Available online at www.sciencedirect.com

ScienceDirect

IFAC Papers Online CONFERENCE PAPER ARCHIVE

IFAC PapersOnLine 59-18 (2025) 403–408

An Efficient Numerical Function Optimization Framework for Constrained Nonlinear Robotic Problems ⋆

Sait Sovukluk[1] **Christian Ott**[1,2]

[1] *Automation and Control Institute (ACIN), TU Wien, 1040 Vienna, Austria (e-mail: sovukluk@acin.tuwien.ac.at, christian.ott@tuwien.ac.at).*

[2] *Institute of Robotics and Mechatronics, German Aerospace Center (DLR), 82234 Weßling, Germany.*

Abstract: This paper presents a numerical function optimization framework designed for constrained optimization problems in robotics. The tool is designed with real-time considerations and is suitable for online trajectory and control input optimization problems. The proposed framework does not require any analytical representation of the problem and works with constrained block-box optimization functions. The method combines first-order gradient-based line search algorithms with constraint prioritization through nullspace projections onto constraint Jacobian space. The tool is implemented in C++ and provided online for community use, along with some numerical and robotic example implementations presented in the end.

Keywords: Nonlinear System Optimization, Constrained Robotic Problem Optimization.

1. INTRODUCTION

Numerical optimization (Nocedal and Wright (1999)) is one of the most fundamental backbones of robotics. Numerical optimization methods emerge to address problems that have no analytical solutions and are subject to constraints. Such problems constitute a vast variety of applications, some examples can be listed as mechanical design topology optimization for the best jumping performance (Haldane et al. (2017)), behavioral optimizations to perform certain tasks with the least energy consumption (Mei et al. (2004); Paes et al. (2014)), a control input optimization that also accounts for multiple tasks, limits, contact constraints, and closed kinematic chains (Sovukluk et al. (2023a); Khatib et al. (2022)), motion planning for complex robotic systems (Dai et al. (2014); Sovukluk et al. (2023b); Westervelt et al. (2018)), apex-to-apex periodic behavior optimization for nonlinear and underactuated systems (Sovukluk et al. (2024)), path planning optimizations for autonomous devices (Gasparetto et al. (2015)), and so on.

Convex optimization of linear(ized) robotic system problems are studied and explored extensively (Nocedal and Wright (1999); Lewis et al. (2012)). Such problems can scale up to thousands of parameters and still be solved in real-time (Stellato et al. (2020); Bambade et al. (2022); Pandala et al. (2019)). On the other hand, nonlinear system optimization problems are less straightforward to generalize and do not scale as such due to the numerical complexities and computational costs. Such problems are addressed by nonlinear programming (Bertsekas (1997)) and are also covered extensively. Some well-known community-available tools, such as CasADi (Andersson et al. (2019)) and IPOPT (Wächter and Biegler (2006)) are available for analytically representable nonlinear problems.

Robotic optimization problems, on the other hand, are usually not analytically representable due to the high dimensionality and nonlinearities. Let

$$\boldsymbol{M}(\boldsymbol{\nu})\dot{\boldsymbol{\nu}} + \boldsymbol{C}(\boldsymbol{q},\boldsymbol{\nu})\boldsymbol{\nu} + \boldsymbol{\tau}_g(\boldsymbol{q}) = \begin{bmatrix}\boldsymbol{0}\\ \boldsymbol{\tau}\end{bmatrix} + \boldsymbol{J}_c(\boldsymbol{q})^\top \boldsymbol{f}_c, \qquad (1)$$

be a floating base robot dynamics, where $\boldsymbol{q}$ is a set of configuration variables and $\boldsymbol{\nu} = (\boldsymbol{\nu}_b, \boldsymbol{\nu}_j)$ are the generalized velocity where $\boldsymbol{\nu}_b = (\boldsymbol{v}_b, \boldsymbol{\omega}_b) \in \mathbb{R}^6$ is the linear and angular velocity of the floating base and $\boldsymbol{\nu}_j \in \mathbb{R}^n$ is the generalized velocity of the joints. Due to their complexity and nonlinearity, such system dynamics are usually calculated through rigid body algorithms (Featherstone (2014)). Some community available tools are Pinocchio (Carpentier et al. (2015–2021, 2019)), MuJoCo (Todorov et al. (2012)), and RBDL (Felis (2016)). As a result, the analytical representation of gradients and Hessians of optimization problems that include system dynamics and kinematics such as

$$\begin{aligned} &\min_{\boldsymbol{x}} f(\boldsymbol{x}, \textit{System Dynamics}(\boldsymbol{x}), \textit{Kinematics}(\boldsymbol{x}))\\ &\text{such that}\\ &\boldsymbol{g}_{\text{eq}}(\boldsymbol{x}, \textit{System Dynamics}(\boldsymbol{x}), \textit{Kinematics}(\boldsymbol{x})) = \boldsymbol{0}\\ &\boldsymbol{g}_{\text{ineq}}(\boldsymbol{x}, \textit{System Dynamics}(\boldsymbol{x}), \textit{Kinematics}(\boldsymbol{x})) < \boldsymbol{0} \end{aligned} \qquad (2)$$

is not possible. As numerical Hessian estimation of such nonlinear and high dimensional problems is too expensive, a simple gradient-descent based numerical optimization method that can still take account of equality and inequality constraints may become preferable for performance and real-time implementation purposes.

⋆ This work was supported in part by the Robot Industry Core Technology Development Program under Grant 00416440 funded by the Korea Ministry of Trade, Industry and Energy (MOTIE).

10.1016/j.ifacol.2025.10.254

This paper proposes a numerical function optimization framework that is specialized for constrained nonlinear robotic problems. The method combines first-order gradient-based line search algorithms with constraint prioritization through nullspace projections of constraint Jacobian space. The framework does not require any analytic representation of the problem and works fully numerically. Hence, it can also be referred to as a black-box optimizer. As it is too costly to compute the Hessian matrix numerically, this method employs only gradient-based search algorithms. This selection results in slower convergence but reduces the iteration cost greatly. The framework also provides a set of special update routines to update the system dynamics and kinematics either every inner iteration or once per outer iteration for performance considerations. The authors also provide the precompiled C++ libraries (ENFORCpp) for community use along with multiple numerical and one three-link robotic arm optimization implementation examples:

https://github.com/ssovukluk/ENFORCpp $(\star)$

2. METHOD

2.1 Optimization Problem

Let

$$\begin{aligned} &\min_{\boldsymbol{x}} f(\boldsymbol{x}) \\ &\text{such that} \\ &\boldsymbol{g}_{\text{ec}}(\boldsymbol{x}) = \boldsymbol{0} \\ &\boldsymbol{g}_{\text{ic}}(\boldsymbol{x}) < \boldsymbol{0} \end{aligned} \tag{3}$$

be an optimization problem as described in (2), where $f(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}_{\geq 0}$, $\boldsymbol{g}_{\text{ec}}(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}^{n_{\text{ec}}}$, and $\boldsymbol{g}_{\text{ic}}(\boldsymbol{x}) : \mathbb{R}^n \to \mathbb{R}^{n_{\text{ic}}}$ are the cost, equality constraint, and inequality constraint functions, respectively. Furthermore, $n \in \mathbb{N}_{>0}$, $n_{ec} \in \mathbb{N}_0$, and $n_{ic} \in \mathbb{N}_0$ represent the number of optimization parameters, number of equality constraints, and number of inequality constraints, respectively.

2.2 Nullspace Projection

The optimization problem requires finding the parameter set that both minimizes the cost function $f(\boldsymbol{x})$ and satisfies the equality and inequality constraints. Such problems cannot be solved simply by iterating through a gradient estimation. First, the constraints should be satisfied, and then the cost function should be iterated in the allowed directions that do not disturb any constraints but still reduce the cost. Let g_k be the k^{th} active constraint function. Furthermore, let a Jacobian matrix $\boldsymbol{J}_{\text{ac}} \in \mathbb{R}^{n_{\text{ac}} \times n}$ collect gradient transpose of all active constraint functions,

$$\boldsymbol{J}_{\text{ac}} = \begin{bmatrix} \partial g_1 / \partial \boldsymbol{x} \\ \partial g_2 / \partial \boldsymbol{x} \\ \vdots \\ \partial g_{n_{\text{ac}}} / \partial \boldsymbol{x} \end{bmatrix}, \tag{4}$$

where n_{ac} represent the number of the active constraints, which may differ from the total constraint number $(n_{ec} + n_{ic})$ as not all inequality constraints may be active if they are far from the boundary. Also, let

$$\nabla f^* = \text{proj}_{N(\boldsymbol{J})} \nabla f = (\boldsymbol{I} - \boldsymbol{J}^\top (\boldsymbol{J}\boldsymbol{J}^\top)^{-1} \boldsymbol{J}) \nabla f \tag{5}$$

be a nullspace projection (orthogonal projection onto nullspace) of the constraint Jacobian space such that moving towards the $-\nabla f^*$ direction reduces the cost function without disturbing the active constraints of the problem. Such projections will be used for hierarchical optimization between the constraint and cost functions.

2.3 Optimization Strategy

The optimization strategy is built on a hierarchical framework. First, all constraint functions are optimized one by one, in the nullspace of previous ones such that they do not disturb the previous constraints. Then the cost function is optimized in the nullspace of all active constraints to look for a minimal solution that satisfies all active constraints.

An example optimization sequence: Assume an optimization problem with two constraints all of which are active, i.e., $n_{\text{ac}} = n_{\text{ec}} + n_{\text{ic}} = 2$. A solution sequence for such a problem can be summarized as:

(1) Estimate ∇g_1, gradient of the first constraint.
(2) Optimize for g_1 through ∇g_1.
(3) Estimate ∇g_2.
(4) Project ∇g_2 onto the nullspace of the constraint Jacobian, i.e., $\nabla g_2^* = \text{proj}_{N(\boldsymbol{J}_1)} \nabla g_2$, where $\boldsymbol{J}_1 = [\nabla g_1]^\top$.
(5) Optimize for g_2 through ∇g_2^*.
(6) Estimate ∇f, the cost function gradient.
(7) Project ∇f onto the nullspace of the constraint Jacobian, i.e., $\nabla f^* = \text{proj}_{N(\boldsymbol{J}_2)} \nabla f$, where $\boldsymbol{J}_2 = [\nabla g_1, \nabla g_2]^\top$.
(8) Optimize for the cost function through ∇f^*.

2.4 Equality Constraint Optimization Subroutine

Solving for the equality constraints is a similar procedure to solving for the cost function. As described in the previous subsection, each equality constraint is solved in the nullspace of the previous equality constraint Jacobian. The details of the equality constraint optimization algorithm are provided in Alg. 1.

2.5 Inequality Constraint Optimization Subroutine

The inequality constraints are less straightforward to handle than the equality constraints due to two reasons. First, they may not always be active. Second, they are directional. As a result, these constraints should be checked continuously and activated when necessary. Furthermore, if the iteration direction aligns with the inequality constraint's gradient direction, the constraint should not be activated as any iteration in the given optimal direction would not violate the constraint. The activation conditions are summarized through an example in Fig. 1. A direction check between two vectors can be easily done through the dot operator. Let $\alpha \in \mathbb{R}$ be an angle between two arbitrary vectors $\boldsymbol{v}_0$ and $\boldsymbol{v}_1$, then

$$\begin{cases} \boldsymbol{v}_0 \cdot \boldsymbol{v}_1 > 0, & 0 < \alpha < \pi/2 \\ \boldsymbol{v}_0 \cdot \boldsymbol{v}_1 < 0, & \pi/2 < \alpha < \pi \\ \boldsymbol{v}_0 \cdot \boldsymbol{v}_1 = 0, & \boldsymbol{v}_0 \perp \boldsymbol{v}_1. \end{cases}$$

The details of the inequality constraint optimization algorithm are provided in Alg. 2.

Algorithm 1 Optimizing for equality constraints.

Require: $\boldsymbol{x} \in \mathbb{R}^n$, $n_{\text{ce}} \geq 0$
$\boldsymbol{J}_{\text{eq}}$.empty()
$n_{\text{ac}} \leftarrow 0$
for ($k \leftarrow 1$; $k \leq n_{\text{ec}}$; ++k) **do**
  $n_{\text{ac}} = n_{\text{ac}} + 1$
  $\text{cost} = g_{\text{ec}}(\boldsymbol{x})$
  $\nabla g_{\text{ec},k} = \text{sign}(\text{cost}) \times \text{numericalGradient}(g_{\text{ec},k}(\boldsymbol{x}))$
  if $||\nabla g_{\text{ec},k}|| \approx 0$ **then**
    Continue ▷ Zero gradient, skip to the next.
  end if
  $\boldsymbol{J}_{\text{eq}}$.rowAppend($\nabla g_{\text{ec},k}^{\top}$)
  if $|\text{cost}| \approx 0$ **then**
    Continue ▷ No need to solve, skip to the next.
  end if
  if $n_{\text{ac}} > 1$ **then** ▷ Perform projection if necessary.
    $\nabla \boldsymbol{g}^* = \text{proj}_{N(\boldsymbol{J}_{\text{eq}})} \nabla g_{\text{ec},k}$
  else
    $\nabla \boldsymbol{g}^* = \nabla g_{\text{ec},1}$
  end if
  if $||\nabla \boldsymbol{g}^*|| \approx 0$ **then**
    Continue ▷ Empty nullspace, skip to the next.
  end if
  stepLength $\leftarrow$ initial_step_length
  while true **do**
    $\boldsymbol{x}^* = \boldsymbol{x} - \text{stepLength} \times \nabla \boldsymbol{g}^*$
    $\text{cost}^* = g_{\text{ec},k}(\boldsymbol{x}^*)$
    if $|\text{cost}^*| > |\text{cost}|$ **then**
      Break ▷ Minimal point has reached.
    end if
    if $\text{sign}(\text{cost}^*) \neq \text{sign}(\text{cost})$ **then**
      ▷ Zero crossing, reverse gradient direction.
      $\nabla \boldsymbol{g}^* = -\nabla \boldsymbol{g}^*$
    end if
    $\boldsymbol{x} \leftarrow \boldsymbol{x}^*$
    stepLength = stepLength × step_multiplier
  end while
end for

2.6 Cost Function Optimization Subroutine

The cost function has the least priority in the optimization problem. The cost is reduced only if there is still any freedom or additional dimension left in the nullspace of the constraint Jacobian space. The cost function optimization algorithm is provided in Alg. 3.

2.7 Overall Framework and Termination Conditions

The proposed overall framework combines Alg. 1-3 along with some set of termination conditions. These conditions are iteration number, step tolerance, and cost tolerance, respectively. Each conditions are checked per outer iteration, that is one iteration of the overall framework altogether with Alg. 1-3. Similarly, the gradient descent iteration in each algorithm loop is called inner iteration.

The step tolerance checks the norm of the optimization parameter vector difference per outer iteration steps. If the parameter change is smaller than a given tolerance, the optimizer terminates. The same approach is also followed for the cost function return value check per outer iteration. The overall framework is provided in Alg. 4.

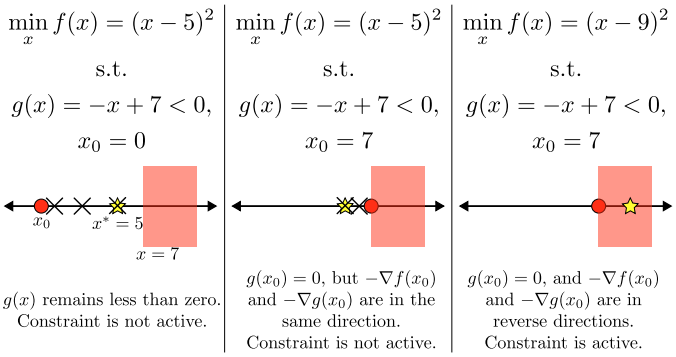

Fig. 1. Inequality constraint activation conditions where the red dot, yellow star, and red region represent the initial condition, desired position, and constraint, respectively. The left problem never reaches the activation point. The middle problem starts from the zero crossing point, but the cost function gradient is already in the same direction as the constraint gradient. Hence, the cost function iteration will not exceed the constraint. The right problem also starts from the zero crossing point, and the cost function and constraint gradients are in reverse directions. The constraint is activated. The cost function gradient should be projected onto the nullspace of the constraint gradient, which results in a zero vector in this case. As a result x remains at x_0.

3. SOFTWARE INTERFACE

The numerical function optimization tool proposed in this paper is provided as a C++ dynamic library for community use. The software interface requires the users to write their optimization problems inside a given *ProblemDescription* object template. There are four predefined functions, whose declaration and name cannot be changed, but only the content. These functions are *costFunction*, *equalityConstraintFunction*, *inequalityConstraintFunction*, and *interimFunction*, respectively. The first three are used for cost and constraint function iterations and calculations. These functions are repeatedly called every per inner iteration for numerical iterations and numerical gradient estimations. Even though the users are free to implement whatever they desire inside these functions' bodies, they should also consider the performance outcomes. An expensive computation inside these functions results in high computational cost. The *interimFunction*, on the other hand, is called once per outer iteration and does not go into numerical gradient estimation. This function is placed to allow users to do less occasional parameter updates for performance considerations. For details and implementations, refer to "NFO.hpp" and "ProblemDescription.hpp" provided in ⋆.

4. NUMERICAL RESULTS

This section analyses five different problems with a parameter set given in Table. 1. Three of these are numerical, and two are robotic system examples. The implementation codes for the first four examples are provided in ⋆. The optimization times correspond to a daily use desktop computer with AMD Ryzen 7 5800X CPU.

Algorithm 2 Optimizing for inequality constraints.

Require: $\boldsymbol{x} \in \mathbb{R}^n$ ▷ From equality constraint algorithm.
Require: $\boldsymbol{J}_{\text{eq}}$, n_{ce} ▷ From equality constraint algorithm.
$\boldsymbol{J}_{\text{ineq}}$.empty()
for ($k \leftarrow 1$; $k \leq n_{\text{ic}}$; ++k) **do**
  $\boldsymbol{J}_{\text{active}} \leftarrow \boldsymbol{J}_{\text{eq}}$
  cost $= g_{\text{ic}}(\boldsymbol{x})$
  if cost < 0 **then**
    Continue ▷ No need to solve, skip to the next.
  end if
  $\nabla g_{\text{ic},k} = \text{numericalGradient}(g_{\text{ic},k}(\boldsymbol{x}))$
  if $||\nabla g_{\text{ic},k}|| \approx 0$ **then**
    Continue ▷ Zero gradient, skip to the next.
  end if
  $n_{\text{ac}} = n_{\text{ac}} + 1$
  $\boldsymbol{J}_{\text{ineq}}$.rowAppend($\nabla g_{\text{ic},k}^{\top}$)
  ▷ Check previous active constraints' grad directions.
  if rowNumber($\boldsymbol{J}_{\text{ineq}}$) > 1 **then**
    for ($i \leftarrow 1; i \leq \text{rowNumber}(\boldsymbol{J}_{\text{ineq}})$; ++i) **do**
      if $\boldsymbol{J}_{\text{ineq}}.\text{row}(i) \cdot \nabla g_{\text{ic},k} < 0$ **then**
        $\boldsymbol{J}_{\text{active}}$.rowAppend($\boldsymbol{J}_{\text{ineq}}.\text{row}(i)$)
      end if
    end for
  end if
  ▷ Then initiate the projection operation.
  if $n_{\text{ac}} > 1$ **then**
    $\nabla \boldsymbol{g}^* = \text{proj}_{N(\boldsymbol{J}_{\text{active}})} \nabla g_{\text{ic},k}$
  else
    $\nabla \boldsymbol{g}^* = \nabla g_{\text{ec},1}$
  end if
  if $||\nabla \boldsymbol{g}^*|| \approx 0$ **then**
    Continue ▷ Empty nullspace, skip to the next.
  end if
  stepLength $\leftarrow$ initial_step_length
  while true **do**
    $\boldsymbol{x}^* = \boldsymbol{x} - \text{stepLength} \times \nabla \boldsymbol{g}^*$
    cost* $= g_{\text{ic},k}(\boldsymbol{x}^*)$
    if cost* < 0 **then**
      Break ▷ Zero crossing has reached.
    end if
    $\boldsymbol{x} \leftarrow \boldsymbol{x}^*$
    stepLength = stepLength $\times$ step_multiplier
  end while
end for

Table 1. List of given optimization parameters.

Parameter Name	Value
initial_step_length	10^{-6}
step_multiplier	2
step_tol	10^{-4}
cost_tol	10^{-4}

4.1 Example 1: A Simple Convex Optimization Problem

Assume the following optimization problem,

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) = \sum_{k=1}^{5} (x_k - k)^2$$
$$\text{such that}$$
$$x_1 + 5 = 0, \;\; x_2 - 5 = 0,$$
$$x_3 + 3 < 0, \;\; x_4 - 3 < 0, \;\; \boldsymbol{x}_0 = \boldsymbol{0}$$

Algorithm 3 Optimizing for the cost function.

Require: $\boldsymbol{x} \in \mathbb{R}^n$ ▷ From inequality constraint alg.
Require: $\boldsymbol{J}_{\text{eq}}$, n_{ce} ▷ From inequality constraint alg.
Require: $\boldsymbol{J}_{\text{ineq}}$ ▷ From inequality constraint alg.
$\boldsymbol{J}_{\text{active}} \leftarrow \boldsymbol{J}_{\text{eq}}$
cost $= f(\boldsymbol{x})$
$\nabla f = \text{numericalGradient}(f(\boldsymbol{x}))$
if $||\nabla f|| \approx 0$ **then**
  Continue ▷ Zero gradient, skip.
end if
▷ Check previous active constraints' grad directions.
if rowNumber($\boldsymbol{J}_{\text{ineq}}$) > 0 **then**
  for ($i \leftarrow 1; i \leq \text{rowNumber}(\boldsymbol{J}_{\text{ineq}})$; ++i) **do**
    if $\boldsymbol{J}_{\text{ineq}}.\text{row}(i) \cdot \nabla f < 0$ **then**
      $\boldsymbol{J}_{\text{active}}$.rowAppend($\boldsymbol{J}_{\text{ineq}}.\text{row}(i)$)
    end if
  end for
end if
▷ Initiate the projection operation if necessary.
if $n_{\text{ac}} > 0$ **then**
  $\nabla \boldsymbol{f}^* = \text{proj}_{N(\boldsymbol{J}_{\text{active}})} \nabla f$
  if $||\nabla f^*|| \approx 0$ **then**
    Continue ▷ Empty nullspace, skip.
  end if
end if
stepLength $\leftarrow$ initial_step_length
while true **do**
  $\boldsymbol{x}^* = \boldsymbol{x} - \text{stepLength} \times \nabla \boldsymbol{f}^*$
  cost* $= f(\boldsymbol{x}^*)$
  if cost* $>$ cost **then**
    Break ▷ Minimal point has reached.
  end if
  ▷ Check if $\boldsymbol{x}^*$ violates any inequality constraint.
  for ($i \leftarrow 1; i \leq n_{\text{ic}}; i = i + 1$) **do**
    if $g_{\text{ineq}}(\boldsymbol{x}^*) > 0$ **then**
      Break the while loop.
    end if
  end for
  $\boldsymbol{x} \leftarrow \boldsymbol{x}^*$
  stepLength = stepLength $\times$ step_multiplier
end while

Algorithm 4 The overall optimization framework.

Require: $\boldsymbol{x}_0$ ▷ Initial condition.
Require: max_iter
Require: step_tol
Require: cost_tol
$\boldsymbol{x} \leftarrow \boldsymbol{x}_0$
for (iter $\leftarrow 1$; iter $\leq$ max_iter; ++iter) **do**
  $\boldsymbol{x}_0 \leftarrow \boldsymbol{x}$
  Call Alg. 1
  Call Alg. 2
  cost$_0$ $\leftarrow f(\boldsymbol{x})$
  Call Alg. 3
  if $||\boldsymbol{x} - \boldsymbol{x}_0|| <$ step_tol **then**
    Exit
  end if
  if $|\text{cost} - \text{cost}_0| <$ cost_tol **then**
    Exit
  end if
end for

where the unconstrained solution is obvious and equal to $[1, 2, 3, 4, 5]$. The constrained optimization solution is $\boldsymbol{x}^* = [-5, 5, -3, 3, 5]^\top$. The first two parameters are equality constrained. Hence, the orthogonal projection of the cost function gradient onto the nullspace of the equality constraint Jacobian space results in zero in the direction of x_1 and x_2. The third and fourth parameters approach their unconstrained values as much as the inequality constraint allows. The fifth parameter, on the other hand, does not have any constraint and converges into its unconstrained optimal value. The optimization problem takes $8\mu s$ and is solved in 6 iterations.

4.2 Example 2: Rosenbrock's Function

Rosenbrock's function is a standard test function in optimization. Finding the minimum is a challenge for some algorithms because the function has a shallow minimum inside a deeply curved valley. Assume the following nonlinear constrained optimization problem with Rosenbrock's function,

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) = 100(x_2 - x_1^2)^2 + (1 - x_1)^2$$
$$\text{such that}$$
$$x_1^2 + x_2^2 \leq 1 \;\text{ and }\; \boldsymbol{x}_0 = [0, 0].$$

The minimum is found at $\boldsymbol{x}^* = [0.7864, 0.6177]^\top$ in 201 iterations which takes $35\mu s$.

4.3 Example 3: Number 71 From the Hock-Schittkowsky Test Suite (Hock and Schittkowski (1980))

A more challenging optimization problem can be found in the Hock-Schittkowsky test suite. Problem HS071 includes a high number of constraints that continuously disturb each other. For the given problem,

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) = x_1 x_4 (x_1 + x_2 + x_3) + x_3$$
$$\text{such that}$$
$$x_1 x_2 x_3 x_4 \geq 25$$
$$x_1^2 + x_2^2 + x_3^2 + x_4^2 = 40$$
$$1 \leq x_1, x_2, x_3, x_4 \leq 5$$
$$x_0 = (1, 5, 5, 1)$$

the minimum is found at $\boldsymbol{x}^* \approx [1.00, 4.74, 3.82, 1.38]^\top$ in 100 iterations which takes $139\mu s$.

4.4 Example 4: A 3-link Arm Configuration Optimization

Assume a three-link planar robotic arm system as shown in Fig. 2. The end effector position $\boldsymbol{p}(\boldsymbol{x})$ is a nonlinear function of optimization parameters:

$$\begin{bmatrix} p_x \\ p_y \end{bmatrix} = \begin{bmatrix} \cos(x_1) + \cos(x_1 + x_2) + \cos(x_1 + x_2 + x_3) \\ \sin(x_1) + \sin(x_1 + x_2) + \sin(x_1 + x_2 + x_3) \end{bmatrix}.$$

Similarly, the gravity vector of the system dynamics is given as

$$\boldsymbol{\tau}_g(\boldsymbol{x}) = \begin{bmatrix} 2.5\cos(x_1) + 1.5\cos(x_1 + x_2) + 0.5\cos(x_1 + x_2 + x_3) \\ 1.5\cos(x_1 + x_2) + 0.5\cos(x_1 + x_2 + x_3) \\ 0.5\cos(x_1 + x_2 + x_3) \end{bmatrix} g.$$

The gravity vector is equivalent to the amount of joint torques required to hold the robot in a static configuration. Hence, an optimization problem can be defined such that the end effector is placed at a desired position with a

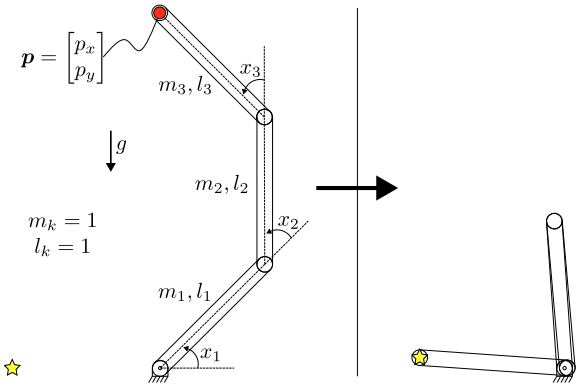

Fig. 2. Left: A three-link planar robot arm system, where the red dot and yellow star represent the initial condition and the desired position, respectively. Right: The result of the optimization problem where the arm folds onto itself to minimize the static torque requirement while satisfying the equality constraint.

configuration that requires a minimum amount of torque to remain stationary. The problem formulation follows

$$\min_{\boldsymbol{x}} f(\boldsymbol{x}) = \boldsymbol{\tau}_g^\top \boldsymbol{\tau}_g$$
$$\text{such that}$$
$$p_x(\boldsymbol{x}) = -1.0, \;\; p_y(\boldsymbol{x}) = 0.0, \;\; \textit{and} \;\; \boldsymbol{x}_0 = [\pi/4 \;\; \pi/4 \;\; \pi/4]^\top.$$

The optimized robot configuration is shown in Fig. 2. The minimum is found at $\boldsymbol{x}^* \approx [1.647, 3.141, -1.647]^\top$ in 8 iterations which takes $42\mu s$.

4.5 Humanoid Robot Posture Optimization

A high dimensional nonlinear robotic optimization problem that is solved with the proposed function optimization framework can be found in Sovukluk et al. (2025). The optimization problem covers finding a set of joint trajectories which results in a proper running motion. More specifically, the optimization problem covers how a running humanoid robot should swing its limbs during the flight phases such that at the next landing, the feet are at desired locations and the torso is kept upright. The snapshots of the optimized trajectories are shown in Fig. 3. The optimization problem includes 32 optimization parameters along with 14 nonlinear constraints and takes $1.92ms$ to solve (Sovukluk et al. (2025)). In the implementation, the *interimFunction* is used effectively for performance considerations, such that, during the trajectory optimization, the system dynamics are updated at the beginning of every outer iteration rather than the inner iterations.

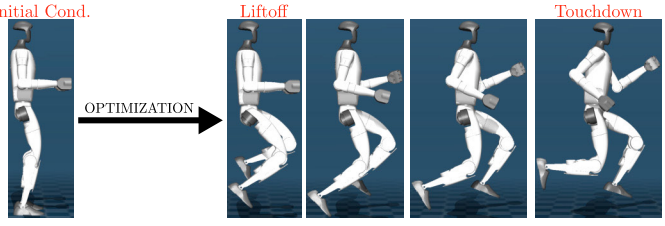

Fig. 3. The snapshots of the optimized trajectory.

5. CONCLUSION

This paper proposes a numerical function optimization framework designed to solve constrained nonlinear robotic problems in real-time. The proposed framework does not require any analytical representation of the problem and can also be referred as a constrained black-box optimization tool. The method combines first-order gradient-based line search algorithms with constraint prioritization through nullspace projections of constraint Jacobian space. Consequently, it usually has a slower convergence rate per iteration as the Hessian is ignored. On the other hand, the computational cost per iteration is much less as it only requires basic numerical gradient calculations. The capability of the framework is proven through numerous complex numerical and robotic problems ranging from a simple robotic arm to a complex humanoid robot.

REFERENCES

Andersson, J.A., Gillis, J., Horn, G., Rawlings, J.B., and Diehl, M. (2019). Casadi: a software framework for nonlinear optimization and optimal control. *Mathematical Programming Computation*, 11, 1–36.

Bambade, A., El-Kazdadi, S., Taylor, A., and Carpentier, J. (2022). Prox-qp: Yet another quadratic programming solver for robotics and beyond. In *RSS 2022-Robotics: Science and Systems*.

Bertsekas, D.P. (1997). Nonlinear programming. *Journal of the Operational Research Society*, 48(3), 334–334.

Carpentier, J., Saurel, G., Buondonno, G., Mirabel, J., Lamiraux, F., Stasse, O., and Mansard, N. (2019). The pinocchio c++ library – a fast and flexible implementation of rigid body dynamics algorithms and their analytical derivatives. In *IEEE International Symposium on System Integrations (SII)*.

Carpentier, J., Valenza, F., Mansard, N., et al. (2015–2021). Pinocchio: fast forward and inverse dynamics for poly-articulated systems. https://stack-of-tasks.github.io/pinocchio.

Dai, H., Valenzuela, A., and Tedrake, R. (2014). Whole-body motion planning with centroidal dynamics and full kinematics. In *2014 IEEE-RAS International Conference on Humanoid Robots*, 295–302. doi: 10.1109/HUMANOIDS.2014.7041375.

Featherstone, R. (2014). *Rigid body dynamics algorithms*. Springer.

Felis, M.L. (2016). Rbdl: an efficient rigid-body dynamics library using recursive algorithms. *Autonomous Robots*, 1–17. doi:10.1007/s10514-016-9574-0.

Gasparetto, A., Boscariol, P., Lanzutti, A., and Vidoni, R. (2015). Path planning and trajectory planning algorithms: A general overview. *Motion and operation planning of robotic systems: Background and practical approaches*, 3–27.

Haldane, D.W., Yim, J.K., and Fearing, R.S. (2017). Repetitive extreme-acceleration (14-g) spatial jumping with salto-1p. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 3345–3351. IEEE.

Hock, W. and Schittkowski, K. (1980). Test examples for nonlinear programming codes. *Journal of optimization theory and applications*, 30, 127–129.

Khatib, O., Jorda, M., Park, J., Sentis, L., and Chung, S.Y. (2022). Constraint-consistent task-oriented whole-body robot formulation: Task, posture, constraints, multiple contacts, and balance. *The International Journal of Robotics Research*, 41(13-14), 1079–1098. doi: 10.1177/02783649221120029.

Lewis, F.L., Vrabie, D., and Syrmos, V.L. (2012). *Optimal control*. John Wiley & Sons.

Mei, Y., Lu, Y.H., Hu, Y., and Lee, C. (2004). Energy-efficient motion planning for mobile robots. In *IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004*, volume 5, 4344–4349 Vol.5. doi:10.1109/ROBOT.2004.1302401.

Nocedal, J. and Wright, S.J. (1999). *Numerical optimization*. Springer.

Paes, K., Dewulf, W., Elst, K.V., Kellens, K., and Slaets, P. (2014). Energy efficient trajectories for an industrial abb robot. *Procedia CIRP*, 15, 105–110. doi:https://doi.org/10.1016/j.procir.2014.06.043. 21st CIRP Conference on Life Cycle Engineering.

Pandala, A.G., Ding, Y., and Park, H.W. (2019). qpswift: A real-time sparse quadratic program solver for robotic applications. *IEEE Robotics and Automation Letters*, 4(4), 3355–3362.

Sovukluk, S., Englsberger, J., and Ott, C. (2023a). Whole body control formulation for humanoid robots with closed/parallel kinematic chains: Kangaroo case study. In *2023 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 10390–10396. doi: 10.1109/IROS55552.2023.10341391.

Sovukluk, S., Englsberger, J., and Ott, C. (2024). Highly maneuverable humanoid running via 3d slip+foot dynamics. *IEEE Robotics and Automation Letters*, 9(2), 1131–1138. doi:10.1109/LRA.2023.3342668.

Sovukluk, S., Ott, C., and Ankaralı, M.M. (2023b). Cascaded model predictive control of underactuated bipedal walking with impact and friction considerations. In *2023 IEEE-RAS 22nd International Conference on Humanoid Robots (Humanoids)*, 1–8. doi: 10.1109/Humanoids57100.2023.10375153.

Sovukluk, S., Schuller, R., Englsberger, J., and Ott, C. (2025). Realtime limb trajectory optimization for humanoid running through centroidal angular momentum dynamics. doi:10.48550/arXiv.2501.17351. URL `https://arxiv.org/abs/2501.17351`.

Stellato, B., Banjac, G., Goulart, P., Bemporad, A., and Boyd, S. (2020). OSQP: an operator splitting solver for quadratic programs. *Mathematical Programming Computation*, 12(4), 637–672. doi:10.1007/s12532-020-00179-2.

Todorov, E., Erez, T., and Tassa, Y. (2012). Mujoco: A physics engine for model-based control. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 5026–5033. IEEE. doi: 10.1109/IROS.2012.6386109.

Wächter, A. and Biegler, L.T. (2006). On the implementation of an interior-point filter line-search algorithm for large-scale nonlinear programming. *Mathematical programming*, 106, 25–57.

Westervelt, E.R., Grizzle, J.W., Chevallereau, C., Choi, J.H., and Morris, B. (2018). *Feedback control of dynamic bipedal robot locomotion*. CRC press.