SOLNP+ User Manual

SOLNP+ Team: Jiyuan Tan, Jinsong Liu, Tianhao Liu

December 11, 2024

1 Brief Introduction

SOLNP+ is a derivative-free solver proposed by Professor Yinyu Ye for general nonlinear constrained problems in 1989 (Origin name SOLNP), which was first implemented in Matlab. It solves the following problem,

$$\begin{aligned} \min_{x} \ & f(x) \\ s.t. \ & g(x) = 0, \\ & l_h \leqslant h(x) \leqslant u_h, \\ & l_x \leqslant x \leqslant u_x, \end{aligned}$$

where $f : \mathbb{R}^n \to \mathbb{R}, g : \mathbb{R}^n \to \mathbb{R}^{m_1}, h : \mathbb{R}^n \to \mathbb{R}^{m_2}$ are smooth functions.

The solver is designed for black-box optimization, where the user cannot provide an explicit analytic expression. We suggest that users should not use SOLNP+ for problems with explicit formulas as other (first-order or second-order) methods may be more efficient than the zero-order methods. SOLNP+ is suitable for the following situations.

- The user only has access to the value of the functions.
- There are some noises when calculating the functions.

Based on the original code of Professor Ye, SOLNP+ is implemented by the SOLNP+ team in ANSI C with some improvements. Currently, SOLNP+ provides Matlab, C, Python and Julia interface.

2 Installation

2.1 Prerequisites

Before installation, a supported C/C++ compiler should be installed. See compiler for supported C/C++ compiler in Matlab. Besides, you need to install LAPACK (or MKL) LAPACK and OSQP in your computer. The CMake version should be later than 3.5.1..

2.2 Download

The user can download the source in SOLNP+.

2.3 Installation: Matlab

After downloading the files, the user can find make_solnp.m in the dictionary. Change the path of OSQP and LAPACK (or MKL) in make_solnp.m. If the user choose to link MKL, please use command make_solnp(1). Otherwise, please use make_solnp(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";
```

Run make_solnp.m with the Matlab software. If the command box shows

```
MEX completed successfully.
```

it means SOLNP+ has been successfully installed. The user should add path WORKING to Matlab path, where WORKING is the dictionary where the user download the source file.

2.4 Installation: Python

Before installation, you need to specify the directories for LAPACK(or MKL) and OSQP in CMakeLists.txt. If the user choose to link MKL, please set

LINK_MKL as 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)
```

The user can find the setup.py in the interface/python folder.
First, the user need to compile SOLNP+ source code.
On Linux or macOS:

```
cd SOLNP_PLUS_ROOT
mkdir build
cd build
cmake ..
cmake --build .
```

On Windows:

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

After compiling source code, you can Install and test Python Interface:

```
cd SOLNP_PLUS_ROOT
cd python
python setup.py install
cd test
python test.py
```

2.5 Installation: Julia

To install SOLNP+ Julia API, you need to do the following steps:

1. Downloading SOLNP+ source code and compile C code as a dynamic library libsolnp.
2. Set LIBSOLNP_PATH as the absolute path of libsolnp in the file SOLNP/julia/SOLNP/src/utilities/SOLNP_CONST.jl.
3. Type in julia in command line to start a Julia REPL. Then type in] to enter Pkg mode and type in dev [path to SOLNP/julia/SOLNP]. Type in Backspace to leave Pkg mode.
4. Now you can type in using SOLNP to use SOLNP+ Julia API.

To test whether everything is ready, try to run test file in Julia REPL by typing in

```
julia> include("julia/SOLNP/test/test.jl")
```

Succeed if you see

```
    -----Rosen-----
SOLNP+--> The user does not provided gradient
   function of cost in the fun structure.
SOLNP+--> SOLNP uses zero-order method instead.
SOLNP+--> Success! Completed in 13 iterations
         The infeasibility is 0.000000e+00.
SOLNP+--> SOLNP finished.
[0.996790382657634, 0.9935672082328988]
130


-----Cost-----
SOLNP+--> The user does not provided gradient
   function of cost in the fun structure.
SOLNP+--> SOLNP uses zero-order method instead.
SOLNP+--> SOLNP finished.
[1.0000616906864084, -2.6673343399478595e-5]
```

3 SOLNP+ Usage

3.1 Function File

The user should provide a function that returns the objective and constraint function value at given point p.

The input and output of the function is as following.In Matlab and Python interface,

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

p is the input variable used to evaluate the function values. $p(:, i)$ is the i-th points.

The output f is a vector. $f(1)$ is the value of the objective function evaluated at x. $f(2)$ through $f(m_1+1)$ are the values of the equality constraints, and $f(m_1+2)$ through $f(m_1+m_2+1)$ are the values of the inequality con-

straints. The order of the inequality constraints should be consistent with the input.

For gradient/Hessian, the output of the function should be a column vector g which contains the gradient/Hessian of objective, equality constraints and inequality constraints in order. The Hessian matrix should be stored in a column-first way.

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

These functions can be stored in a structure as input(See (b) for details).

3.2 Input

The user can use the following command to call SOLNP+.

C

```
SOLNP_PLUS(
    SOLNPProb *prob,
    SOLNPSettings *options,
    SOLNPSol *output,
    SOLNPInfo *info,
    cost_temple *cost_fun,
    g_temple *grad_fun,
    h_temple *hess_fun,
    solnp_float *l,
    solnp_float *h)
```

grad_fun and hess_fun are the the gradient and Hessian of objective and constraint functions. If the users would like to use zero-order method, they can set the variables to be SOLNP_NULL. l and h are initiate values of the Lagrangian multiplier and Hessian, which can also be set to SOLNP_NULL if the users do not have any information. prob and options are introduced below. output and info are the output information of SOLNP+, which are introduced in output section. The user need to free prob,options, output and info at the end of the program. See the example test file for details.

Matlab:

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

Python:

```
from pysolnp import SOLNP
mysolnp = SOLNP(prob=prob, cost=cost, option =
   options)
info = mysolnp.run()
```

where prob and options are structures/dictionary that contains the information of the problem and parameters of the algorithm. fun structure/-dictionary contains the information of objective function. The fields of prob are

- prob.pbl is the lower bound of the box constraints (l_x in (3)). The default setting is $-\infty$.
- prob.pbu is the upper bound of the box constraints (u_x in (3)).The default setting is $+\infty$.
- prob.ibl is the lower bound of the inequality constraints (l_h in (3)). The default setting is $-\infty$.
- prob.ibu is the upper bound of the inequality constraints (u_h in (3)).The default setting is $+\infty$.
- prob.p0 is the initiate point of the algorithm. It should be within the interior of the box constraints. If the user does not provide initiate point and both prob.pbl and prob.pbu are finite, the default initiate point is set to be the center point of the box, i.e., (prob.pbl + prob.pbu)/2.
- prob.ib0 is the estimated inequality value of the inequality constraints. It should be within the interior of the inequality constraints. If the user does not provide the estimation and both prob.ibl and prob.ibu are finite, the default estimation is set to be the center point of the box, i.e., (prob.ibl + prob.ibu)/2.

For example, if we want to solve a equality constrained problem with initiate point $(1, 1, 2, 3)$, we may set

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

If we want to solve a problem with inequality constraints and box constraints, we may set

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

Julia:

```
function solnp_solve(;
  m::Int64=-1,
  ibl::Vector{Float64}=Vector{Float64}(),
  ibu::Vector{Float64}=Vector{Float64}(),
  ib0::Vector{Float64}=Vector{Float64}(),
  pbl::Vector{Float64}=Vector{Float64}(),
  pbu::Vector{Float64}=Vector{Float64}(),
  p::Vector{Float64}=Vector{Float64}(),
  cost,
  grad=nothing,
  hess=nothing,
  option::Dict{String,Float64}=Dict{String,
     Float64}(),
  l::Vector{Float64}=Vector{Float64}(),
  h::Matrix{Float64}=Matrix{Float64}(I, 0, 0)
)
```

Arguments are presented as below.
options contains the parameters of the algorithm. The fields of options

Argument	Type	Default	Description
`m`	`Int64`	-1	Number of constraints, SOLNP will call `cost` once to determine it if not given.
`ibl`	`Vector{Float64}`	$-\infty$	Lower bound for inequality constraints
`ibu`	`Vector{Float64}`	$+\infty$	Upper bound for inequality constraints
`ib0`	`Vector{Float64}`	$\frac{ibl+ibu}{2}$	Value estimation for inequality constraints
`pbl`	`Vector{Float64}`	$-\infty$	Lower bound for variables
`pbu`	`Vector{Float64}`	$+\infty$	Upper bound for variables
`p`	`Vector{Float64}`	$\frac{pbl+pbu}{2}$	Value estimation for variables
`cost`	`Function`: `Vector{Float64}` $\rightarrow$ `Vector{Float64}` or `Float64`		Value of functions containing objective, equality and inequality constraints
`grad`	`Function`: `Vector{Float64}` $\rightarrow$ `Vector{Float64}` or `Nothing`	nothing	Gradient (concatenated as a vector) of functions containing objective, equality and inequality constraints
`hess`	`Function`: `Vector{Float64}` $\rightarrow$ `Vector{Float64}` or `Nothing`	nothing	Hessian (concatenated as a vector) of functions containing objective, equality and inequality constraints
`option`	`Dict{String, Float64}`	Default settings	Options for SOLNP+
`l`	`Vector{Float64}`	Zero vector	Estimate of dual variables
`h`	`Matrix{Float64}`	Identity matrix	Estimate of initial BFGS matrix

are divided into two categories, basic setting and advance setting. If you are not familiar with the code, we suggest that you only change the basic setting.

Basic Settings:

- options.tol: Convergence tolerance. The algorithm will stop when the relative difference is less than options.tol. Default value 10^{-4}.
- options.tol_con: Tolerance of infeasibility. Default value 10^{-3}.
- options.maxfev: Maximum number of function evaluations. Default value $500n$. n is the dimension of the input p.
- options.max_iter: Maximum number of outer iterations. Default value 50.
- options.min_iter: Maximum number of inner iterations. Default value 10.
- options.cen_diff: If equal to 1, the algorithm uses center difference to calculate gradient. Default value 0. We suggest using center difference with block coordinate descent (See options.rs).
- options.noise: whether there are noises when evaluating function value. options.noise = 1 means there are noises and the algorithm will choose

to use adaptive delta when computing approximated gradient. options.noise = 2 means calculating a proper step size by estimating the noise level. options.noise = 3 combines previous two methods, using the two methods alternately. options.noise = 0 means there are no noise and the algorithm will use fixed delta. Default value 1.

- options.delta_end: The lower bound of delta. If the user chooses to use adaptive delta, delta will not be lower than options.delta_end during calculation. Default value 10^{-5}.

- options.rs: Use random sampling in gradient estimation or not. options.rs=1 means using random sampling. options.rs=2 means using block coordinate. Default value options.rs = 0. Random sampling can reduce function cost. However, turn on this option in constrained optimization problems may lead to poor performance. We suggest to turn on this option in large-scale unconstrained optimization problem. Use options.gd_step to change the step size. The default step size is 0.1.

- options.batchsize: If you choose to use random sampling in gradient estimation, the algorithm will use options.batchsize samples to estimate gradient each time. Default value options.batchsize = $\max(1, \min(\dim/4, 50))$.

- options.rescue: If you don't have an objective function or you just want to find a feasible solution of the problem, set options.rescue = 1. Default value options.rescue = 0.

- options.qpsolver: Default value 1.

  - options.qpsolver = 1: Use interior method to solve QP subproblem.
  - options.qpsolver = 2: Use OSQP to solve QP subproblem.

Advanced Settings:

- options.tol_restart: Tolerance of restart. SOLNP+ uses gradient of augmented Lagrange function to decide whether to restart. Mind that due to limitation of zero-order method, the gradient may be large even the point is close to the optimal point. Default value 1.

- options.bfgs: Use BFGS Update or not. options.bfgs=1 means using BFGS update. Default value options.bfgs = 1.
- options.pen_l1: The l1 penalty parameter. Default value options.l1_pen = 1.
- options.k_i: The expand ratio of step size δ: $\delta^{k+1} = k_i \cdot \delta^k$. It will increase more quickly with a larger k_i value. Default value $k_i = 3$.
- options.k_r: The reduce ratio parameter of step size δ: $\delta^{k+1} = \delta^k / k_r$. It will reduce more quickly with a larger k_r value. Default value $k_r = 9$.
- options.c_i: Used for determine when the step size δ should increase. It will increase more quickly with a smaller c_i value. Default value $c_i = 30$.
- options.c_r: Used for determine when the step size δ should reduce. It will reduce more quickly with a larger c_r value. Default value $c_r = 10$.
- options.rho: The initiate penalty parameter of augmented Lagrange function. Default value 1.
- options.re_time: Maximum number of restart time. Default value 5.
- options.delta: The initiate delta when computing approximated gradient. Default value: 1 if options.noise = 1, 10^{-5} if options.noise = 0.
- options.scale: Scale the problem or not. options.scale = 1 means scaling in each iteration. Default value options.scale = 1.
- options.ls_time: Maximum number of function evaluations when performing line search each time. Default value 10.

The fields of fun structure are

- fun.cost: the objective function. Necessary for optimization.
- fun.grad: the gradient function. Optional for optimization.
- fun.hessian: the hessian function. Optional for optimization.

3.3 Output

The output structure has the following fields

- output.p: The primal solution.
- output.best_fea_p: Point with the smallest constraints violation.
- output.obj: Objective function value.
- output.l: The dual solution.
- output.h: The estimated Hessian matrix.
- output.jh: An array contains the history of all primal value. The i+1th entry of the array is objective value after i outer iterations.
- output.ch: An array contains the history of infeasibility. The i+1th entry of the array is infeasibility after i outer iterations.
- output.iter: Number of outer iteration.
- output.count_cost: Number of function evaluations.
- output.count_grad: Number of gradient evaluations.
- output.constraint: The norm of the constraints value.
- output.ic: Estimated inequality constraints value.
- output.status: Status of the algorithm
  - output.status = 1: Success
  - output.status = 0: Exiting after maximum number of function evaluation.
  - output.status = -1: Exiting after maximum number of iterations. Tolerance not achieved.
  - output.status = -2: Exiting after maximum number of iterations. Tolerance of infeasibility achieved. The algorithm fails to converge.
  - output.status = -3: Exiting because of unknown numerical error.

In the C version, the info provides information on the running time.

- info.total_time: total running time.
- info.qpsolver_time: total running time of QP solver.
- info.cost_time: total running time of calculating objective function.

3.4 Examples

The user can find some test examples in the dictionary ./interface .

3.5 Result Analysis

After the algorithm has completed the iterative process, you should be careful to check the

final solution to see if it is feasible and optimal by common sense. There is no guarantee that the

final solution is the global solution. Restart SOLNP+ if it terminates at the maximum number of major iterations.

4 SOLNP+ Algorithm

By adding slack, we can turn the problem into

$$\begin{aligned} &\min_x f(x) \\ &s.t.\ g(x) = 0, \\ &\quad l_x \leqslant x \leqslant u_x. \end{aligned}$$

We use ALM framework to solve the problem. At each step, we solve the following linearized problem of (4).

$$\begin{aligned} \min_x \quad & F_k(x) = f(x) - y_k^T(g(x) - \mathcal{L}g(x, x_k)) + \frac{\rho}{2}\|g(x) - \mathcal{L}g(x, x_k)\|^2 \\ \text{subject to} \quad & \mathcal{L}g(x, x_k) = 0 \\ & l_x \leq x \leq u_x, \end{aligned}$$

where (x^k, y^k) are the primal point and Lagrange multiplier we obtain in the previous step. ρ is the parameter of augmented Lagrangian. To solve

(5), we use quasi newton method. First, we use difference of function value to approximate the gradient, i.e.,

$$(\nabla F_k(z))_i \approx (\nabla_{\delta_k} F_k(z))_i = \frac{F_k(z+\delta_k e_i) - F_k(z)}{\delta_k},$$

where δ is a small scale. SOLNP+ first finds an interior feasible point as an initiate point p_k^0 and solve the following QP at each inner iteration.

$$\begin{aligned} \min_p \quad & \frac{1}{2}(p-p^i)^T H_k^i (p-p^i) + \langle \nabla_{\delta_k} F_k(p^i), p-p^i \rangle \\ \text{subject to} \quad & \mathcal{L}g(p, x^k) = 0 \\ & l_x \le p \le u_x, \end{aligned}$$

where p_k^i is the last iteration point in inner iteration and we use BFGS update for H_k^i, i.e.,

$$H_k^{i+1} = H_k^i + \frac{s_g s_g^T}{s_g^T s_x} - \frac{H_k^i s_x (H_k^i s_x)^T}{s_x^T H_k^i s_x}, \quad H_1^0 = I,$$

where $s_x = p_k^{i+1} - p_k^i$, $s_g = \nabla_{\delta_k} F(p_k^{i+1}) - \nabla_{\delta_k} F(p_k^i)$.

Suppose u is the solution of (6). Let $\widehat{p_k^i} = p_k^i + u$. We use line search to obtain p_k^{i+1}:

$$p_k^{i+1} = \alpha^* p_k^i + (1-\alpha^*)\widehat{p_k^i}, \qquad \alpha^* = \arg\min_{\alpha \in [0,1]} F_k(\alpha p_k^i + (1-\alpha)\widehat{p_k^i}).$$