Anadarko Summer Internship 2017

FIDO OPTIMIZATION FRAMEWORK MANUAL

Yimin Liu
Mentor - Sathish Sankaran
Supervisor - Christian Noll
Anadarko

September 8th, 2017

I Installation

On Linux:

1. Install Anaconda Python 2.7 version.

2. Add Anaconda2 path to environmental variable.

```
export PATH = "[Anaconda2]/bin:$PATH"
```

3. Compile PSO-MADS code.

```
cd [PSO−MADS]/src
make
```

   To compile in parallel, such as with 8 processors

```
make −j8
```

4. Add PSO-MADS path to environmental variables.

```
export NOMAD_HOME = [PASO−MADS]
export PATH = "[PSO−MADS]/bin:$PATH"
```

5. Install the PyNomad python package

```
cd [PSO−MADS]/pyNomad_Beta
python setup_PyNomad.py install
```

6. Install the fido_opt python package

```
cd [FIDO]
pip install .
```

7. To run FIDO in parallel, install the mpi4py python package.

   First options (recommended):

```
conda install −c anaconda mpi4py
```

   Besides the mpi4py package, this will also install the mpich2 package. It will create two executables – mpirun and mpiexec – under [Anaconda2]/bin. The mpich2 works well with mpi4py, therefore the first option is recommended.

   To avoid installing the mpich2 package, use the second option:

```
pip install mpi4py
```

II Basic Usage

1. Create run_fido.py file

```
from fido_opt import FiDO
main = FiDO()
main.run()
```

2. Create input file param.in (See FiDO Keyword Documentation)

```
MODEL: Test

VARS:
        X:
        N: 5
        INIT: 1.0
        MIN: 0.0
        MAX: 2.0

ALGO: MADS

NUM_OBJ: 1

MAX_ITER: 20

MAX_EVAL: 100
```

3. Run in serial

```
python run_fido.py param.in
```

4. Run in parallel with 4 processors

```
export FIDO_USE_MPI=TRUE
mpiexec -np 4 python run_fido.py param.in
```

III Input file

The main input file follows the YAML format [1]. The YAML format is designed for both human readability and interaction with scripting languages. An online tutorial for learning the YAML syntax is available in [2]. With the PyYAML python package [4], the main input file is converted into a Python dictionary.

1. Syntax

The basic syntax for the main input file is keyword-value pair, separated by a colon. The value for a keyword can be a single entry. For example,

```
NUM_OBJ: 1
```

is a keyword-value pair meaning the value for keyword NUM_OBJ, number of objective functions, is 1.

The value for a keyword can also be a nested structure. Structure is shown through indentation. For example,

```
VARS:
        X1:
                MIN: 0
                MAX: 2
                INIT: 1
        X2:
                MIN: 2
                MAX: None
                INIT: 3
```

means there are two decision variables (keyword VARS), X1 and X2. Variable X1 has lower bound 0, upper bound 2 and initial guess 1. Variable X2 has lower bound 2, no upper bound and initial guess 3. Note that 'None' is used to indicated no lower/upper bounds.

The value of a keyword can also be a list. List items are denoted by a dash. The above example is equivalent to

```
VARS:
    X:
        N: 2
        MIN:
            - 0
            - 2
        MAX:
            - 2
            - None
        INIT:
            - 1
            - 3
```

which means the decision variables X has dimension N=2. The lower bound, upper bound and initial guess for X is thus a list of two items.

PyYAML converts keyword-value pair in the input file into nested Python dictionary. List is converted into Python list.

IV Keyword description

ALGO

Parent keywords	-
Value	single entry

This keyword specifies the optimization algorithm. The value for this keyword can be a single entry of the following values

- MADS (for 'Mesh Adaptive Direct Search'),
- PSO (for 'Particle Swarm Optimization'),
- PSO-MADS (for the hybrid algorithm of PSO and MADS).

DEFAULT: MADS
Example:

```
ALGO: MADS
```

CONSTRAINT

Parent keywords	-
Value	single entry

This keyword specifies the number and type of constraints. The value for this keyword is a nested structure with two sub-keywords [N] and [METHOD]. Sub-keyword [N] specifies the number of constraints. Sub-keyword [METHOD] specifies the type of each constraint.
Type of constraints (Please see NOMAD user manual for more details):

1. EB: extreme barrier method.
2. PB: progressive barrier method.

3. F: filter method.

In PSO-MADS algorithm, only PB and F are supported.

DEFAULT: 0
Example:

```
CONSTRAINT:
        N: 1
        METHOD: EB
```

```
CONSTRAINT:
        N: 2
        METHOD: PB PB
```

NUM_OBJ

Parent keywords	-
Value	single entry

This keyword specifies the number of objective functions. Currently, only single objective function is supported. DEFAULT: 1
Example:

```
NUM_OBJ: 1
```

MAX_EVAL

Parent keywords	-
Value	single entry

This keyword specifies the max number of function evaluations. The value for this keyword is a single integer.
DEFAULT: inf
Example:

```
MAX_EVAL: 200
```

MAX_ITER

Parent keywords	-
Value	single entry

This keyword specifies the max number of iterations. The value for this keyword is a single integer.
DEFAULT: inf
Example:

```
MAX_ITER: 100
```

MODEL

Parent keywords	-
Value	single entry

This keyword specifies the simulation model to use. The value for this keyword can be a single entry of the following values

- Nexus, built-in NEXUS model,
- R, built-in R model,
- BB, built-in black-box model,
- Test, built-in PROXY model.

DEFAULT: TEST
Example:

```
MODEL: Nexus
```

PSO_MADS

Parent keywords	-
Value	Sub-keywords

This keyword specifies the PSO-MADS parameters with three sub-keywords [PSO_SWARM_SIZE], [PSO_ITER_PER_SEARCH], [PSO_MAX_NUM_SEARCH].

1. PSO_SWARM_SIZE: number of particles.
2. PSO_ITER_PER_SEARCH: number of PSO iterations per PSO search.
3. PSO_MAX_NUM_SEARCH: maximum number of PSO searches.

Example:

```
PSO_MADS:
    PSO_SWARM_SIZE: 8
    PSO_ITER_PER_SEARCH: 3
    PSO_MAX_NUM_SEARCH: 10
```

Information for the PSO-MADS algorithm is available in [3].

V Source Code Documentation

Source code documentation is available in HTML format at

```
[FIDO]/docs/src-doc/_build/html/index.html
```

Comments in the source code follow the Google style python docstrings. The source code documentation is generated automatically using the Sphinx python package.
To update the source code documentation (after modifying the source code):

```
cd [FIDO]/docs/src-doc
sphinx-apidoc -o . ../../fido_opt
make html
```

Install required packages: Sphinx

```
pip install Sphinx
```

Read the Docs theme for Sphinx:

```
pip install sphinx_rtd_theme
```

The Sphinx napoleon extension for using Google style docstrings in Sphinx

```
pip install sphinxcontrib-napoleon
```

More information: Sphinx tutorial, Google style python docstrings, ,Read the Docs theme for Sphinx, The Sphinx napoleon extension.

Bibliography

[1] Ben-Kiki, O., Evans, C., and dt Net, I. (2009). *YAML ain't markup language*. http://www.yaml.org/start.html. Accessed: 2017-06-26.

[2] Brenecki, A. and SG, S. (2017). *Learn YAML in Y minutes*. https://learnxinyminutes.com/docs/yaml/. Accessed: 2017-06-26.

[3] Isebor, O. J., Echeverría Ciaurri, D., Durlofsky, L. J., et al. (2014). Generalized field-development optimization with derivative-free procedures. *SPE Journal*, 19(05):891–908.

[4] Simonov, K. (2016). Pyyaml. http://pyyaml.org/wiki/PyYAML. Accessed: 2017-06-26.