ZeroNP 求解器

使用说明

2025 年 5 月 2 日

1 求解器介绍

ZeroNP 是本项目设计可求解有约束非凸非线性规划问题的零阶优化求解器，并融合了对特殊问题的定制化算法以提高求解效率。ZeroNP 结合了有限差分法、增广拉格朗日方法以及现代二次规划求解算法，以解决梯度和海森估计、函数近似、噪声过滤和鞍点逃离等关键技术方面的挑战。通过这些创新技术的应用，ZeroNP 旨在实现对大规模难以求导的非线性规划问题的快速高效求解，并为准确率指标优化等典型非线性约束规划问题的求解提供有力支持。经实验验证，ZeroNP 性能达到或超过国际上当前主流的零阶非线性约束规划求解器。

ZeroNP 用于求解一般非线性约束问题。它解决以下问题：

$$\min_x f(x)$$

$$s.t. \quad g(x) = 0$$

$$l_h \le h(x) \le u_h$$

$$l_x \le x \le u_x$$

其中，$f: R^n \to R$，$g: R^n \to R^{m_1}$，$h: R^n \to R^{m_2}$是光滑函数。

该求解器专为黑盒优化设计，即用户无法提供显式的解析表达式。我们建议用户不要在具有显式公式的问题中使用 ZeroNP，因为其他（如一阶或二阶）方法可能比零阶方法更高效。ZeroNP 适用于以下情况：

- 用户只能访问函数的值。
- 计算函数时存在一定噪声。

ZeroNP 由 ZeroNP 团队用 ANSI C 进行改进与实现。目前，ZeroNP 提供 Matlab 和 Python 两种接口可以调用。ZeroNP 求解器下载链接（Github 链接，若因网络问题无法正常访问，可以使用备用链接）：https://github.com/mulimoxi/ZeroNP 。ZeroNP 备用链接：https://gitee.com/muli-moxi/ZeroNP

2 运行环境要求

目前，我们在 macOS、Linux 和 Windows 系统上提供了 MATLAB 和 Python 接口。在安装之前，应先安装支持的 C/C++ 编译器。请参阅编译器，以了解 Matlab 中受支持的 C/C++ 编译器。此外，在线性代数库方面，我们同时支持英特尔数学核心函数库（Intel

MKL）和开源的线性代数包（LAPACK），因此您需要在计算机上提前安装 LAPACK（或 MKL，二选一即可）。注意：在 macOS 系统上，无需安装任何线性代数库，因为我们使用了苹果公司专有的 **Accelerate 框架**。在 Linux 和 Windows 系统上，你可以通过在 **CMakeLists.txt** 文件中配置参数 **LINK_MKL** 来决定是否使用 MKL。如果无法自动找到 LAPACK 或 MKL，你可以在 **CMakeLists.txt** 文件中手动指定路径。

同时我们建议您安装 OSQP，因为使用 OSQP 在部分问题上可以提升求解器的性能。您也可以选择不安装，若没有安装，则使用算法默认的求解器。注：安装 OSQP 后，您还需要指定 OSQP 目录的环境变量。

export OSQP_HOME= your_path_to_osqp

在正式安装前，您需要确保系统中 CMake 版本高于 3.5.1，并可以提前在 ZeroNP 中下载源码以便后续安装。

3 求解器安装方法

下载源码后，用户可以在 ZeroNP/interface/ 目录中找到 Matlab 和 Python 两种接口。下面具体介绍这些接口的安装方法。

3.1 安装：Matlab

下载文件后，用户可以在 interface/Matlab 目录中找到 **make_zeronp.m**。请在 **make_zeronp.m** 中更改 OSQP 和 LAPACK（或 MKL）的具体路径。如果用户想要选择链接 MKL，请使用命令 make_zeronp (1)，否则请使用 make_zeronp (0)。

```
% change your mkl path here
mkl_path = "YOUR_MKL_PATH";
% change your lapack path here
lapack_path = "Your_LAPACK_PATH";
% change your osqp path here
osqp_path = "YOUR_OSQP_PATH";
```

在 Matlab 软件中运行 **make_zeronp.m** 的步骤如下：

```
cd ZERONP_PLUS_ROOT
cd interface/Matlab
make_zeronp
```

如果命令窗口显示：

```
MEX completed successfully.
```

这意味着 ZeroNP 已成功安装。

用户应将路径 **WORKING** 添加到 Matlab 的路径中，其中 **WORKING** 是用户下载源文件的目录。添加方法可参考：

```
addpath(WORKING_PATH);
savepath
```

添加目录后，在 Matlab 中每次可以直接调用 ZeroNP 函数，无需重复指定工作路径。

3.2 安装：Python

对于使用 Python 接口，在安装之前，您需要在 CMakeLists.txt 文件中提前指定 LAPACK（或 MKL）和 OSQP 的路径。如果用户选择链接 MKL，请将 LINK_MKL 设置为 1。

```
# Configure MKL and OSQP roots
set(ENV{MKL_ROOT} YOUR_MKL_PATH)
set(ENV{LAPACK_ROOT} YOUR_LAPACK_PATH)
set(ENV{OSQP_ROOT} YOUR_OSQP_PATH)
```

并需要您**编译** ZeroNP 的**源代码**，编译 ZeroNP 源代码步骤：

在 Linux 或 macOS 上：

```
cd ZERONP_PLUS_ROOT
mkdir build
cd build
cmake ..
cmake --build .
```

在 Windows 上：

```
cd ZERONP_PLUS_ROOT
mkdir build
cd build
cmake -G "MinGW Makefiles" ..
cmake --build .
```

然后，用户可以在 interface/python 文件夹中找到 **setup.py**。

注：首先需要编译源代码，然后您才可以正确地安装和测试 Python 接口：

```
cd ZERONP_PLUS_ROOT
cd interface/Python
python setup.py install
cd test
python test.py
```

如果在执行以上命令后，可以正常输出测试结果，则说明 Python 接口已被正确安装，Python 接口的使用方式将在后文介绍。

4 求解器使用方法

4.1 函数文件

用户应提供一个函数，该函数可以返回在给定点 p 处的目标函数和约束函数值。

函数的输入和输出如下。在 Matlab 和 Python 接口中：

```
function f = cost(p)
```

其中，p 是用于计算函数值的输入变量。$p(:,\mathrm{i})$ 表示第 i 个点。

输出 f 是一个向量。$f(1)$ 是在 x 处计算的目标函数值。$f(2)$到 $f(m_1 + 1)$ 是等式约束的值。$f(m_1 + 2)$ 到 $f(m_1 + m_2 + 1)$ 是不等式约束的值。不等式约束的顺序应与输入一致。

对于梯度（Gradient）或 海森矩阵（Hessian），函数的输出应为列向量 g，其中包含目标函数，等式约束和不等式约束的梯度或海森矩阵。海森矩阵应按列优先（column-first）的方式存储。

```
function f = grad(p) or function f = Hessian(p)
```

这些函数可以作为结构体（structure）存储为输入，详情请见 4.2 节。

4.2 输入

用户可以使用以下命令调用 ZeroNP。

Matlab：

```
info = ZeroNP(prob, options,fun);
```

Python：

```
from pyzeronp import ZERONP
myzeronp = ZERONP(prob=prob, cost=cost, option = options)
info = myzeronp.run()
```

其中，**prob** 和 **options** 是包含问题的信息和算法参数的结构体或字典。**fun** 结构体/字典包含目标函数的信息。

prob 的字段如下：

- **prob.pbl** 是箱约束的下界 (l_x)。默认设置为 −∞。
- **prob.pbu** 是箱约束的上界 (u_x)。默认设置为 +∞。
- **prob.ibl** 是不等式约束的下界 (l_h)。默认设置为 −∞。
- **prob.ibu** 是不等式约束的上界(u_h)。默认设置为 +∞。
- **prob.p0** 是算法的初始点。它应在箱约束的内部。如果用户未提供初始点，且 **prob.pbl** 和 **prob.pbu** 都是有限的，默认的初始点设置为箱的中心点，即 (prob.pbl + prob.pbu) / 2。
- **prob.ib0** 是不等式约束的估计值。它应在不等式约束的内部。如果用户未提供估计值，且 **prob.ibl** 和 **prob.ibu** 都是有限的，则默认的估计值设置为中心点，即 (prob.ibl + prob.ibu) / 2。

例如，如果我们想求解一个带有初始点 (1, 1, 2, 3) 的等式约束问题，可以设置：

```
prob.p0 = [1,1,2,3]';
```

如果我们想求解一个带有不等式约束和箱约束的问题，可以设置：

```
prob.pbl = l_x;
prob.pbu = u_x;
prob.ibl = l_h;
prob.ibu = u_h;
```

具体参数如下所示：

参数	类型	默认值	描述
m	Int64	-1	约束的数量，ZeroNP 将调用 cost 一次来确定（如果未给定）
ibl	Vector{Float64}	-∞	不等式约束的下界
ibu	Vector{Float64}	+∞	不等式约束的上界
ib0	Vector{Float64}	(ibl + ibu) / 2	不等式约束的值估计
pbl	Vector{Float64}	-∞	变量的下界
pbu	Vector{Float64}	+∞	变量的上界
p	Vector{Float64}	(pbl+ pbu) / 2	变量的值估计
cost	Function：Vector{Float64} → Vector{Float64} or Float64	无	包含目标函数、等式和不等式约束的值函数
grad	Function：Vector{Float64} → Vector{Float64} or Nothing	无	包含目标函数、等式和不等式约束的函数的梯度（合并成一个向量）
hess	Function：Vector{Float64} → Vector{Float64} or Nothing	无	包含目标函数、等式和不等式约束的函数的海森矩阵（合并成一个向量）
option	Dict{String, Float64}	默认设置	ZeroNP 的选项
l	Vector{Float64}	零向量	对偶变量的估计
h	Matrix{Float64}	单位矩阵	初始 BFGS 矩阵的估计

fun 的字段如下：

- **fun.cost**: 目标函数。优化必需项。
- **fun.grad**: 梯度函数。可选项。
- **fun.hessian**: Hessian 函数。可选项。

4.3 算法参数介绍

options 中包含算法的参数。options 的字段分为两类：**基本设置**和**高级设置**。如果您不熟悉代码，建议您只更改基本设置。

基本设置：

- **options.tol**: 收敛容忍度。当相对差异小于该值时，算法将停止。默认值为 10^{-4}。
- **options.tol_con**: 不可行性容忍度。默认值为 10^{-3}。
- **options.maxfev**: 最大函数评估次数。默认值为 $500n$，n 是输入维度 p。
- **options.max_iter**: 外迭代的最大次数。默认值为 50。
- **options.min_iter**: 内迭代的最大次数。默认值为 10。
- **options.cen_diff**: 如果设置为 1，算法使用中心差分计算梯度。默认值为 0。建议使用中心差分和块坐标下降的结合（参见 **options.rs**）。
- **options.noise**: 在评估函数值时是否存在噪声。options.noise = 1 表示存在噪声，算法会选择使用自适应 delta 来计算近似梯度。options.noise = 2 表示计算适当的步长来估算噪声级别。options.noise = 3 表示结合了前两种方法，交替使用这两种方法。options.noise = 0 表示没有噪声，算法将使用固定的 delta。默认值为 1。
- **options.delta_end**: delta 的下界。如果用户选择使用自适应 delta，在计算过程中 delta 将不低于 options.delta_end。默认值为 10^{-5}。
- **options.rs**: 在梯度估计中是否使用随机采样。options.rs = 1 表示使用随机采样。options.rs = 2 表示使用块坐标下降。默认值为 options.rs = 0。随机采样可以减少函数开销。然而，在约束优化问题中开启此选项可能导致性能不佳。我们建议在大规模无约束优化问题中开启此选项。
- **options.gd_step**：使用 options.gd_step 来更改步长大小，默认步长大小为 0.1。
- **options.batchsize**: 如果您选择在梯度估计中使用随机采样，算法将会使用 options.batchsize 个样本来估算每次梯度。默认值为 max(1, min(dim/4, 50))。
- **options.rescue**: 如果没有目标函数，或者只是想找到问题的一个可行解，设置 options.rescue = 1。默认值为 options.rescue = 0。
- **options.drsom**: 该选项用于无约束优化，采用 DRSOM 算法。设置 options.drsom = 1 以使用 DRSOM 算法。默认值为 options.drsom = 0。

- **options.qpsolver**: 默认值为 1。
  - **options.qpsolver = 1**: 使用内点法来求解二次规划子问题。
  - **options.qpsolver = 2**: 使用 OSQP 来求解二次规划子问题。

高级设置：

- **options.tol_restart**: 重启容忍度。ZeroNP 使用增广拉格朗日函数的梯度来决定是否重启。请注意，由于零阶方法的限制，当点接近最优点时，梯度也可能会很大。默认值为 1。
- **options.bfgs**: 是否使用 BFGS 更新。options.bfgs = 1 表示使用 BFGS 更新。默认值为 options.bfgs = 1。
- **options.pen_l1**: L1 惩罚参数。默认值为 options.l1_pen = 1。
- **options.k_i**: 步长 δ 扩展比率: $\delta^{k+1} = k_i * \delta^k$。它会随着 k_i 值增大而更快速地增加步长。默认值为 $k_i = 3$。
- **options.k_r**: 步长 δ 缩减比率: $\delta^{k+1} = \delta^k / k_r$。它会随着 k_r 值增大而更快速地减少步长。默认值为 $k_r = 9$。
- **options.c_i**: 用于确定何时应该增加步长 δ。它会随着 c_i 值减小而更快速地增加步长。默认值为 $c_i = 30$。
- **options.c_r**: 用于确定何时应该减少步长 δ。它会随着 c_r 值增大而更快速地减少步长。默认值为 $c_r = 10$。
- **options.rho**: 增广拉格朗日函数的初始惩罚参数。默认值为 1。
- **options.re_time**: 最大重启次数。默认值为 5。
- **options.delta**: 计算近似梯度时的初始 delta。默认值为 1（如果 options.noise = 1）；为 10^{-5}（如果 options.noise = 0）。
- **options.scale**: 是否缩放问题。options.scale = 1 表示在每次迭代中进行缩放。默认值为 options.scale = 1。
- **options.ls_time**: 每次执行线搜索时的最大函数评估次数。默认值为 10。

4.4 输出

输出 **output** 结构体具有以下字段：

- **output.p**: 原始解。
- **output.best_fea_p**: 约束违反最小的点。
- **output.obj**: 目标函数值。
- **output.l**: 对偶解。
- **output.h**: 估计的 Hessian 矩阵。
- **output.jh**: 包含所有函数值历史的数组。数组中的第 i+1 项是经过 i 次外部迭代后的目标值。
- **output.ch**: 包含不可行性历史的数组。数组中的第 i+1 项是经过 i 次外部迭代后的不可行性值。
- **output.iter**: 外迭代次数。
- **output.count_cost**: 函数评估次数。
- **output.count_grad**: 梯度评估次数。
- **output.constraint**: 约束值的范数。
- **output.ic**:估计的不等式约束值。
- **output.solve_time**:问题求解时间。
- **output.status**: 算法的状态
    - **output.status = 1**: 成功。
    - **output.status = 0**: 退出，达到最大函数评估次数。
    - **output.status = -1**: 退出，达到最大迭代次数，容忍度未达到。
    - **output.status = -2**: 退出，达到最大迭代次数，可行性未达到。算法未能收敛。
    - **output.status = -3**: 退出，由于未知的数值错误。

您可以根据算法的输出判断问题是否已经成功求解。若 output.status = 0/-1/-2，我们建议可以适当增加函数评估次数和最大迭代次数，往往能得到更好的解。

4.5 使用示例

用户可以在 ZeroNP /interface/ 目录中找到各个对应接口的一些测试示例。这里展示 Python 接口和 Matlab 接口的使用实例：

Python 接口：

代码：

```
from pyzeronp import ZERONP
import numpy as np

# def cost(x: np.ndarray, par: int):
def cost(x: np.ndarray):
    # x is a vector of length 2
    f1 = (x[0]-5)**2 + x[1]**2-25
    f2 = -x[0]**2 + x[1]
    f = np.array([f1, f2])

    return f

# define initial point and bound
p0 = [4.9, .1]
ib0 = [1]
ibl = [0]

# define  prob
prob = {'p0': p0, 'ib0': ib0, 'ibl': ibl}

# use the solver
myzeronp = ZERONP(prob=prob, cost=cost)
solution = myzeronp.run()
print(solution)
```

输出结果：

```
ZeroNP--> The user does not provided gradient function of cost in the fun structure.
ZeroNP--> ZERONP uses zero-order method instead.
ZeroNP--> Iteration 1: obj = 4.9500e+02, infeasibility = 0.0000e+00
ZeroNP--> Iteration 2: obj = 4.9500e+02, infeasibility = 0.0000e+00
ZeroNP--> Iteration 3: obj = -8.0927e+00, infeasibility = 0.0000e+00
ZeroNP--> Iteration 4: obj = -8.4887e+00, infeasibility = 0.0000e+00
ZeroNP--> Iteration 5: obj = -8.4931e+00, infeasibility = 0.0000e+00
ZeroNP--> Iteration 6: obj = -8.4980e+00, infeasibility = 0.0000e+00
ZeroNP--> Success! Completed in 7 iterations
      The infeasibility is 0.000000e+00.
ZeroNP--> ZERONP finished.
{'iter': 7, 'count_cost': 56, 'count_grad': 0, 'count_hess': 0, 'constraint': 0.0, 'restart_time': 0, 'obj': -8.498273935826045, 'status': 1, 'solve_time': 0.009854155, 'qpsolver_time': 0.0007748750000000001, 'p': array([1.23474806, 1.52466516]), 'best_fea_p': array([ 4.69899762, 22.80146964]), 'ic': array([1.23474806]), 'jh': array([-24.98 , -10.72663464, -11.56983557, -8.84301334,
      -8.48866284, -8.49305664, -8.49795992]), 'ch': array([24.91 , 6.13138031, 1.60780245, 0.14795925, 0. ,
      0. , 0. ]), 'count_h': array([ 1., 14., 29., 38., 44., 47., 53.]), 'l': array([3.04944187]), 'h': array([3.04944187])}
```

Matlab 接口：

在 Matlab 中我们将测试 cutest 问题，因此需提前安装 macup(cutest 问题在 Matlab 中的接口)，安装链接：[MatCUTEst - File Exchange - MATLAB Central](#)。

代码：

```
% MATLAB implementation
% solve cutest problem

% load problem
prob = 'HS1'
problem = macup(prob);

% Get initial point from test data
x0 = problem.x0;

% Extract problem dimensions
n = length(x0);
m = problem.numcon;
```

```
% Setup ZeroNP problem structure
ZeroNP_prob = struct();
ZeroNP_prob.p0 = x0;

% Set bounds if they exist
if ~isempty(problem.lb)
   ZeroNP_prob.pbl = problem.lb ;
end
if ~isempty(problem.ub)
   ZeroNP_prob.pbu = problem.ub;
end

% Setup constraints for ZeroNP
ibl = [];
ibu = [];

% Handle linear constraints
if ~isempty(problem.Aeq)
   n_lin_eq = size(problem.Aeq, 1);
   ibl = [ibl; problem.beq];
   ibu = [ibu; problem.beq];
end

if ~isempty(problem.Aineq)
   n_lin_ineq = size(problem.Aineq, 1);
   ibl = [ibl; -Inf(n_lin_ineq, 1)];
   ibu = [ibu; problem.bineq];
end

% Handle nonlinear constraints
if problem.numnlcon > 0
   % Nonlinear inequality constraints
   if problem.numnlineq > 0
      ibl = [ibl; -Inf(problem.numnlineq, 1)];
      ibu = [ibu; zeros(problem.numnlineq, 1)];
   end

   % Nonlinear equality constraints
   if problem.numnleq > 0
      ibl = [ibl; zeros(problem.numnleq, 1) - 1e-6];
      ibu = [ibu; zeros(problem.numnleq, 1)];
   end
end
```

```
% Set constraint bounds if any constraints exist
if ~isempty(ibl)
    ZeroNP_prob.ibl = ibl;
    ZeroNP_prob.ibu = ibu;
end

% Define the objective and constraint function
fun.cost = @(x) ZeroNP_cost(x, problem);

% Get ZeroNP options from test data
ZeroNP_options = struct();
ZeroNP_options.tol = 1e-4;
ZeroNP_options.qpsolver = 1;

ZeroNP_details = ZeroNP(ZeroNP_prob, ZeroNP_options, fun);
fprintf('Successfully solved problem: %s\n', prob);
fprintf('ZeroNP_time %s\n', ZeroNP_details.solve_time);
fprintf('ZeroNP_obj %s\n', ZeroNP_details.obj);

% Define the cost function for ZeroNP
function result = ZeroNP_cost(x, problem)
    % Calculate objective
    f = problem.objective(x);

    % Initialize constraint vectors
    c_all = [];

    % Handle linear constraints first
    if ~isempty(problem.Aeq)
        c_eq_lin = problem.Aeq * x ;
        c_all = [c_all; c_eq_lin];
    end
    if ~isempty(problem.Aineq)
        c_ineq_lin = problem.Aineq * x ;
        c_all = [c_all; c_ineq_lin];
    end

    % Handle nonlinear constraints if they exist
    if problem.numnlcon > 0
        [c_ineq, c_eq] = problem.nonlcon(x);

        % Add nonlinear constraints to the constraint vector
        if ~isempty(c_ineq)
            c_all = [c_all; c_ineq];
        end
        if ~isempty(c_eq)
            c_all = [c_all; c_eq];
        end
    end

    % Combine objective and constraints
    if isempty(c_all)
        result = f;
    else
        result = [f; c_all];
    end
end
```

输出结果：

```
ZeroNP--> The user does not provided gradient function of cost in the fun structure.
ZeroNP--> ZERONP uses zero-order method instead.
ZeroNP--> Iteration 1: obj = 4.0000e+00, infeasibility = 0.0000e+00
ZeroNP--> Iteration 2: obj = 4.0000e+00, infeasibility = 0.0000e+00
ZeroNP--> Iteration 3: obj = 4.0000e+00, infeasibility = 0.0000e+00
ZeroNP--> Iteration 4: obj = 3.9904e+00, infeasibility = 0.0000e+00
ZeroNP--> Iteration 5: obj = 3.9846e+00, infeasibility = 0.0000e+00
ZeroNP--> Iteration 6: obj = 1.1045e+00, infeasibility = 0.0000e+00
ZeroNP--> Iteration 7: obj = 7.1820e-01, infeasibility = 0.0000e+00
ZeroNP--> Iteration 8: obj = 2.5548e-01, infeasibility = 0.0000e+00
ZeroNP--> Iteration 9: obj = 8.9312e-02, infeasibility = 0.0000e+00
ZeroNP--> Iteration 10: obj = 7.4193e-02, infeasibility = 0.0000e+00
ZeroNP--> Iteration 11: obj = 4.5830e-02, infeasibility = 0.0000e+00
ZeroNP--> Iteration 12: obj = 1.5009e-02, infeasibility = 0.0000e+00
ZeroNP--> Iteration 13: obj = 2.0351e-03, infeasibility = 0.0000e+00
ZeroNP--> Iteration 14: obj = 4.0344e-04, infeasibility = 0.0000e+00
ZeroNP--> Iteration 15: obj = 9.9906e-06, infeasibility = 0.0000e+00
ZeroNP--> Success! Completed in 16 iterations
      The infeasibility is 0.000000e+00.
total time is:5.416974e-03
Successfully solved problem: HS1
ZeroNP_time 5.416974e-03
ZeroNP_obj 9.069215e-06
```

4.6 结果分析

在算法完成迭代过程后，应谨慎检查最终解是否可行且最优（通过一定的常识与经验）。需要注意的是，算法并不保证最终解一定是全局最优解。如果算法在达到最大迭代次数时终止，您可以尝试重启 ZeroNP。