The ‘Quick-Start’ manual for the SAO*i* algorithm

Albert A. Groenwold*

Department of Mechanical Engineering, University of Stellenbosch, Stellenbosch, South Africa.

Version 0.9.0. Released at 12h39 on February 4, 2017.

1 Problem statement

The SAO*i* algorithm is aimed at large-scale nonlinear inequality-constrained optimization problems $\mathrm{P}_{\mathrm{NLP}}$ of the form

$$\begin{aligned} \min \quad & f_0(\boldsymbol{x}) & & \\ \text{subject to} \quad & f_j(\boldsymbol{x}) \leq 0, & & j = 1, 2, \cdots, n_i, \\ & \check{x}_i \leq x_i \leq \hat{x}_i, & & i = 1, 2, \cdots, n, \end{aligned} \tag{1}$$

where $f_0(\boldsymbol{x})$ represents a real-valued scalar objective function, and the $f_j(\boldsymbol{x})$ represent n_i real-valued scalar *inequality* constraint functions[1]. $f_0(\boldsymbol{x})$ and the $f_j(\boldsymbol{x})$ depend on the n real (design) variables $\boldsymbol{x} = \{x_1, x_2, \cdots, x_n\}^T \in \mathcal{X} \subset \mathcal{R}^n$, hence $\check{x}_i$ and $\hat{x}_i$ respectively indicate lower and upper bounds on variable x_i. The functions $f_j(\boldsymbol{x})$, $j = 0, 1, 2, \cdots, n_i$ are assumed to be (at least) once continuously differentiable.

More specifically: the algorithm is aimed at large scale ‘simulation-based’ optimization, understood to be optimization problems with computationally demanding numerical simulations or models in the optimization loop. Typical examples include the optimization of systems or structures modeled using the finite element method (FEM), computational fluid dynamics (CFD) simulations, etc.

2 Algorithm structure

Using the SAO*i* algorithm is very simple: it merely requires the modification of the three ‘user’ subroutines mentioned in the following.

2.1 User routines

The user routines to be modified by the user are:

- `Initialize.f`
- `Functions.f`
- `Gradients.f`

*Versions earlier than 7.0.0 were documented in collaboration with L.F.P. Etman, *Department of Mechanical Engineering, Eindhoven University of Technology, Eindhoven, the Netherlands.*

[1] It is not required that constraint functions are present. Viz., simple bound constrained problems may also be considered. However, it is *required* that the primal design variables x_i are bounded; unbounded problems may be optimized using artificial large bounds.

Subroutine `Gradients.f` is only required if the parameter `finite_diff` in the routine `Initialize.f` is set to `false`, else finite differences are resorted to. (It is normally preferable to not compute the gradients using finite differences, if at all possible.)

The three user routines mentioned above are briefly discussed in the following.

2.1.1 Initialize.f

In `Initialize.f` it is only necessary to specify the number of design variables `n`, the number of inequality constraints `ni`, the starting points `x(i)`, and the lower and upper bounds `x_lower(i)` and `x_upper(i)` respectively. Note that the bounds *have* to be specified. If no bounds are present, suitably 'large' numbers may be used.

It is possible to specify a few optional parameters. Of these, the most important are briefly discussed below:

- Set `approx_f=1` to select an approximation function appropriate for quadratic-like objective functions, and `approx_f=4`, `5` or `6` for monotonic-like objective functions (the default is `1`).
- Set `approx_c=1` to select an approximation function appropriate for quadratic-like constraint functions, and `approx_c=4`, `5` or `6` for monotonic-like constraint functions (the default is `1`).
- Set `force_converge=0` to not enforce convergence, and `force_converge=1`, `2` or `3` to enforce convergence (the default is `0`).
- Set `finite_diff=.true.` to use finite differences (the default is `.false.`).

For additional information, see the file `Initialize.f` in the software distribution, and also the `'Not-So-Short'` manual.

2.1.2 Functions.f

In `Functions.f` it is required to specify the objective functions f_0, as well as the constraint functions f_j, $j = 1, 2, \cdots, m$.

For additional information, see the file `Functions.f` in the software distribution.

2.1.3 Gradients.f

`Gradients.f` is only needed if the parameter `finite_diff` in `Initialize.f` is set to `.false.`

In this case, it is required to specify the partial derivatives of objective function $\partial f(\boldsymbol{x})/\partial x_i$, $i = 1, 2, \cdots n$, and the partial derivatives of the inequality constraints $\partial f_j(\boldsymbol{x})/\partial x_i$, $j = 1, 2, \cdots n_i$, $i = 1, 2, \cdots n$.

For additional information, see the file `Gradients.f` in the software distribution.

2.2 Output

A number of output files are created by the SAO*i* algorithm, namely:

- `Variables.out`

  This file lists the initial point $x_i^{\{0\}}$, the final point $x_i^{\{*\}}$, and the lower and upper bounds $\check{x}_i$ and $\hat{x}_i$ respectively for each component $i = 1, 2, \cdots, n$.

- `History.out`

  This file lists the function values, maximum constraint violations, step size information, and the number of active bounds and constraints as the iterations proceed. This file echoes the data written to the screen.

- `Tolerance-X.out`

  This file lists the achieved tolerance w.r.t. the primal variables $\boldsymbol{x}$.

- `Tolerance-KKT.out`

  This file lists the achieved tolerance w.r.t. the KKT residual.

- `Constraints.out`

  This file lists the final values of the constraints and their associated dual variables.

- `Warnings.out`

  This file lists (non-fatal) warnings issued during execution, which may or may not influence convergence, but also fatal errors. For the warnings, severity 0 implies that convergence is definitely not impaired, whereas severity 10 implies that convergence has almost certainly been impaired.

3 Installation and 'make'

3.1 On Linux or Unix based systems

- Download the file `SAOi.tar.gz` into a suitable directory, and unpack this file (using `tar -xvzf SAOi.tar.gz`).
- If the gfortran compiler is available, simply type 'make' at the command prompt.
- If the gfortran compiler is not available, edit the file `Makefile` to select a suitable compiler and then type 'make'.
- Then run the executable `SAOi`. This will run the example problem included in the current version of the algorithm.

For the sake of interfacing with external solvers, the installation also assumes that a C-compiler is available (typically, the *GNU gcc* compiler).

3.2 On other platforms

Some minor editing of the Makefile will probably be required.

4 Availability, conditions of use and everything else

See the `'Not-So-Short'` manual.