) and frame (region of size Δ^k inside the solid blue lines) for GPS and MADS after 1 successful iteration.

The overall MADS algorithm is given in Algorithm 1, and consists of three main parts: a search phase, a poll phase, and a mesh/frame adaptation phase. In the search phase, a set of points located on the mesh is generated (with no restriction on the method used to generate the points). The cost function is evaluated at these points, and if any have a lower cost value than the current iterate, the mesh and frame size are enlarged, and the search phase is repeated. If no better point is found, the poll phase is started.

In the poll phase, a set of polling directions that form a positive spanning set of the search space are generated to create a set of $m+1$ or more poll points contained inside the frame. The cost function is evaluated at each poll point. MADS can perform these evaluations in parallel and in an opportunistic fashion, meaning it can evaluate multiple poll points at once and end the current iteration immediately after finding a point with lower cost value.

The final step in the MADS algorithm is to adapt the mesh and frame based on the results of the polling. If a lower cost has been found, then the mesh and frame get enlarged by the adjustment parameter τ to allow for a faster exploration of the search space. If no lower cost value is found, then the mesh and frame sizes are shrunk by τ to concentrate the search closer to the current iterate. MADS can be terminated once the mesh size reaches a pre-determined threshold, meaning that there is no better point within that distance of the current iterate.

The poll and mesh adaptation phases are the only two phases required in every iteration of the MADS algorithm to get convergence. The search phase is optional; however, its use can speed up algorithm convergence, since it allows for more varied exploration of the search space. This can help the algorithm explore faster than the mesh can enlarge, and help it escape local minimums.

3.2 Constraint Handling

Implementing constraints inside MADS can be handled in at least one of two ways: (1) directly constrain the poll points and either reject any that violate the constraints or modify the poll points, (2) compute a metric indicating the amount of constraint violation and use it in the algorithm. If a constraint only contains variables in the algorithm's search space (e.g. only containing c and not w), option 1

allows for any poll points that violate the constraint to be skipped and not evaluated. Doing this can help to speed up the optimization process when $\mathcal{F}$ is expensive to evaluate, but does not provide any information to guide MADS from infeasible points to the feasible space.

To handle constraints that include both the poll point c and internal variables w, the cost function can be redefined to be an extremal barrier function

$$\mathcal{F}_\Omega(c) = \begin{cases} \mathcal{F}(c), & \forall c \in \Omega, \\ \infty, & \forall c \notin \Omega. \end{cases} \tag{2}$$

The barrier term (2) ensures that MADS optimizes over only the feasible points by forcing all infeasible points to have infinite cost. While easy to implement, this has several drawbacks, including (i) no information is provided to guide MADS from infeasible points to the feasible space, (ii) a feasible starting point is required.

Alternately, Audet and Dennis Jr (2009) proposed a progressive barrier technique that uses the constraint violation to keep both a feasible and infeasible iterate. In the poll phase, the n_p polling directions are used to generate a set of n_p poll points around both iterates ($2n_p$ total poll points). By tracking the best infeasible iterate, MADS can overcome the drawbacks of the extremal barrier method.

We define the set $\Psi \supset \Omega$ as the relaxed constraint set, where some constraints are implemented using a progressive barrier form and others use an extremal barrier form. The function $\mathcal{H} : \mathbb{R}^{k+j} \to \mathbb{R}$ is then used to compute the constraint violation, and is defined as

$$\mathcal{H}(c,w) = \begin{cases} 0, & \text{if } (c,w) \in \Omega, \\ \sum_{l \in K_\Psi} \left(\max\{\omega_l(c,w), 0\}\right)^2, & \text{if } (c,w) \in \Psi \setminus \Omega, \\ \infty, & \text{otherwise,} \end{cases}$$

where K_Ψ is the set of indices for the constraints $\omega(\cdot)$ that use the progressive barrier form.

Inside MADS, the progressive barrier relies on the notion of a dominating iterate, which is a point that is better than the others in a specific sense.

Definition 1a. (Dominating Feasible Point). The feasible point $p \in \Omega$ *dominates* the point $y \in \Omega$, denoted $p \prec_{\mathcal{F}} y$, when $\mathcal{F}(p) < \mathcal{F}(y)$.

Definition 1b. (Dominating Infeasible Point). The infeasible point $p \in \Psi \setminus \Omega$ *dominates* the point $y \in \Psi \setminus \Omega$, denoted $p \prec_{\mathcal{H}} y$, when $\mathcal{F}(p) < \mathcal{F}(y)$ and $\mathcal{H}(p, w_p) \leq \mathcal{H}(y, w_y)$ (with at least one strict inequality).

For feasible points, dominating means that the point has the smallest cost value, while for infeasible points it means that the point has both the smallest cost value and the smallest constraint violation.

An iteration of the MADS algorithm is one of three types: *dominating*, *improving* or *unsuccessful*. A dominating iteration is when MADS finds a new point that has both a lower cost value and also a lower (possibly 0) constraint violation, so the iterates are updated, and the mesh is expanded to search in a larger region around the new iterates. In an improving iteration, no points with a lower cost were found, but an infeasible point with a constraint violation smaller than the penalty parameter η was found. In this case the new point becomes the next infeasible iterate, the mesh remains unchanged, and η is set to the constraint violation at the new infeasible iterate. All other iterations are considered unsuccessful, so the mesh will be shrunken to search in a closer region to the iterates in the next iteration. Formal definitions for these iteration types can be found in Audet and Dennis Jr (2009).

The penalty parameter η is nonincreasing with iteration number, starting at ∞ and decreasing to 0 as the algorithm runs. This means that at the beginning, MADS is prioritizing the search for the lowest cost value by allowing large constraint violations, but over time η decreases and forces the infeasible iterates to move towards the feasible space. The penalty barrier constraint method behaves similar to a filter method in other optimization algorithms, such as SQP, but requires the barrier/penalty parameter to be only nonincreasing and not strictly decreasing.

3.3 Meshing

In MADS, the frame and mesh are controlled by different parameters (Δ^k and δ^k, respectively), with

$$\delta^k = \min\{\Delta^k, (\Delta^k)^2\} \tag{3}$$

normally used. The main difference between MADS and GPS is that in GPS $\delta_k = \Delta_k$ at all times. By allowing the mesh to shrink faster than the frame size, more polling points are created around the iterate c^k. For example, Fig. 1a shows the result of one iteration of MADS using update rule (3). In this case there are 24 possible polling points for MADS, versus 8 for GPS in Fig. 1b.

3.4 Convergence

The fact that the mesh in MADS shrinks faster than the frame size means that, as the algorithm converges, there will be an asymptotically dense set of poll directions created. It was shown in Audet and Dennis (2006) that this set of asymptotically dense poll directions allows for the algorithm to converge to a Clarke stationary point with non-negative Clarke derivatives (e.g. a local minimum for the non-smooth function) when linear constraints are placed on the search space. In contrast, GPS loses the theoretical convergence guarantees when simple bound constraints are applied. Additionally, when a progressive barrier approach is used for handling nonlinear constraints, it was shown in Audet and Dennis Jr (2009) that MADS is still effective at finding the stationary point.

4. NMPC PROBLEM FORMULATION FOR DERIVATIVE-FREE OPTIMIZATION

The core of the problem formulation we propose is a continuous-time shooting method based on Gibson (2015), where the search space for MADS is the set of all input trajectories. Each function evaluation in the MADS algorithm simulates the system over the time horizon with the chosen input trajectory, and then computes the overall cost value and constraint violation for that trajectory. To easily handle the path constraints and the Lagrange term, we introduce an augmentation scheme where the system dynamics are augmented with new states representing the Lagrange term and the path constraint violation.

Algorithm 1 Mesh Adaptive Direct Search with Progressive Barrier Constraints (Audet and Hare, 2017)

Let: $\mathbb{M}^k$ be the set of all mesh points using mesh size δ^k
Let: $\prec_{\mathcal{F}}$ and $\prec_{\mathcal{H}}$ be as given in Definitions 1a and 1b
Require: $\Delta^0 \in (0, \infty)$ ▷ Initial frame size
Require: $\tau \in (0, 1)$ ▷ Mesh size adjustment parameter
Require: $\epsilon_{stop} \in [0, \infty)$ ▷ Stopping tolerance
Require: c_f^0 **And\Or** c_i^0 ▷ Initial mesh centers

$k \leftarrow 0$
while $\Delta^k \geq \epsilon_{stop}$ **do** [1]
  $\mathbb{V}^k \leftarrow \emptyset$
  $\delta^k \leftarrow \min\{\Delta^k, (\Delta^k)^2\}$ ▷ Compute mesh size
  1) Search Phase:
  Generate search points $\mathbb{S}^k \subset \mathbb{M}^k$
  for all $s \in \mathbb{S}^k$ **do**
    Compute $\mathcal{F}$ and $\mathcal{H}$ at s
    if $s \prec_{\mathcal{F}} c_f^k$ **then**
      $c_f^{k+1} \leftarrow s$ ▷ Dominating
      goto line 35
    else if $s \prec_{\mathcal{H}} c_i^k$ **then**
      $c_i^{k+1} \leftarrow s$, $\eta^{k+1} \leftarrow \mathcal{H}(s)$ ▷ Dominating
      goto line 35
    end if
  end for
  2) Poll Phase:
  Generate positive spanning set $\mathbb{D}^k$
  $\mathbb{P}^k \leftarrow \{c_f^k + \delta^k d \quad \forall d \in \mathbb{D}^k\} \cup \{c_i^k + \delta^k d \quad \forall d \in \mathbb{D}^k\}$
  for all $p \in \mathbb{P}^k$ **do**
    Compute $\mathcal{F}$ and $\mathcal{H}$ at p
    if $p \prec_{\mathcal{F}} c_f^k$ **then**
      $c_f^{k+1} \leftarrow p$ ▷ Dominating
      goto line 35
    else if $p \prec_{\mathcal{H}} c_i^k$ **then**
      $c_i^{k+1} \leftarrow p$, $\eta^{k+1} \leftarrow \mathcal{H}(p)$ ▷ Dominating
      goto line 35
    else if $\mathcal{H}(p) < \eta^k$ **then**
      $\mathbb{V}^k \leftarrow \mathbb{V}^k \cup \{p\}$ ▷ Improving
    end if
  end for
  if Improving **then**
    $v \leftarrow \text{argmax}\{\mathcal{H}(v) : \mathcal{H}(v) < \eta^k, v \in \mathbb{V}^k\}$
    $c_i^{k+1} \leftarrow v$, $\eta^{k+1} \leftarrow \mathcal{H}(v)$
  end if
  3) Mesh/Frame Update:
  if Dominating **then**
    $\Delta^{k+1} \leftarrow \tau^{-1}\Delta^k$ ▷ Expand frame size
  else if Improving **then**
    $\Delta^{k+1} \leftarrow \Delta^k$ ▷ Frame doesn't change
  else
    $\Delta^{k+1} \leftarrow \tau\Delta^k$ ▷ Shrink frame size
  end if
  $k \leftarrow k + 1$
end while

[1] If a variable is not set in an iteration, it retains the same value in the next iteration.

4.1 Cost Functional

We split the NMPC cost functional (1a) into its two terms, and augment the system dynamics with the Lagrange term by adding a new state $l(t)$ that represents the value of the Lagrange term at time t. This new state is governed by

$$\dot{l}(t) = L(x(t), u(t), t), \tag{4}$$

which is computed alongside the system's trajectory.

4.2 Path Constraints

The constraints (1c) are formulated as progressive barrier constraints inside the set Ψ. An L_1 measure of constraint violation is used for each constraint, meaning the value reported as the violation experienced by constraint i is

$$v_i = \int_0^{t_f} \max\{0, g_i(x(t), u(t), t)\} dt. \tag{5}$$

To compute the integral (5), we add new states $v(t)$ that represent the constraint violation over time, with those states being governed by the dynamics equation

$$\dot{v}(t) = g^+(x(t), u(t), t) \tag{6}$$

where g^+ is the vector function representing the element-wise computation of $\max\{0, g_i(x(t), u(t), t)\}$.

4.3 Overall Problem Formulation

An implementation of the formulation described in this section is given in Algorithm 2. The first step is to construct the input trajectory described by the current poll point c. The structure of the input trajectory will depend on the problem, but possible options are: (i) a zero-order hold trajectory with the input values at each sample given by the elements of the poll point, (ii) an interpolated polynomial with the interpolation points given by the elements of the poll point, (iii) a feedback policy with the parameters of the policy given by the elements of the poll point. After constructing the input trajectory, the augmented system (7) is simulated over the horizon length using a suitable numerical solver for differential equations.

After the simulation completes, the boundary condition violation is computed, and the total constraint violation is then computed using (9), where ρ are weights that can be applied to various path and boundary constraints to give them more influence in the algorithm. Finally, the value of the NMPC cost function (1a) for the selected poll point is computed using (10), which computes the Mayer term and then adds the final value of the Lagrange state $l(t_f)$.

4.4 Discussion of the Augmentation Scheme

The proposed augmentation scheme provides an easy way to include constraints in the derivative-free optimization problem, but it may not be the most efficient way. Moving the constraints and the Lagrange term into the dynamics allows them to be on the same mesh as the dynamics, and removes the need for additional quadrature schemes to compute the final cost and constraint violation after the system has been simulated, as is usually done in other solvers. While this removes the need for an algorithm to handle the mesh refinement, it introduces a requirement

Algorithm 2 Function evaluation for the NMPC problem

Let: c be the point in the search space being evaluated

1: Construct the input trajectory u from c

2: Simulate the augmented dynamics (7) using an appropriate solver for the differential equations

$$\begin{bmatrix}0\\0\\0\end{bmatrix} = \begin{bmatrix} f(x(t),\dot{x}(t),u(t),t) \\ L(x(t),u(t),t) - \dot{l}(t) \\ g^{+}(x(t),u(t),t) - \dot{v}(t) \end{bmatrix} \tag{7}$$

3: Compute the violation of the boundary conditions

$$v_b = \sum_i \rho_{b_i} |h_i(x(0),u(0),t_0,x(t_f),u(t_f),t_f)| \tag{8}$$

4: Compute the overall constraint violation

$$\mathcal{H} \leftarrow v_b^2 + \sum_i \rho_i (v_i(t_f))^2 \tag{9}$$

5: Compute the cost function value

$$\mathcal{F} \leftarrow \Phi(x(t_f),u(t_f),t_f) + l(t_f) \tag{10}$$

that the dynamics solver must have an error control mechanism to limit the numerical integration error.

While easy to use, the augmentation scheme can also introduce inefficiencies to the solver. By introducing the dynamics equations (4) and (6), we have made the solvability and the stiffness of (7) be dependent on the Lagrange term and the constraints as well. This means that when the cost or constraints vary over time or are defined by stiff equations, they can cause the dynamics solver to struggle and possibly require smaller step sizes.

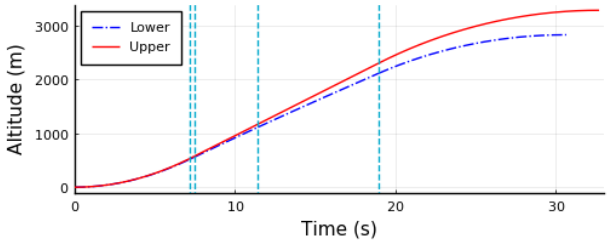
5. NUMERICAL EXAMPLES

We demonstrate the use of the formulation from Section 4 by solving a robust rocket throttle control problem. We examine a laboratory scale rocket aiming to reach its apogee at a target altitude of 10,000 feet, governed by

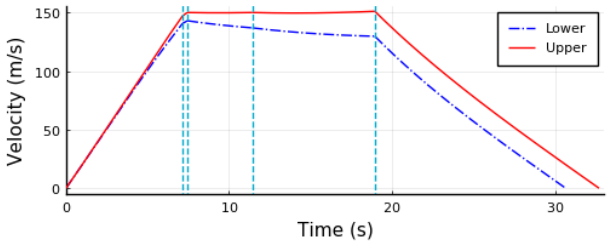
$$\dot{x}(t) = f(x(t)) = \begin{bmatrix} v_v(t) \\ \frac{T(t) - 0.5 C_D \rho(h(t)) \frac{\pi d^2}{4} (v_v(t))^2)}{m(t)} - g(h(t)) \\ -\frac{T(t)}{I_{sp}} \end{bmatrix},$$

with the state vector $x = [h\ v_v\ m]'$ composed of the rocket's altitude, vertical velocity and mass, respectively. $g(h)$ and $\rho(h)$ are the gravitational constant and the air density at altitude h, respectively, and d is the rocket diameter. The model has an uncertain drag coefficient $C_D \in [C_{D_l}, C_{D_u}]$ and specific impulse $I_{sp} \in [I_{sp_l}, I_{sp_u}]$. The thrust is a function of time, $T(\cdot)$, that takes values from the set $T(t) \in [0, T_{max}]$.

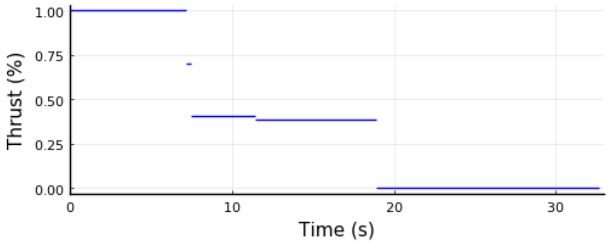
It can be shown that the open-loop system is monotone with respect to the throttle setting and physical parameters (C_D, I_{sp}) (Angeli and Sontag, 2003), so all possible trajectories will lie in a tube defined by an upper x_u and lower x_l trajectory (using the upper and lower drag coefficients and specific impulses, respectively). We use a piecewise constant throttle function, and optimize over both the throttle settings T_i and the switching times σ_i. A robust version of the problem then optimizes over the two trajectories x_l and x_u, with the objective to minimize the largest deviation of the two trajectories from a target apogee in the min-max formulation



(a) Altitude profile



(b) Velocity profile



(c) Input trajectory

Fig. 2. Open-loop solution to (11) with 5 phases (Switching times are at the dotted lines).

$$\min_{T,\sigma} \max_{x_l,x_u} \{|h_u(t_{f_u}) - 3048|, |3048 - h_l(t_{f_l})|\} \tag{11a}$$

$$\text{s.t. } \dot{x}_l(t) = f(x_l(t)),\ \dot{x}_u(t) = f(x_u(t)) \tag{11b}$$

$$x_l(0) = x_u(0) = [0\ 0\ 33.5]' \tag{11c}$$

$$v_{v_l}(t_{f_l}) = v_{v_u}(t_{f_u}) = 0 \tag{11d}$$

$$v_{v_l}(t) \le 150 \qquad \forall t \in [0, t_{f_l}] \tag{11e}$$

$$v_{v_u}(t) \le 150 \qquad \forall t \in [0, t_{f_u}] \tag{11f}$$

$$m_l(t_{f_l}) \ge 26,\ m_u(t_{f_u}) \ge 26 \tag{11g}$$

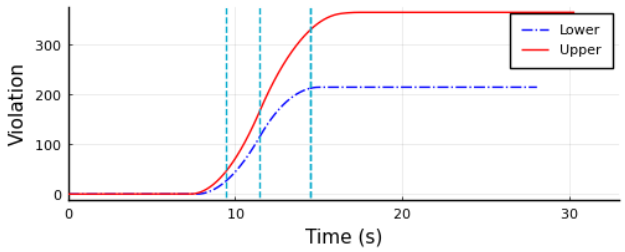
$$0 \le \sigma_1 \le \sigma_2 \le \cdots \le \sigma_i \tag{11h}$$

$$\sigma_i \le \min\{t_{f_l}, t_{f_u}\} \tag{11i}$$

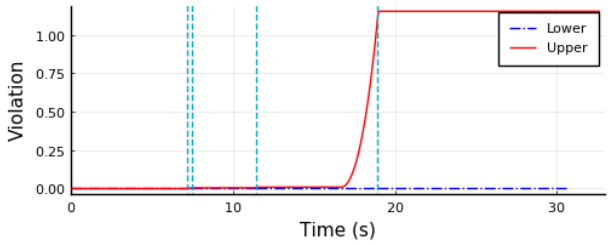
$$T_i \in [0, T_{max}] \qquad \forall i \tag{11j}$$

where the same thrust function is used for both the upper and lower trajectories. The trajectories share the same thrust profile T_i and switching times σ_i, but have their own final times given by t_{f_u} and t_{f_l} for the upper and lower trajectories respectively. As an example path constraint, we constrain the rocket's velocity to be less than 150 m/s, representing a modeling constraint where the drag coefficient uncertainity bounds are only known below 150 m/s.

MADS was used to solve (11) with a thrust trajectory composed of five intervals. This resulted in optimal throttle settings shown in Fig. 2c, which resemble the theoretically optimal 'bang-singular-bang' solution. The resulting altitude and velocity profiles can be seen in Figures 2a



(a) No constraints enforced



(b) Constraints enforced as progressive barrier constraints

Fig. 3. Cumulative violation of constraints (11e) and (11f)

and 2b, respectively. In the velocity profile, the constraint of 150 m/s is satisfied by both the upper and lower trajectories when the constraint is implemented using additional states, as described in Section 4.2.

The evolution of the violation state for constraints (11e) and (11f) can be seen in Fig. 3. The violations experienced when the velocity constraints are not enforced are shown in Fig. 3a, and are monotonically increasing along the horizon. Once the constraints are enforced, the constraint violation decreases significantly, as shown in Fig. 3b.

The non-zero violation for the upper trajectory is due to the current implementation of the progressive barrier constraints in the DirectSearch.jl [1] Julia package that was used. In the current implementation, there is no method for passing the internal variables w from the computation of $\mathcal{F}$ to the computation of $\mathcal{H}$. This means that (9) could not be used to compute the progressive barrier penalty value, so instead the velocity constraints were implemented by adding a penalty term in the cost for the deviation from zero of the final value of the velocity constraint states $v(t)$.

6. CONCLUSIONS AND FUTURE DIRECTIONS

We presented a formulation for NMPC that includes path constraints and the Lagrange cost term in a form suitable for use with MADS. We have also shown that this can be used to solve problems with non-differentiable cost functions, discontinuous dynamics and free end times.

This formulation provides the initial steps to implement NMPC with MADS, but there are more open questions remaining. In the robust rocket control example in Section 5, we showed a zero-order hold input sequence, but this sequence can become impractical to optimize over for systems with a long horizon or many inputs. Additional work should be done to examine how to effectively optimize over the other input representations given in Section 4.3, and to determine if there is a preferred representation to use.

[1] https://github.com/ImperialCollegeLondon/DirectSearch.jl

The reliance on a single shooting method for simulating the system trajectories in Algorithm 2 is also problematic. Small changes to the input sequence/initial conditions can drastically affect the simulated trajectory and the resulting constraint violation measure, so searching for an input trajectory that doesn't violate the boundary conditions or path constraints becomes more difficult. Future work should explore if a multiple shooting paradigm can be adapted into the proposed NMPC framework for MADS, with a particular focus on efficient handling of the defect constraints between the shooting regions. Additionally, switching to a multiple shooting approach may add opportunities for parallelism in the formulation itself by simulating all shooting intervals in parallel.

Finally, the example demonstrates this framework in use as an open-loop solver for optimal control problems, but the parallelization potential and simplicity of the computations may make the MADS-based framework beneficial for closed-loop control. More work should be done to examine how this framework behaves in a closed-loop implementation, and how a warm-started MADS implementation could be used inside a real-time iteration type framework.

REFERENCES

Angeli, D. and Sontag, E.D. (2003). Monotone control systems. *IEEE Transactions on Automatic Control*, 48(10), 1684–1698.

Audet, C. and Dennis, J.E. (2006). Mesh Adaptive Direct Search Algorithms for Constrained Optimization. *SIAM Journal on Optimization*, 17(1), 188–217.

Audet, C. and Dennis Jr, J.E. (2009). A progressive barrier for derivative-free nonlinear programming. *SIAM Journal on Optimization*, 20(1), 445–472.

Audet, C. and Hare, W. (2017). *Derivative-Free and Blackbox Optimization*. Springer, Cham, Switzerland.

Dæhlen, J.S., Eikrem, G.O., and Johansen, T.A. (2014). Nonlinear model predictive control using trust-region derivative-free optimization. *Journal of Process Control*, 24(7), 1106–1120.

Gibson, J.D. (2015). A direct search approach to optimization for nonlinear model predictive control. *Optimal Control Applications and Methods*, 36(2), 139–157.

Joos, A., Heritier, P., Huber, C., and Fichter, W. (2012). Method for Parallel FPGA Implementation of Nonlinear Model Predictive Control. In *1st IFAC Workshop on Embedded Guidance, Navigation and Control in Aerospace*, 73–78. IFAC, Bangalore, India.

Kouro, S., Perez, M.A., Rodriguez, J., Llor, A.M., and Young, H.A. (2015). Model Predictive Control: MPC's Role in the Evolution of Power Electronics. *IEEE Industrial Electronics Magazine*, 9(4), 8–21.

Sadrieh, A. and Bahri, P.A. (2011). Application of Graphic Processing Unit in Model Predictive Control. In *21st European Symposium on Computer-Aided Process Engineering (ESCAPE-21)*, volume 29, 492–496. Elsevier B.V., Porto Carras Resort, Chalkidiki, Greece.

Xu, F., Chen, H., Gong, X., and Mei, Q. (2016). Fast Nonlinear Model Predictive Control on FPGA Using Particle Swarm Optimization. *IEEE Transactions on Industrial Electronics*, 63(1), 310–321.