pyOptSparse

MDO Lab

Aug 18, 2022

TABLE OF CONTENTS

pyOptSparse is an object-oriented framework for formulating and solving nonlinear constrained optimization problems in an efficient, reusable, and portable manner. Some key features of pyOptSparse include:

- Object-oriented development, which maintains independence between the optimization problem formulation and its solution by different optimizers
- The use of sparse matrices throughout the code, to more efficiently handle large-scale optimization problems
- Parallel model execution under MPI, both for expensive analyses that must be done in parallel, and for parallel function evaluation when using certain gradient-free optimizers
- The optimization histories can be stored during the optimization process, and a partial history can also be used to hot-restart the optimization
- A post-processing GUI utility called OptView to analyze optimization results

pyOptSparse is a fork of pyOpt. However, it is not backwards compatible with pyOpt and thus optimization scripts will need to be modified to use pyOptSparse.

CHAPTER

ONE

GETTING STARTED

To get started, please see the *Installation Instructions* and the *Quickstart*.

1.1 Installation Instructions

1.1.1 Conda

Conda packages are available on `conda-forge` and can be installed via

```
$ conda install -c conda-forge pyoptsparse
```

This would install pyOptSparse with the built-in optimizers, as well as IPOPT. If you wish to use optimizers not packaged by `conda`, e.g. SNOPT, then you must build the package from source.

1.1.2 Building from source

Requirements

pyOptSparse has the following dependencies:

- Python 3.7 or 3.8, though other Python 3 versions will likely work
- C and Fortran compilers. We recommend `gcc` and `gfortran` which can be installed via the package manager for your operating system.

Please make sure these are installed and available for use. In order to use NSGA2, SWIG (v1.3+) is also required, which can be installed via the package manager. If those optimizers are not needed, then you do not need to install SWIG. Simply comment out the corresponding lines in `pyoptsparse/pyoptsparse/setup.py` so that they are not compiled. The corresponding lines in `pyoptsparse/pyoptsparse/__init__.py` must be commented out as well.

Python dependencies are automatically handled by `pip`, so they do not need to be installed separately. The only exception is `numpy`, which is required as part of the build process and therefore must be present before installing.

Note:

- In Linux, the python header files (`python-dev`) are also required.
- **We do not support operating systems other than Linux.** For macOS users, the conda package may work out of the box if you do not need any non-default optimizers. For Windows users, a conda package is on the way, if it's not already in the repos. This comes with the same disclaimer as the macOS conda package. Alternatively, follow the *conda build instructions* below as this will work on any platform.

Installation

The easiest and recommended way to install pyOptSparse is with `pip`. First clone the repository into a location which is not on the `$PYTHONPATH`, for example `$HOME/packages/`. Then in the root `pyoptsparse` folder type

```
$ pip install .
```

For those not using virtual environments, a user install may be needed

```
$ pip install . --user
```

If you plan to modify pyOptSparse, installing with the developer option, i.e. with `-e`, will save you from re-installing each time you modify the Python code.

Note: Some optimizers are proprietary, and their sources are not distributed with pyOptSparse. To use them, please follow the instructions on specific optimizer pages.

Specifying compilers

To specify a non-default compiler (e.g. something other than `/usr/bin/gcc`), meson recognizes certain special environment variables. For example, to specify the Intel compilers, simply run

```
$ FC=$(which ifort) CC=$(which icc) pip install .
```

1.1.3 Installing OptView

OptView and OptView-Dash have separate dependencies that must be installed. To install pyOptSparse including those dependencies, run

```
$ pip install .[optview]
```

1.1.4 Testing

pyOptSparse provides a set of unit and regression tests to verify the installation. To run these tests, first install `testflo` which is a testing framework developed by the OpenMDAO team:

```
$ pip install testflo
```

Then, in the project root directory, type:

```
$ testflo . -v
```

to run all tests.

1.1.5 Update or Uninstall

To update pyOptSparse, first delete the `meson_build` directory, then update the package using `git`. For stability, users are encouraged to stick to tagged releases. Install the package normally via `pip`.

To uninstall the package, type

```
$ pip uninstall pyoptsparse
```

1.1.6 Conda Build Instructions

The following instructions explain how to build and install pyOptSparse in a conda environment. This has the advantage that `conda` can be used to install all the necessary dependencies in an isolated and reproducible environment. Considering how finicky Windows can be with ABI compatibility among various compilers, this is the recommended approach. The guide will work on any platform, however.

The only build requirement for the build is a working `conda` installation as all compilers and dependencies are pulled from the `conda-forge` repos, with the exception of a Windows build, which requires Visual Studio 2017 C++ Build Tools.

First, we need to create the `conda` environment. An `environment.yml` file is provided in the `pyoptsparse` repo:

Linux/OSX

Windows

```
conda create -y -n pyos-build
conda activate pyos-build
conda config --env --add channels conda-forge
conda config --env --set channel_priority strict

conda env update -f .github/environment.yml
```

```
conda create -y -n pyos-build
conda activate pyos-build
conda config --env --add channels conda-forge
conda config --env --set channel_priority strict

conda env update -f .github\environment.yml
conda install libpgmath
```

Next, we need to tell the compiler where to find IPOPT:

Linux/OSX

Windows

```
export IPOPT_DIR="$CONDA_PREFIX"
```

```
set IPOPT_DIR=%CONDA_PREFIX%\Library
```

Finally, build the wheel and install it using pip:

Linux/OSX

Windows

```
# build wheel
python -m build -n -x .

# install wheel
pip install --no-deps --no-index --find-links dist pyoptsparse
```

```
# set specific compiler flags
set CC=cl
set FC=flang
```

(continues on next page)

(continued from previous page)

```
set CC_LD=link

# build wheel
python -m build -n -x .

# install wheel
pip install --no-deps --no-index --find-links dist pyoptsparse
```

1.2 Quickstart

The following shows how to get started with pyOptSparse by solving Schittkowski's TP37 constrained problem. First, we show the complete program listing and then go through each statement line by line:

```
# First party modules
from pyoptsparse import SLSQP, Optimization


# rst begin objfunc
def objfunc(xdict):
    x = xdict["xvars"]
    funcs = {}
    funcs["obj"] = -x[0] * x[1] * x[2]
    conval = [0] * 2
    conval[0] = x[0] + 2.0 * x[1] + 2.0 * x[2] - 72.0
    conval[1] = -x[0] - 2.0 * x[1] - 2.0 * x[2]
    funcs["con"] = conval
    fail = False

    return funcs, fail


# rst begin optProb
# Optimization Object
optProb = Optimization("TP037 Constraint Problem", objfunc)

# rst begin addVar
# Design Variables
optProb.addVarGroup("xvars", 3, "c", lower=[0, 0, 0], upper=[42, 42, 42], value=10)

# rst begin addCon
# Constraints
optProb.addConGroup("con", 2, lower=None, upper=0.0)

# rst begin addObj
# Objective
optProb.addObj("obj")

# rst begin print
# Check optimization problem
print(optProb)
```

(continues on next page)

(continued from previous page)

```
# rst begin OPT
# Optimizer
optOptions = {"IPRINT": -1}
opt = SLSQP(options=optOptions)

# rst begin solve
# Solve
sol = opt(optProb, sens="FD")

# rst begin check
# Check Solution
print(sol)
```

Start by importing the pyOptSparse package:

```
# First party modules
from pyoptsparse import SLSQP, Optimization
```

Next we define the objective function that takes in the design variable *dictionary* and returns a *dictionary* containing the constraints and objective, as well as a (boolean) flag indicating if the objective function evaluation was successful. For the TP37, the objective function is a simple analytic function:

```
def objfunc(xdict):
    x = xdict["xvars"]
    funcs = {}
    funcs["obj"] = -x[0] * x[1] * x[2]
    conval = [0] * 2
    conval[0] = x[0] + 2.0 * x[1] + 2.0 * x[2] - 72.0
    conval[1] = -x[0] - 2.0 * x[1] - 2.0 * x[2]
    funcs["con"] = conval
    fail = False

    return funcs, fail
```

Notes:

- The `xdict` variable is a dictionary whose keys are the names from each *addVar* and *addVarGroup* call. The line

  ```
  x = xdict["xvars"]
  ```

  retrieves an array of length 3 which are all the variables for this optimization.

- The line

  ```
  conval = [0] * 2
  ```

  creates a list of length 2, which stores the numerical values of the two constraints. The `funcs` dictionary return must contain keys that match the constraint names from *addCon* and *addConGroup* as well as the objectives from *addObj* calls. This is done in the following calls:

```
    funcs["obj"] = -x[0] * x[1] * x[2]
    funcs["con"] = conval
```

Now the optimization problem can be initialized:

```
# Optimization Object
optProb = Optimization("TP037 Constraint Problem", objfunc)
```

This creates an instance of the optimization class with a name and a reference to the objective function. To complete the setup of the optimization problem, the design variables and constraints need to be defined.

Design variables and constraints can be added either one-by-one or as a group. Adding variables by group is generally recommended for related variables:

```
# Design Variables
optProb.addVarGroup("xvars", 3, "c", lower=[0, 0, 0], upper=[42, 42, 42], value=10)
```

This calls adds a group of 3 variables with name `xvars`. The variable bounds (side constraints) are 0 for the lower bounds, and 42 for the upper bounds. The initial values for each variable is 10.0.

Now, we must add the constraints. Like design variables, these may be added individually or by group. It is recommended that related constraints are added by group where possible:

```
# Constraints
optProb.addConGroup("con", 2, lower=None, upper=0.0)
```

This call adds two variables with name `con`. There is no lower bound for the variables and the upper bound is 0.0.

We must also assign the the key value for the objective using the *addObj* call:

```
# Objective
optProb.addObj("obj")
```

The optimization problem can be printed to verify that it is set up correctly:

```
# Check optimization problem
print(optProb)
```

which produces the following table:

```
Optimization Problem -- TP037 Constraint Problem
================================================================================
   Objective Function: objfunc


   Objectives
      Index  Name            Value
         0  obj     0.000000E+00

   Variables (c - continuous, i - integer, d - discrete)
```

(continues on next page)

(continued from previous page)

```
      Index  Name      Type      Lower Bound            Value      Upper Bound      Status
         0  xvars_0      c     0.000000E+00     1.000000E+01     4.200000E+01
         1  xvars_1      c     0.000000E+00     1.000000E+01     4.200000E+01
         2  xvars_2      c     0.000000E+00     1.000000E+01     4.200000E+01

   Constraints (i - inequality, e - equality)
      Index  Name Type          Lower           Value           Upper    Status ␣
↪Lagrange Multiplier (N/A)
         0  con    i  -1.000000E+20    0.000000E+00    0.000000E+00         u    9.
↪00000E+100
         1  con    i  -1.000000E+20    0.000000E+00    0.000000E+00         u    9.
↪00000E+100
```

To solve an optimization problem with pyOptSparse an optimizer must be initialized. The initialization of one or more optimizers is independent of the initialization of the optimization problem. To initialize SLSQP, which is an open-source, sequential least squares programming algorithm that comes as part of the pyOptSparse package, use:

```
# Optimizer
optOptions = {"IPRINT": -1}
opt = SLSQP(options=optOptions)
```

This initializes an instance of SLSQP with the option IPRINT set to -1. All other options will be set to the default values, which can be found on the optimizer-specific pages. For example, the default options for SLSQP can be found in the page *SLSQP*.

Now TP37 can be solved using pyOptSparse's automatic finite difference for the gradients:

```
# Solve
sol = opt(optProb, sens="FD")
```

We can print the solution objection to view the result of the optimization:

```
# Check Solution
print(sol)
```

which produces the following output:

```
Optimization Problem -- TP037 Constraint Problem
================================================================================
   Objective Function: objfunc

   Solution:
--------------------------------------------------------------------------------
   Total Time:                    0.0062
      User Objective Time :       0.0001
      User Sensitivity Time :     0.0011
      Interface Time :            0.0046
      Opt Solver Time:            0.0003
   Calls to Objective Function :      22
   Calls to Sens Function :            8
```

(continues on next page)

(continued from previous page)

```
  Objectives
     Index  Name            Value
        0  obj    -3.456000E+03

  Variables (c - continuous, i - integer, d - discrete)
     Index  Name      Type      Lower Bound            Value      Upper Bound     Status
        0  xvars_0      c     0.000000E+00     2.399997E+01     4.200000E+01
        1  xvars_1      c     0.000000E+00     1.200001E+01     4.200000E+01
        2  xvars_2      c     0.000000E+00     1.200000E+01     4.200000E+01

  Constraints (i - inequality, e - equality)
     Index  Name Type          Lower           Value           Upper    Status ␣
↪Lagrange Multiplier (N/A)
        0  con    i  -1.000000E+20    7.564591E-07    0.000000E+00         u    9.
↪00000E+100
        1  con    i  -1.000000E+20   -7.200000E+01    0.000000E+00              9.
↪00000E+100
```

1.3 Guide

pyOptSparse is designed to solve general, constrained nonlinear optimization problems of the form:

$$\begin{aligned} \min \quad & f(x) \\ \text{with respect to} \quad & x \\ \text{such that} \quad & g_{j,\mathrm{L}} \leq g_j(x) \leq g_{j,\mathrm{U}}, \quad j = 1, ..., m \\ & x_{i,\mathrm{L}} \leq x_i \leq x_{i,\mathrm{U}}, \quad i = 1, ..., n \end{aligned}$$

where: x is the vector of n design variables, $f(x)$ is a nonlinear function, and $g(x)$ is a set of m nonlinear functions.

Equality constraints are specified using the same upper and lower bounds for the constraint. i.e., $g_{j,\mathrm{L}} = g_{j,\mathrm{U}}$. The ordering of the constraints is arbitrary; pyOptSparse reorders the problem automatically depending on the requirements of each individual optimizer.

The optimization class is created using the following call:

```
optProb = Optimization("name", objFun)
```

The general template of the objective function is as follows:

```
def obj_fun(xdict):
    funcs = {}
    funcs["obj_name"] = function(xdict)
    funcs["con_name"] = function(xdict)
    fail = False  # Or True if an analysis failed

    return funcs, fail
```

where:

- `funcs` is the dictionary of constraints and objective value(s)

- `fail` can be a Boolean or an int. False (or 0) for successful evaluation and True (or 1) for unsuccessful. Can also be 2 when using SNOPT and requesting a clean termination of the run.

If the Optimization problem is unconstrained, `funcs` will contain only the objective key(s).

1.3.1 Design Variables

The simplest way to add a single continuous variable with no bounds (side constraints) and initial value of 0.0 is to simply call *addVar*:

```
optProb.addVar("var_name")
```

This will result in a scalar variable included in the `x` dictionary call to `obj_fun` which can be accessed by doing

```
x["var_name"]
```

A more complex example will include lower bounds, upper bounds and a non-zero initial value:

```
optProb.addVar("var_name", lower=-10, upper=5, value=-2)
```

The `lower` or `upper` keywords may be specified as `None` to signify there is no bound on the variable.

Finally, an additional keyword argument `scale` can be specified which will perform an internal design variable scaling. The `scale` keyword will result in the following:

$$x_{\text{opt}} = x_{\text{user}} \times \text{scale}$$

The purpose of the scale factor is ensure that design variables of widely different magnitudes can be used in the same optimization. It is desirable to have the magnitude of all variables within an order of magnitude or two of each other.

The *addVarGroup* call is similar to *addVar* except that it adds a group of 1 or more variables. These variables are then returned as a numpy array within the x-dictionary. For example, to add 10 variables with no lower bound, and a scale factor of 0.1:

```
optProb.addVarGroup("con_group", 10, upper=2.5, scale=0.1)
```

1.3.2 Constraints

The simplest way to add a single constraint with no bounds (i.e., not a very useful constraint!) is to use the function *addCon*:

```
optProb.addCon("not_a_real_constraint")
```

To include bounds on the constraints, use the `lower` and `upper` keyword arguments. If `lower` and `upper` are the same, it will be treated as an equality constraint:

```
optProb.addCon("inequality_constraint", upper=10)
optProb.addCon("equality_constraint", lower=5, upper=5)
```

Like design variables, it is often necessary to scale constraints such that all constraint values are approximately the same order of magnitude. This can be specified using the `scale` keyword:

```
optProb.addCon("scaled_constraint", upper=10000, scale=1.0 / 10000)
```

Even if the `scale` keyword is given, the `lower` and `upper` bounds are given in their un-scaled form. Internally, pyOptSparse will use the scaling factor to produce the following constraint:

$$\text{con}_{\text{opt}} = \text{con}_{\text{user}} \times \text{scale}$$

In the example above, the constraint values are divided by 10000, which results in a upper bound (that the optimizer sees) of 1.0.

Constraints may also be flagged as linear using the `linear=True` keyword option. Some optimizers can perform special treatment on linear constraint, often ensuring that they are always satisfied exactly on every function call (SNOPT for example). Linear constraints also require the use of the `wrt` and `jac` keyword arguments. These are explained below.

One of the major goals of pyOptSparse is to enable the use of sparse constraint Jacobians, hence the *Sparse* in the name! Manually computing sparsity structure of the constraint Jacobian is tedious at best and become even more complicated as optimization scripts are modified by adding or deleting design variables and/or constraints. pyOptSparse is designed to greatly facilitate the assembly of sparse constraint Jacobians, alleviating the user of this burden. The idea is that instead of the user computing a dense matrix representing the constraint Jacobian, a "dictionary of keys" approach is used which allows incrementally specifying parts of the constraint Jacobian. Consider the optimization problem given below:

```
          varA (3)   varB (1)   varC (3)
         +---------------------------------+
conA (2) |          |     X    |     X    |
         ---------------------------------
conB (2) |     X    |          |     X    |
         ---------------------------------
conC (4) |     X    |     X    |     X    |
         ---------------------------------
conD (3) |          |          |     X    |
         +---------------------------------+
```

The `X`'s denote which parts of the Jacobian have non-zero values. pyOptSparse does not determine the sparsity structure of the Jacobian automatically, it must be specified by the user during calls to *addCon* and *addConGroup*. By way of example, the code that generates the hypothetical optimization problem is as follows:

```
optProb.addVarGroup("varA", 3)
optProb.addVarGroup("varB", 1)
optProb.addVarGroup("varC", 3)

optProb.addConGroup("conA", 2, upper=0.0, wrt=["varB", "varC"])
optProb.addConGroup("conB", 2, upper=0.0, wrt=["varC", "varA"])
optProb.addConGroup("conC", 4, upper=0.0)
optProb.addConGroup("conD", 3, upper=0.0, wrt=["varC"])
```

Note that the order of the `wrt` (which stands for with-respect-to) is not significant. Furthermore, if the `wrt` argument is omitted altogether, pyOptSparse assumes that the constraint is dense.

To examine the sparsity pattern, pyOptSparse can generate the ASCII table shown above. To do so, use the following call after adding all the design variables, objectives and constraints:

```
optProb.printSparsity()
```

Using the `wrt` keyword allows the user to determine the overall sparsity structure of the constraint Jacobian. However, we have currently assumed that each of the blocks with an `X` in is a dense sub-block. pyOptSparse allows each of the *sub-blocks* to itself be sparse. pyOptSparse requires this sparsity structure to be specified when the constraint is added. This information is supplied through the `jac` keyword argument. Lets say, that the `(conD, varC)` block of the Jacobian is actually a sparse and linear. By way of example, the call instead may be as follows:

```
jac = sparse.lil_matrix((3, 3))
jac[0, 0] = 1.0
jac[1, 1] = 4.0
jac[2, 2] = 5.0

optProb.addConGroup("conD", 3, upper=0.0, wrt=["varC"], linear=True, jac={"varC": jac})
```

We have created a linked list sparse matrix using `scipy.sparse`. Any SciPy sparse matrix format can be accepted. We have then provided this constraint Jacobian using the `jac` keyword argument. This argument is a dictionary, and the keys must match the design variable sets given in the `wrt` to keyword. Essentially what we have done is specified the which blocks of the constraint rows are non-zero, and provided the sparsity structure of ones that are sparse.

For linear constraints the values in `jac` are meaningful: they must be the actual linear constraint Jacobian values (which do not change). For non-linear constraints, only the sparsity structure (i.e. which entries are nonzero) is significant. The values themselves will be determined by a call to the `sens()` function.

Also note, that the `wrt` and `jac` keyword arguments are only supported when user-supplied sensitivity is used. If automatic gradients from pyOptSparse are used, the constraint Jacobian will necessarily be dense.

Note: Currently, only the optimizers SNOPT and IPOPT support sparse Jacobians.

1.3.3 Objectives

Each optimization will require at least one objective to be added. This is accomplished using a the call to *addObj*:

```
optProb.addObj("obj_name")
```

What this does is tell pyOptSparse that the key `obj_name` in the function returns will be taken as the objective. For optimizers that can do multi-objective optimization (e.g. NSGA2), multiple objectives can be added. Optimizers that can only handle one objective enforce that only a single objective is added to the optimization description.

1.3.4 Specifying Derivatives

Approximating Derivatives

pyOptSparse can automatically compute derivatives of the objective and constraint functions using finite differences or the complex-step method. This is done by simply passing a string to the `sens=` argument when calling an optimizer. See the possible values *here*. In the simplest case, using `sens="FD"` will be enough to run an optimization using forward differences with a default step size.

Analytic Derivatives

If analytic derivatives are available, users can compute them within a user-defined function. This function accepts as inputs a dictionary containing design variable values as well as another dictionary containing objective and constraint values. It returns a nested dictionary containing the gradients of the objective and constraint values with respect to those design variables at the current design point. Specifically, the first-layer keys should be associated with objective and constraint names while the second-layer keys correspond to design variables. The dictionary values are the computed analytic derivatives, either in the form of lists or NumPy arrays with the expected shape. Since pyOptSparse uses string indexing, users need to make sure the keys in the returned dictionary are consistent with the names of design variables, constraints and objectives which were first added to the optimization problem.

Tip:

1. Only the non-zero sub-blocks of the Jacobian need to be defined in the dictionary, and pyOptSparse will assume the rest to be zero.
2. Derivatives of the linear constraints do not need to be given here, since they are constant and should have already been specified via the `jac=` keyword argument when adding the constraint.

For example, if the optimization problem has one objective `obj`, two constraints `con`, and three design variables `xvars`, the returned sensitivity dictionary (with placeholder values) should have the following structure:

```
{"obj": {"xvars": [1, 2, 3]}, "con": {"xvars": [[4, 5, 6], [7, 8, 9]]}}
```

Once this function is constructed, users can pass its function handle to the optimizer when it's called via:

```
sol = opt(optProb, sens=sens, ...)
```

1.3.5 Optimizer Instantiation

There are two ways to instantiate the optimizer object. The first, and most explicit approach is to directly import the optimizer class, for example via

```
from pyoptsparse import SLSQP

opt = SLSQP(...)
```

However, in order to easily switch between different optimizers without having to import each class, a convenience function called *OPT* is provided. It accepts a string argument in addition to the usual options, and instantiates the optimizer object based on the string:

```
from pyoptsparse import OPT

opt = OPT("SLSQP", ...)
```

Note that the name of the optimizer is case-insensitive, so `slsqp` can also be used. This makes it easy to for example choose the optimizer from the command-line, or more generally select the optimizer using strings without preemptively importing all classes.

1.4 Advanced Features

1.4.1 MPI handling

pyOptSparse can optionally run in parallel if a suitable `mpi4py` installation exists. This will be automatically detected and imported at run-time.

If you only want to run in parallel, you can force pyOptSparse to do so by setting the environment variable `PYOPTSPARSE_REQUIRE_MPI` to any one of these values: `['always', '1', 'true', 'yes']` If a suitable `mpi4py` is not available, an exception will be raised and the run terminated.

If you explicitly do not wish to use `mpi4py`, set the environment variable `PYOPTSPARSE_REQUIRE_MPI` to anything other than those values. This can come in handy, for example, if your `MPI` installation is not functioning properly, but you still need to run serial code.

1.4.2 Storing Optimization History

pyOptSparse includes a *History* class that stores all the relevant optimization information an SQL database. This database is updated at every optimization iteration, and can be accessed via both the API described in the linked section, and via *OptView*. By default, the history file is NOT written. To turn the history recording on, use the `storeHistory` attribute when invoking the optimization run, e.g.:

```
sol = opt(optProb, sens=sens, storeHistory="<your-history-file-name>.hst", ...)
```

1.4.3 Hot start

Hot start refers to the way optimizations are initialized. Such start-up procedure is named in contrast to the regular optimization initialization, or "cold start", in which case pyOptSparse simply initializes an optimization job from scratch, using initial design variables set by the user. There are several situations in which such cold starts can be avoided by leveraging on the information from a previous optimization, with the aim to reduce the overall computational time. Suppose you run an optimization that was accidentally terminated prematurely by, for example, an excessively-low iteration limit. If you restarted the optimization using the DVs from the last iteration (but losing all the accumulated history), you will start from a better initial point but will likely incur in a performance penalty. The overall optimization, now split between two jobs, will end up taking far more steps than it would've taken if the original optimization was allowed to progress. This is due to the fact that some of the optimizers wrapped in pyOptSparse need a few initial iterations to build a (local) approximation of the design space. Because of this lack of information, there is a start-up cost to a cold started optimization, where the optimizer has to spend time rebuilding the information. In a sense, for many optimizers the next iterate does not depend only on the current iterate, but effectively all the previous iterates as well!

Hot starting optimizations is a way to address this issue. pyOptSparse has the ability to read in the previous optimization history file, and *replay* the entire history starting at the original design variables, feeding `funcs` to the optimizer each time. For a deterministic optimizer, if all solver settings remain the same, then it should request the same sequence of iterates (up to machine precision), and if those have been previously evaluated, they are read from the history file and passed to the optimizer. If at any point the requested design variables diverge from the history, then we will actually perform a function evaluation, and all subsequent points will be newly evaluated. For this process to work, the following must be true:

- The optimizer is deterministic. Note that a stochastic optimizer can still be made deterministic even if it uses random numbers, as long as the random seed is fixed. If the optimizer performs a random gradient check (e.g. SNOPT), it's best to disable these just in case.
- Optimizer settings that affect the path of the optimization must remain the same. It is perfectly fine to change for example settings related to print outs, but not those affecting the line search.

To use the hot start feature, simply call the optimizer with the option `hotStart = <hot start file>`. See the API documentation for each optimizer for more information. Because the hot start process will store all the previous "restarted" iterations in the new history file, it's possible to restart as many times as you like, each time using the previous history file.

1.4.4 Time limit (for SNOPT only)

The *Optimizer* class in pyOptSparse has an attribute used to set the maximum allowable wall time for optimizations using SNOPT. The code will exit gracefully when such time limit is reached. This feature is particularly useful when running a time-constrained job, as in the case of most HPC systems. To enable this feature, use the `timeLimit` option when invoking the optimizer, as shown below:

```
sol = opt(optProb, sens=sens, timeLimit=24 * 3600, ...)
```

Note that the attribute takes the maximum wall time *in seconds* as an integer number.

Note: pyOptSparse will verify that the computational time is not exceeded before proceeding to the next iteration. It will NOT interrupt an ongoing function or sensitivity evaluation. If your function evaluations are expensive, you should be more conservative when setting the `timeLimit` option for it to be effective.

1.5 Post-processing

There are three post-processing utilities that are provided with pyOptSparse.

- OptView is a GUI designed to quickly and interactively visualize optimization histories
- OptView-Dash is a Dash implementation of OptView
- `History` is a Python class that can be used to read in the history file, and provide API for programmatically extracting data.

1.5.1 OptView

Requirements

`OptView` has the following dependency tree:

```
matplotlib (OptView)
  \-backends
  | \-backend_tkagg
  |   \-FigureCanvasTkAgg (OptView)
  |   \-NavigationToolbar2TkAgg (OptView)
  \-pyplot (OptView)
mpl_toolkits
  \-axes_grid1
  | \-host_subplot (OptView)
  \-axisartist (OptView)
numpy (OptView)
```

For installation instructions, see *Installing OptView*. Although not necessary for most usage, the `dill` package is needed if you wish to save an editable version of the graph produced in `OptView`. `dill` can be installed via `pip` in a terminal using

```
$ pip install dill
```

`view_saved_figure.py` can be used to reformat and view the saved figure.

Usage

`OptView` can be run via terminal from any directory as

```
$ optview histFile
```

Here, `histFile` is the name of the history file to be examined (default is `opt_hist.hst`).

Additionally, you can open multiple history files in the same `OptView` instance by calling them via the command line

```
$ optview histFile1 histFile2 histFile3
```

Each file's contents will be loaded into `OptView` with a flag appended to the end of each variable or function name corresponding to the history file. The first one listed will have '_A' added to the name, the second will have '_B' added, etc. There is currently no limit to the number of history files than can be loaded.

Optionally, if you want to save the generated figures, there is an optional argument:

```
$ optview histFile --output ~/my_figures
```

`outputDirectory` is the name of the desired output directory for saved images. By default, the figure is saved to the directory where you invoked `optview`.

Features

`OptView` has many options and features, including:

- plotting multiple variables on a single plot
- producing stacked plots
- live searchable variable names
- hovering plot labels
- saving the figure to an image or pickling it for later formatting
- refreshing the optimization history on the fly

Although some of these are self-explanatory, the layout and usage of `OptView` will be explained below.

GUI Layout

The window is divided into two sections. The top is the canvas where the figure and graphs will be produced, while the bottom grayed section contains user-selectable options. Here, we will focus on the user options.

The selectable variables are contained on the left hand side of the options panel in scrollable listboxes. You can select multiple items from the listboxes using the normal selection operators such as control and shift. If a selected variable is an array, a third listbox should appear on the right hand side of the options panel, allowing you to select specific sub-variables within the single array variable.

There are three main options when selecting how to produce the graph(s):

- Shared axes - all selected variables are plotted on a single pair of axes
- Multiple axes - each selected variable gets its own y-axis while all selected data shares an x-axis
- Stacked plots - each variable gets its own individual plot and the set is stacked vertically

Most checkbox options should play well with any of these three main options, though there are known issues with using the 'multiple axes' option and delta values or for displaying arrays.

There are seven checkbox options:

- Absolute delta values - displays the absolute difference between one iteration's value and the previous
- Log scale - sets the y-axis as a log scale
- Min/max for arrays - only shows the minimum and maximum value of a variable for each iteration
- Show all for arrays - plots all variables within an array
- Show legend - reveals the legend for the plotted data
- Show bounds - shows the variable bounds as dashed lines
- Show 'major' iterations - a filter to remove the line search iterations from the plotting results; especially useful for SNOPT output

Additionally, four buttons allow control of the plot:

- Refresh history - reloads the history file; used if checking on an optimization run on the fly
- Save all figures - saves .png versions of a basic plot for each variable in the history file
- Save figure - saves a .png and .pickle version of the current plot (the .pickle version can be reformatted afterwards)
- Quit - exits the program

Lastly, there are some miscellaneous features:

- A search box to cull the selectable variables
- A font size slider to control the text size on the plot
- Hoverable tooltips when the cursor is on a plot line
- A variable called *actual_iteration_number* that gives a translation between history file iteration number and run file iteration number. This is especially useful for debugging specific steps of an optimization or comparing values across different histories.

1.5.2 OptView-Dash

This is a Dash implementation of OptView, and has many of the same features offered by OptView. For installation instructions, see *Installing OptView*. To run, use this command:

```
$ optview_dash <filename>
```

Similar to OptView, you can invoke it with multiple history files. To view the dash app, you will have to manually open the server in your browser that is listed in the terminal after running the above command.

Auto-refresh: This follows the same functionality as OptView, allowing you to see the changes of an optimization as it is running.

- If you toggle this checklist button, it will cause the program to default update every 10 seconds, however you may modify this refresh rate using the input box underneath
- Make sure to toggle off this button when you are done or the optimization is complete so it does not add lag.
- This feature also works with multiple history files/optimizations running!

1.5.3 Directly Accessing the History Object

The history file generated by pyOptSparse is just a SqliteDict object. To extract the stored information in Python, first initialize a History object:

```
hist = History("path/to/opt_hist.hst", flag="r")
```

From here, various information can be extracted, using the various `get_` methods. To extract iteration history, use the function `getValues()`. See the page *History* for a full description of the history file structure and the API.

1.6 Major changes compared to pyOpt

The following list summarizes some of the changes/improvements made to pyOpt to get pyOptSparse:

- Elimination of $\mathcal{O}(n^2)$ scaling behaviour when adding large numbers of design variables (>10,000)
- Proper handling of all optimizers when run in parallel environment (only a single optimization instance is run, not one instance per processor)
- More flexible return specification of constraints
- More flexible return specification of constraint *gradients*
- Complete elimination of gradient indexing errors using dictionary-based returns
- Substantial improvement of optimization script robustness through indexing elimination
- Automatic assembly of sparse Jacobians with both dense and sparse sub-blocks
- Automatic conversion of sparse Jacobian to dense for optimizers that cannot handle sparse constraints
- User specification of design variable scaling
- User specification of constraint scaling
- Design variable returns in dictionary format only
- Specification of linear constraints, dense or sparse
- Sparse non-linear Jacobians
- New history file format. Uses SQLite dictionaries.
- Fixed hot start bug where first call to user functions is a gradient. It is now guaranteed, that the first call is to the function evaluation, not the gradient.
- Various bug fixes in SNOPT
- Constraints can be in *any* order, independent of what the individual optimizer requires
- Python 3.x compatibility

1.7 How to Contribute to pyOptSparse

pyOptSparse is an open-source tool, thus we welcome users to submit additions or fixes to the code to make it better for everybody.

1.7.1 Issues

If you have an issue with pyOptSparse, a bug to report, or a feature to request, submit an issue on the GitHub repository. This lets other users know about the issue. If you are comfortable fixing the issue, please do so and submit a pull request.

1.7.2 Coding style

We use black for formatting. Please install it following its documentation, and run it with the command line argument `-l 120`, for example at the project root directory:

```
$ black . -l 120
```

This will automatically format all Python files.

We use flake8 for linting. Please install it following its instructions, and run it at the project root with:

```
$ flake8 .
```

The configuration file we use for flake8 is a combination of this file and the one at the root of this repository. If there are any PEP-8 violations, `flake8` will print out the nature of the violation. We run continuous integration with these tools on all pull requests submitted. For an easier workflow, we recommend integrating these tools with your code editor.

> **Warning:** For a PR to be accepted, it must pass all GitHub checks, which include both formatting checks with `black` and syntax checks with `flake8`.

1.7.3 Documentation

When you add or modify code, make sure to provide relevant documentation that explains the new code. This should be done in code via docstrings and comments, but also in the Sphinx documentation as well if you add a new feature or capability. Look at the `.rst` files in the `doc` section of the repo.

To build documentation locally, go to the `doc` folder and type `make html`. Building the documentation requires `sphinx` and `numpydoc`, as well as the Sphinx RTD theme. To install these dependencies, type

```
$ pip install sphinx numpydoc sphinx-rtd-theme
```

1.7.4 Testing

When you add code or functionality, add tests that cover the new or modified code. These may be units tests for individual components or regression tests for entire models that use the new functionality. All the existing tests can be found under the `test` folder.

1.7.5 Pull requests

Finally, after adding or modifying code, and making sure the steps above are followed, submit a pull request via the GitHub interface. This will automatically go through all of the tests in the repo to make sure everything is functioning properly. The main developers of pyOptSparse will then merge in the request or provide feedback on how to improve the contribution.

1.8 pyOptSparse in published works

pyOptSparse has been used extensively in the field of engineering design optimization. The following is a non-exhaustive list of works that have used pyOptSparse. The citations are organized by the optimization framework for which pyOptSparse is used within.

1.8.1 MACH-Aero

MACH-Aero is an open-source aerodynamic shape optimization framework developed by the MDO Lab at the University of Michigan. Since the vast majority of publications from the MDO Lab uses the MACH framework (and as a result pyOptSparse), only a few select publications from the lab are listed below. The rest are works in conjunction with collaborators.

- Nathalie Bartoli, Mostafa Meliani, Joseph Morlier, Thierry Lefebvre, Mohamed-Amine Bouhlel, and Joaquim R. R. A. Martins. Multi-fidelity efficient global optimization: methodology and application to airfoil shape design. In *AIAA Aviation Forum*. June 2019. doi:10.2514/6.2019-3236.
- Nicolas P. Bons, Charles A. Mader, Joaquim R. R. A. Martins, Ana P. C. Cuco, and Felipe I. K. Odaguil. High-fidelity aerodynamic shape optimization of a full configuration regional jet. In *2018 AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*. Kissimmee, FL, January 2018. doi:10.2514/6.2018-0106.
- Nicolas P. Bons, Joaquim R. R. A. Martins, Charles A. Mader, Matthew McMullen, and Michelle Suen. High-fidelity aerostructural optimization studies of the Aerion AS2 supersonic business jet. In *Proceedings of the AIAA Aviation Forum*. June 2020. doi:10.2514/6.2020-3182.
- Benjamin J. Brelje, Joshua Anibal, Anil Yildirim, Charles A. Mader, and Joaquim R. R. A. Martins. Flexible formulation of spatial integration constraints in aerodynamic shape optimization. *AIAA Journal*, 58(6):2571–2580, June 2020. doi:10.2514/1.J058366.
- Timothy R. Brooks, Gaetan K. W. Kenway, and Joaquim R. R. A. Martins. Benchmark aerostructural models for the study of transonic aircraft wings. *AIAA Journal*, 56(7):2840–2855, July 2018. doi:10.2514/1.J056603.
- Timothy R. Brooks, Joaquim R. R. A. Martins, and Graeme J. Kennedy. High-fidelity aerostructural optimization of tow-steered composite wings. *Journal of Fluids and Structures*, 88:122–147, July 2019. doi:10.1016/j.jfluidstructs.2019.04.005.
- Timothy R. Brooks, Joaquim R. R. A. Martins, and Graeme J. Kennedy. Aerostructural trade-offs for tow-steered composite wings. *Journal of Aircraft*, 2020. doi:10.2514/1.C035699.
- David A. Burdette and Joaquim R. R. A. Martins. Impact of morphing trailing edge on mission performance for the Common Research Model. *Journal of Aircraft*, 56(1):369–384, January 2019. doi:10.2514/1.C034967.
- Gustavo L. O. Halila, Joaquim R. R. A. Martins, and Krzysztof J. Fidkowski. Adjoint-based aerodynamic shape optimization including transition to turbulence effects. *Aerospace Science and Technology*, pages 1–15, December 2020. doi:10.1016/j.ast.2020.106243.
- Ping He and and Joaquim R. R. A. Martins. A hybrid time-spectral approach for aerodynamic shape optimization with unsteady flow. In *Proceedings of the AIAA SciTech Forum*. January 2021. doi:10.2514/6.2021-0278.

- Ping He, Grzegorz Filip, Joaquim R. R. A. Martins, and Kevin J. Maki. Design optimization for self-propulsion of a bulk carrier hull using a discrete adjoint method. *Computers & Fluids*, 192:104259, October 2019. doi:10.1016/j.compfluid.2019.104259.
- Ping He, Charles A. Mader, Joaquim R. R. A. Martins, and Kevin J. Maki. DAFoam: an open-source adjoint framework for multidisciplinary design optimization with OpenFOAM. *AIAA Journal*, March 2020. doi:10.2514/1.J058853.
- Xiaolong He, Jichao Li, Charles A. Mader, Anil Yildirim, and Joaquim R. R. A. Martins. Robust aerodynamic shape optimization—from a circle to an airfoil. *Aerospace Science and Technology*, 87:48–61, April 2019. doi:10.1016/j.ast.2019.01.051.
- Davide Ivaldi, Ney R. Secco, Song Chen, John T. Hwang, and Joaquim R. R. A. Martins. Aerodynamic shape optimization of a truss-braced-wing aircraft. In *Proceedings of the 16th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*. Dallas, TX, June 2015. doi:10.2514/6.2015-3436.
- Jichao Li, Mohamed Amine Bouhlel, and Joaquim R. R. A. Martins. Data-based approach for fast airfoil analysis and optimization. *AIAA Journal*, 57(2):581–596, February 2019. doi:10.2514/1.J057129.
- Jichao Li, Sicheng He, and Joaquim R. R. A. Martins. Data-driven constraint approach to ensure low-speed performance in transonic aerodynamic shape optimization. *Aerospace Science and Technology*, 92:536–550, September 2019. doi:10.1016/j.ast.2019.06.008.
- Jichao Li, Mengqi Zhang, Joaquim R. R. A. Martins, and Chang Shu. Efficient aerodynamic shape optimization with deep-learning-based filtering. *AIAA Journal*, 58(10):4243–4259, October 2020. doi:10.2514/1.J059254.
- Zhoujie Lyu, Gaetan K. W. Kenway, and Joaquim R. R. A. Martins. Aerodynamic shape optimization investigations of the Common Research Model wing benchmark. *AIAA Journal*, 53(4):968–985, April 2015. doi:10.2514/1.J053318.
- Mads H. Aa. Madsen, Frederik Zahle, Niels N. Sørensen, and Joaquim R. R. A. Martins. Multipoint high-fidelity CFD-based aerodynamic shape optimization of a 10 MW wind turbine. *Wind Energy Science*, 4:163–192, April 2019. doi:10.5194/wes-4-163-2019.
- Ney R. Secco and Joaquim R. R. A. Martins. RANS-based aerodynamic shape optimization of a strut-braced wing with overset meshes. *Journal of Aircraft*, 56(1):217–227, January 2019. doi:10.2514/1.C034934.
- Yayun Shi, Charles A. Mader, Sicheng He, Gustavo L. O. Halila, and Joaquim R. R. A. Martins. Natural laminar-flow airfoil optimization design using a discrete adjoint approach. *AIAA Journal*, 58(11):4702–4722, November 2020. doi:10.2514/1.J058944.

1.8.2 OpenMDAO

OpenMDAO is a popular multidisciplinary design and optimization framework developed by NASA Glenn Research Center, and support pyOptSparse as one of the possible optimization drivers.

- Nathalie Bartoli, Thierry Lefebvre, Sylvain Dubreuil, Romain Olivanti, Nicolas Bons, Joaquim R. R. A. Martins, Mohamed Amine Bouhlel, and Joseph Morlier. An adaptive optimization strategy based on mixture of experts for wing aerodynamic design optimization. In *Proceedings of the 18th AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*. Denver, CO, June 2017. doi:10.2514/6.2017-4433.
- Alp Dener, Pengfei Meng, Jason E Hicken, Graeme Kennedy, John Hwang, and Justin S Gray. Kona: a parallel optimization library for engineering-design problems. In *57th AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, 1422. 2016. doi:10.2514/6.2016-1422.
- Justin S. Gray and Joaquim R. R. A. Martins. Coupled aeropropulsive design optimization of a boundary-layer ingestion propulsor. *The Aeronautical Journal*, 123(1259):121–137, January 2019. doi:10.1017/aer.2018.120.

- Tae H Ha, Keunseok Lee, and John T Hwang. Large-scale multidisciplinary optimization under uncertainty for electric vertical takeoff and landing aircraft. In *AIAA SciTech 2020 Forum*, 0904. 2020. doi:10.2514/6.2020-0904.
- John M. Hegseth, Erin E. Bachynski, and Joaquim R. R. A. Martins. Integrated design optimization of spar floating wind turbines. *Marine Structures*, 72:102771, July 2020. (Moan–Faltinsen Best Paper Award). doi:10.1016/j.marstruc.2020.102771.
- John Marius Hegseth, Erin E. Bachynski, and Joaquim R. R. A. Martins. Design optimization of spar floating wind turbines considering different control strategies. *Journal of Physics: Conference Series*, 1669:012010, October 2020. doi:10.1088/1742-6596/1669/1/012010.
- Austin J Herrema, Josef Kiendl, and Ming-Chen Hsu. A framework for isogeometric-analysis-based optimization of wind turbine blade structures. *Wind Energy*, 22(2):153–170, 2019. doi:10.1002/we.2276.
- Daniel Ingraham, Justin S Gray, and Leonard V Lopes. Gradient-based propeller optimization with acoustic constraints. In *AIAA SciTech 2019 Forum*, 1219. 2019. doi:10.2514/6.2019-1219.
- John Jasa, Benjamin Brelje, Justin Gray, Charles A. Mader, and Joaquim R. R. A. Martins. Large-scale path-dependent optimization of supersonic aircraft. *Aerospace*, October 2020. doi:10.3390/aerospace7100152.
- Graeme Kennedy and Yicong Fu. Topology optimization benchmark problems for assessing the performance of optimization algorithms. In *AIAA SciTech 2021 Forum*, 1357. 2021. doi:10.2514/6.2021-1357.
- Michael K. McWilliam, Frederik Zahle, Antariksh Dicholkar, David Verelst, and Taeseong Kim. Optimal aeroelastic design of a rotor with bend-twist coupling. *Journal of Physics: Conference Series*, 1037:042009, jun 2018. doi:10.1088/1742-6596/1037/4/042009.
- Rajaram Attukur Nandagopal and Srikanth Narasimalu. Multi-objective optimization of hydrofoil geometry used in horizontal axis tidal turbine blade designed for operation in tropical conditions of South East Asia. *Renewable Energy*, 146:166–180, 2020. doi:10.1016/j.renene.2019.05.111.
- A. S. Padrón, J. Thomas, A. P. J. Stanley, J. J. Alonso, and A. Ning. Polynomial chaos to efficiently compute the annual energy production in wind farm layout optimization. *Wind Energy Science*, 4(2):211–231, 2019. doi:10.5194/wes-4-211-2019.
- Christian Pavese, Carlo Tibaldi, Frederik Zahle, and Taeseong Kim. Aeroelastic multidisciplinary design optimization of a swept wind turbine blade. *Wind Energy*, 20(12):1941–1953, 2017. doi:10.1002/we.2131.
- Ignas Satkauskas, Evan Gaertner, Pietro Bortolotti, Garrett Barter, and Peter A Graf. Wind turbine rotor design optimization using importance sampling. In *AIAA SciTech 2020 Forum*, 1953. 2020. doi:10.2514/6.2020-1953.
- Jared J. Thomas and Andrew Ning. A method for reducing multi-modality in the wind farm layout optimization problem. In *Journal of Physics: Conference Series*, volume 1037. Milano, Italy, June 2018. The Science of Making Torque from Wind. doi:10.1088/1742-6596/1037/4/042012.
- Frederik Zahle, Carlo Tibaldi, Christian Pavese, Michael K. McWilliam, Jose P. A. A. Blasques, and Morten H. Hansen. Design of an aeroelastically tailored 10 MW wind turbine rotor. *Journal of Physics: Conference Series*, 753(6):062008, 2016.

OpenAeroStruct

OpenAeroStruct is a low-fidelity aerostructural optimization framework implemented in OpenMDAO, and as such can leverage the optimization capabilities of pyOptSparse.

- Timothy R Brooks and Benjamin D Smith. Aerostructural design optimization of the D8 aircraft using active aeroelastic tailoring. In *AIAA Scitech 2020 Forum*, 1967. 2020. doi:10.2514/6.2020-1967.
- Shamsheer S. Chauhan and Joaquim R. R. A. Martins. Low-fidelity aerostructural optimization of aircraft wings with a simplified wingbox model using OpenAeroStruct. In *Proceedings of the 6th International Conference on Engineering Optimization, EngOpt 2018*, 418–431. Lisbon, Portugal, September 2018. Springer. doi:10.1007/978-3-319-97773-7_38.
- John P. Jasa, Shamsheer S. Chauhan, Justin S. Gray, and Joaquim R. R. A. Martins. How certain physical considerations impact aerostructural wing optimization. In *AIAA/ISSMO Multidisciplinary Analysis and Optimization Conference*. Dallas, TX, June 2019. doi:10.2514/6.2019-3242.
- John P. Jasa, John T. Hwang, and Joaquim R. R. A. Martins. Open-source coupled aerostructural optimization using Python. *Structural and Multidisciplinary Optimization*, 57(4):1815–1827, April 2018. doi:10.1007/s00158-018-1912-8.

1.8.3 SUAVE

SUAVE is a conceptual-level aircraft design framework developed at Stanford University that includes multiple analysis fidelities, and provides a pyOptSparse interface.

- Stanislav Karpuk and Ali Elham. Conceptual design trade study for an energy-efficient mid-range aircraft with novel technologies. In *AIAA Scitech 2021 Forum*, 0013. 2021. doi:10.2514/6.2021-0013.
- Michael Kruger and Alejandra Uranga. The feasibility of electric propulsion for commuter aircraft. In *AIAA Scitech 2020 Forum*, 1499. 2020. doi:10.2514/6.2020-1499.

1.8.4 Other

The works which did not use any of the aforementioned frameworks are listed here.

- N. Bartoli, T. Lefebvre, S. Dubreuil, R. Olivanti, R. Priem, N. Bons, Joaquim R. R. A. Martins, and J. Morlier. Adaptive modeling strategy for constrained global optimization with application to aerodynamic wing design. *Aerospace Science and Technology*, 90:85–102, July 2019. doi:10.1016/j.ast.2019.03.041.
- Julien de Mûelenaere and Juan J Alonso. High-fidelity trajectory based MDO for the conceptual design of an air-launched spaceplane. In *2018 AIAA/ASCE/AHS/ASC Structures, Structural Dynamics, and Materials Conference*, 0416. 2018. doi:10.2514/6.2018-0416.
- John T Hwang, Arnav V Jain, and Tae H Ha. Large-scale multidisciplinary design optimization—review and recommendations. In *AIAA Aviation 2019 Forum*, 3106. 2019. doi:10.2514/6.2019-3106.
- Graeme J. Kennedy and Ting Wei Chin. A sequential convex optimization method for multimaterial compliance design problems. *Computers & Structures*, 212:110 –124, 2019. doi:10.1016/j.compstruc.2018.10.007.
- Gaetan K. W. Kenway and Joaquim R. R. A. Martins. Multipoint high-fidelity aerostructural optimization of a transport aircraft configuration. *Journal of Aircraft*, 51(1):144–160, January 2014. doi:10.2514/1.C032150.
- Jichao Li and Jinsheng Cai. Massively multipoint aerodynamic shape design via surrogate-assisted gradient-based optimization. *AIAA Journal*, 58(5):1949–1963, 2020. doi:10.2514/1.J058491.
- Jichao Li, Jinsheng Cai, and Kun Qu. Drag reduction of transonic wings with surrogate-based optimization. In *Asia-Pacific International Symposium on Aerospace Technology*, 1065–1080. Springer, 2018. doi:10.1007/978-981-13-3305-7_85.

- Dev Rajnarayan, Andrew Ning, and Judd A. Mehr. Universal airfoil parametrization using b-splines. In *Proceedings of the AIAA Aviation Forum*. 2018. doi:10.2514/6.2018-3949.
- Smruti Sahoo, Xin Zhao, and Konstantinos Kyprianidis. A review of concepts, benefits, and challenges for future electrical propulsion-based aircraft. *Aerospace*, 7(4):44, 2020. doi:10.3390/aerospace7040044.

1.9 Citation

If you use pyOptSparse, please cite the following paper:

N. Wu, G. Kenway, C. A. Mader, J. Jasa, and J. R. R. A. Martins. pyOptSparse: A Python framework for large-scale constrained nonlinear optimization of sparse systems. Journal of Open Source Software, 5(54), 2564, October 2020. https://doi.org/10.21105/joss.02564

The paper is available online from the Journal of Open Source Software here. To cite this paper, you can use the following BibTeX entry:

```
@article{Wu2020,
    doi = {10.21105/joss.02564},
    year = {2020},
    publisher = {The Open Journal},
    volume = {5},
    number = {54},
    pages = {2564},
    author = {Neil Wu and Gaetan Kenway and Charles A. Mader and John Jasa and Joaquim R.
↪ R. A. Martins},
    title = {pyOptSparse: A Python framework for large-scale constrained nonlinear␣
↪optimization of sparse systems},
    journal = {Journal of Open Source Software}
}
```

1.10 License

```
                    GNU LESSER GENERAL PUBLIC LICENSE
                        Version 3, 29 June 2007

 Copyright (C) 2007 Free Software Foundation, Inc. <https://fsf.org/>
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.


  This version of the GNU Lesser General Public License incorporates
the terms and conditions of version 3 of the GNU General Public
License, supplemented by the additional permissions listed below.

  0. Additional Definitions.

  As used herein, "this License" refers to version 3 of the GNU Lesser
General Public License, and the "GNU GPL" refers to version 3 of the GNU
General Public License.
```

(continues on next page)

(continued from previous page)

```
  "The Library" refers to a covered work governed by this License,
other than an Application or a Combined Work as defined below.

  An "Application" is any work that makes use of an interface provided
by the Library, but which is not otherwise based on the Library.
Defining a subclass of a class defined by the Library is deemed a mode
of using an interface provided by the Library.

  A "Combined Work" is a work produced by combining or linking an
Application with the Library.  The particular version of the Library
with which the Combined Work was made is also called the "Linked
Version".

  The "Minimal Corresponding Source" for a Combined Work means the
Corresponding Source for the Combined Work, excluding any source code
for portions of the Combined Work that, considered in isolation, are
based on the Application, and not on the Linked Version.

  The "Corresponding Application Code" for a Combined Work means the
object code and/or source code for the Application, including any data
and utility programs needed for reproducing the Combined Work from the
Application, but excluding the System Libraries of the Combined Work.

  1. Exception to Section 3 of the GNU GPL.

  You may convey a covered work under sections 3 and 4 of this License
without being bound by section 3 of the GNU GPL.

  2. Conveying Modified Versions.

  If you modify a copy of the Library, and, in your modifications, a
facility refers to a function or data to be supplied by an Application
that uses the facility (other than as an argument passed when the
facility is invoked), then you may convey a copy of the modified
version:

   a) under this License, provided that you make a good faith effort to
   ensure that, in the event an Application does not supply the
   function or data, the facility still operates, and performs
   whatever part of its purpose remains meaningful, or

   b) under the GNU GPL, with none of the additional permissions of
   this License applicable to that copy.

  3. Object Code Incorporating Material from Library Header Files.

  The object code form of an Application may incorporate material from
a header file that is part of the Library.  You may convey such object
code under terms of your choice, provided that, if the incorporated
material is not limited to numerical parameters, data structure
layouts and accessors, or small macros, inline functions and templates
(ten or fewer lines in length), you do both of the following:
```

(continues on next page)

(continued from previous page)

```
   a) Give prominent notice with each copy of the object code that the
   Library is used in it and that the Library and its use are
   covered by this License.

   b) Accompany the object code with a copy of the GNU GPL and this license
   document.

  4. Combined Works.

  You may convey a Combined Work under terms of your choice that,
taken together, effectively do not restrict modification of the
portions of the Library contained in the Combined Work and reverse
engineering for debugging such modifications, if you also do each of
the following:

   a) Give prominent notice with each copy of the Combined Work that
   the Library is used in it and that the Library and its use are
   covered by this License.

   b) Accompany the Combined Work with a copy of the GNU GPL and this license
   document.

   c) For a Combined Work that displays copyright notices during
   execution, include the copyright notice for the Library among
   these notices, as well as a reference directing the user to the
   copies of the GNU GPL and this license document.

   d) Do one of the following:

       0) Convey the Minimal Corresponding Source under the terms of this
       License, and the Corresponding Application Code in a form
       suitable for, and under terms that permit, the user to
       recombine or relink the Application with a modified version of
       the Linked Version to produce a modified Combined Work, in the
       manner specified by section 6 of the GNU GPL for conveying
       Corresponding Source.

       1) Use a suitable shared library mechanism for linking with the
       Library.  A suitable mechanism is one that (a) uses at run time
       a copy of the Library already present on the user's computer
       system, and (b) will operate properly with a modified version
       of the Library that is interface-compatible with the Linked
       Version.

   e) Provide Installation Information, but only if you would otherwise
   be required to provide such information under section 6 of the
   GNU GPL, and only to the extent that such information is
   necessary to install and execute a modified version of the
   Combined Work produced by recombining or relinking the
   Application with a modified version of the Linked Version. (If
   you use option 4d0, the Installation Information must accompany
```

(continues on next page)

(continued from previous page)

```
    the Minimal Corresponding Source and Corresponding Application
    Code. If you use option 4d1, you must provide the Installation
    Information in the manner specified by section 6 of the GNU GPL
    for conveying Corresponding Source.)

  5. Combined Libraries.

  You may place library facilities that are a work based on the
Library side by side in a single library together with other library
facilities that are not Applications and are not covered by this
License, and convey such a combined library under terms of your
choice, if you do both of the following:

    a) Accompany the combined library with a copy of the same work based
    on the Library, uncombined with any other library facilities,
    conveyed under the terms of this License.

    b) Give prominent notice with the combined library that part of it
    is a work based on the Library, and explaining where to find the
    accompanying uncombined form of the same work.

  6. Revised Versions of the GNU Lesser General Public License.

  The Free Software Foundation may publish revised and/or new versions
of the GNU Lesser General Public License from time to time. Such new
versions will be similar in spirit to the present version, but may
differ in detail to address new problems or concerns.

  Each version is given a distinguishing version number. If the
Library as you received it specifies that a certain numbered version
of the GNU Lesser General Public License "or any later version"
applies to it, you have the option of following the terms and
conditions either of that published version or of any later version
published by the Free Software Foundation. If the Library as you
received it does not specify a version number of the GNU Lesser
General Public License, you may choose any version of the GNU Lesser
General Public License ever published by the Free Software Foundation.

  If the Library as you received it specifies that a proxy can decide
whether future versions of the GNU Lesser General Public License shall
apply, that proxy's public statement of acceptance of any version is
permanent authorization for you to choose that version for the
Library.
```

1.10.1 ALPSO

```
ALPSO - Augmented Lagrangian Particle Swarm Optimizer
Copyright (c) 2011, Dr. Ruben E. Perez (Ruben.Perez@rmc.ca)
                and Peter W. Jansen (Peter.Jansen@rmc.ca)

ALPSO is free software: you can redistribute it and/or modify it
under the terms of the GNU General Public License as published by
the Free Software Foundation, either version 3 of the License, or
any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU General Public License for more details.

You should have received a copy of the GNU General Public License
along with this program. If not, see <http://www.gnu.org/licenses/>.

Non-free versions of ALPSO are available under terms different
from those of the General Public License. For more information
related to such a license, future developments and/or technical
support, contact Dr. Ruben E. Perez (Ruben.Perez@rmc.ca) or
Peter W. Jansen (Peter.Jansen@rmc.ca).

In addition, we kindly ask you to acknowledge the code authors in
any program, application or publication in which you use it. For
published works that use ALPSO we suggest referencing:

P. Jansen and R. Perez, "Constrained Structural Design Optimization
via a Parallel Augmented Lagrangian Particle Swarm Optimization
Approach", International Journal of Computers and Structures,
Vol. 89, No. 13-14, pp. 1352-1366, 2011.

----------------------------------------------------------------------

                    GNU GENERAL PUBLIC LICENSE
                       Version 3, 29 June 2007

 Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/>
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

                            Preamble

  The GNU General Public License is a free, copyleft license for
software and other kinds of works.

  The licenses for most software and other practical works are designed
to take away your freedom to share and change the works.  By contrast,
the GNU General Public License is intended to guarantee your freedom to
share and change all versions of a program--to make sure it remains free
software for all its users.  We, the Free Software Foundation, use the
```

(continues on next page)

(continued from previous page)

```
GNU General Public License for most of our software; it applies also to
any other work released this way by its authors.  You can apply it to
your programs, too.

  When we speak of free software, we are referring to freedom, not
price.  Our General Public Licenses are designed to make sure that you
have the freedom to distribute copies of free software (and charge for
them if you wish), that you receive source code or can get it if you
want it, that you can change the software or use pieces of it in new
free programs, and that you know you can do these things.

  To protect your rights, we need to prevent others from denying you
these rights or asking you to surrender the rights.  Therefore, you have
certain responsibilities if you distribute copies of the software, or if
you modify it: responsibilities to respect the freedom of others.

  For example, if you distribute copies of such a program, whether
gratis or for a fee, you must pass on to the recipients the same
freedoms that you received.  You must make sure that they, too, receive
or can get the source code.  And you must show them these terms so they
know their rights.

  Developers that use the GNU GPL protect your rights with two steps:
(1) assert copyright on the software, and (2) offer you this License
giving you legal permission to copy, distribute and/or modify it.

  For the developers' and authors' protection, the GPL clearly explains
that there is no warranty for this free software.  For both users' and
authors' sake, the GPL requires that modified versions be marked as
changed, so that their problems will not be attributed erroneously to
authors of previous versions.

  Some devices are designed to deny users access to install or run
modified versions of the software inside them, although the manufacturer
can do so.  This is fundamentally incompatible with the aim of
protecting users' freedom to change the software.  The systematic
pattern of such abuse occurs in the area of products for individuals to
use, which is precisely where it is most unacceptable.  Therefore, we
have designed this version of the GPL to prohibit the practice for those
products.  If such problems arise substantially in other domains, we
stand ready to extend this provision to those domains in future versions
of the GPL, as needed to protect the freedom of users.

  Finally, every program is threatened constantly by software patents.
States should not allow patents to restrict development and use of
software on general-purpose computers, but in those that do, we wish to
avoid the special danger that patents applied to a free program could
make it effectively proprietary.  To prevent this, the GPL assures that
patents cannot be used to render the program non-free.

  The precise terms and conditions for copying, distribution and
modification follow.
```

(continues on next page)

(continued from previous page)

```
                    TERMS AND CONDITIONS

  0. Definitions.

  "This License" refers to version 3 of the GNU General Public License.

  "Copyright" also means copyright-like laws that apply to other kinds of
works, such as semiconductor masks.

  "The Program" refers to any copyrightable work licensed under this
License.  Each licensee is addressed as "you".  "Licensees" and
"recipients" may be individuals or organizations.

  To "modify" a work means to copy from or adapt all or part of the work
in a fashion requiring copyright permission, other than the making of an
exact copy.  The resulting work is called a "modified version" of the
earlier work or a work "based on" the earlier work.

  A "covered work" means either the unmodified Program or a work based
on the Program.

  To "propagate" a work means to do anything with it that, without
permission, would make you directly or secondarily liable for
infringement under applicable copyright law, except executing it on a
computer or modifying a private copy.  Propagation includes copying,
distribution (with or without modification), making available to the
public, and in some countries other activities as well.

  To "convey" a work means any kind of propagation that enables other
parties to make or receive copies.  Mere interaction with a user through
a computer network, with no transfer of a copy, is not conveying.

  An interactive user interface displays "Appropriate Legal Notices"
to the extent that it includes a convenient and prominently visible
feature that (1) displays an appropriate copyright notice, and (2)
tells the user that there is no warranty for the work (except to the
extent that warranties are provided), that licensees may convey the
work under this License, and how to view a copy of this License.  If
the interface presents a list of user commands or options, such as a
menu, a prominent item in the list meets this criterion.

  1. Source Code.

  The "source code" for a work means the preferred form of the work
for making modifications to it.  "Object code" means any non-source
form of a work.

  A "Standard Interface" means an interface that either is an official
standard defined by a recognized standards body, or, in the case of
interfaces specified for a particular programming language, one that
is widely used among developers working in that language.
```

(continues on next page)

(continued from previous page)

```
  The "System Libraries" of an executable work include anything, other
than the work as a whole, that (a) is included in the normal form of
packaging a Major Component, but which is not part of that Major
Component, and (b) serves only to enable use of the work with that
Major Component, or to implement a Standard Interface for which an
implementation is available to the public in source code form.  A
"Major Component", in this context, means a major essential component
(kernel, window system, and so on) of the specific operating system
(if any) on which the executable work runs, or a compiler used to
produce the work, or an object code interpreter used to run it.

  The "Corresponding Source" for a work in object code form means all
the source code needed to generate, install, and (for an executable
work) run the object code and to modify the work, including scripts to
control those activities.  However, it does not include the work's
System Libraries, or general-purpose tools or generally available free
programs which are used unmodified in performing those activities but
which are not part of the work.  For example, Corresponding Source
includes interface definition files associated with source files for
the work, and the source code for shared libraries and dynamically
linked subprograms that the work is specifically designed to require,
such as by intimate data communication or control flow between those
subprograms and other parts of the work.

  The Corresponding Source need not include anything that users
can regenerate automatically from other parts of the Corresponding
Source.

  The Corresponding Source for a work in source code form is that
same work.

  2. Basic Permissions.

  All rights granted under this License are granted for the term of
copyright on the Program, and are irrevocable provided the stated
conditions are met.  This License explicitly affirms your unlimited
permission to run the unmodified Program.  The output from running a
covered work is covered by this License only if the output, given its
content, constitutes a covered work.  This License acknowledges your
rights of fair use or other equivalent, as provided by copyright law.

  You may make, run and propagate covered works that you do not
convey, without conditions so long as your license otherwise remains
in force.  You may convey covered works to others for the sole purpose
of having them make modifications exclusively for you, or provide you
with facilities for running those works, provided that you comply with
the terms of this License in conveying all material for which you do
not control copyright.  Those thus making or running the covered works
for you must do so exclusively on your behalf, under your direction
and control, on terms that prohibit them from making any copies of
your copyrighted material outside their relationship with you.
```

(continues on next page)

(continued from previous page)

```
  Conveying under any other circumstances is permitted solely under
the conditions stated below.  Sublicensing is not allowed; section 10
makes it unnecessary.

  3. Protecting Users' Legal Rights From Anti-Circumvention Law.

  No covered work shall be deemed part of an effective technological
measure under any applicable law fulfilling obligations under article
11 of the WIPO copyright treaty adopted on 20 December 1996, or
similar laws prohibiting or restricting circumvention of such
measures.

  When you convey a covered work, you waive any legal power to forbid
circumvention of technological measures to the extent such circumvention
is effected by exercising rights under this License with respect to
the covered work, and you disclaim any intention to limit operation or
modification of the work as a means of enforcing, against the work's
users, your or third parties' legal rights to forbid circumvention of
technological measures.

  4. Conveying Verbatim Copies.

  You may convey verbatim copies of the Program's source code as you
receive it, in any medium, provided that you conspicuously and
appropriately publish on each copy an appropriate copyright notice;
keep intact all notices stating that this License and any
non-permissive terms added in accord with section 7 apply to the code;
keep intact all notices of the absence of any warranty; and give all
recipients a copy of this License along with the Program.

  You may charge any price or no price for each copy that you convey,
and you may offer support or warranty protection for a fee.

  5. Conveying Modified Source Versions.

  You may convey a work based on the Program, or the modifications to
produce it from the Program, in the form of source code under the
terms of section 4, provided that you also meet all of these conditions:

    a) The work must carry prominent notices stating that you modified
    it, and giving a relevant date.

    b) The work must carry prominent notices stating that it is
    released under this License and any conditions added under section
    7.  This requirement modifies the requirement in section 4 to
    "keep intact all notices".

    c) You must license the entire work, as a whole, under this
    License to anyone who comes into possession of a copy.  This
    License will therefore apply, along with any applicable section 7
    additional terms, to the whole of the work, and all its parts,
```

(continues on next page)

(continued from previous page)

```
    regardless of how they are packaged.  This License gives no
    permission to license the work in any other way, but it does not
    invalidate such permission if you have separately received it.

    d) If the work has interactive user interfaces, each must display
    Appropriate Legal Notices; however, if the Program has interactive
    interfaces that do not display Appropriate Legal Notices, your
    work need not make them do so.

  A compilation of a covered work with other separate and independent
works, which are not by their nature extensions of the covered work,
and which are not combined with it such as to form a larger program,
in or on a volume of a storage or distribution medium, is called an
"aggregate" if the compilation and its resulting copyright are not
used to limit the access or legal rights of the compilation's users
beyond what the individual works permit.  Inclusion of a covered work
in an aggregate does not cause this License to apply to the other
parts of the aggregate.

  6. Conveying Non-Source Forms.

  You may convey a covered work in object code form under the terms
of sections 4 and 5, provided that you also convey the
machine-readable Corresponding Source under the terms of this License,
in one of these ways:

    a) Convey the object code in, or embodied in, a physical product
    (including a physical distribution medium), accompanied by the
    Corresponding Source fixed on a durable physical medium
    customarily used for software interchange.

    b) Convey the object code in, or embodied in, a physical product
    (including a physical distribution medium), accompanied by a
    written offer, valid for at least three years and valid for as
    long as you offer spare parts or customer support for that product
    model, to give anyone who possesses the object code either (1) a
    copy of the Corresponding Source for all the software in the
    product that is covered by this License, on a durable physical
    medium customarily used for software interchange, for a price no
    more than your reasonable cost of physically performing this
    conveying of source, or (2) access to copy the
    Corresponding Source from a network server at no charge.

    c) Convey individual copies of the object code with a copy of the
    written offer to provide the Corresponding Source.  This
    alternative is allowed only occasionally and noncommercially, and
    only if you received the object code with such an offer, in accord
    with subsection 6b.

    d) Convey the object code by offering access from a designated
    place (gratis or for a charge), and offer equivalent access to the
    Corresponding Source in the same way through the same place at no
```

(continues on next page)

(continued from previous page)

```
    further charge.  You need not require recipients to copy the
    Corresponding Source along with the object code.  If the place to
    copy the object code is a network server, the Corresponding Source
    may be on a different server (operated by you or a third party)
    that supports equivalent copying facilities, provided you maintain
    clear directions next to the object code saying where to find the
    Corresponding Source.  Regardless of what server hosts the
    Corresponding Source, you remain obligated to ensure that it is
    available for as long as needed to satisfy these requirements.

    e) Convey the object code using peer-to-peer transmission, provided
    you inform other peers where the object code and Corresponding
    Source of the work are being offered to the general public at no
    charge under subsection 6d.

  A separable portion of the object code, whose source code is excluded
from the Corresponding Source as a System Library, need not be
included in conveying the object code work.

  A "User Product" is either (1) a "consumer product", which means any
tangible personal property which is normally used for personal, family,
or household purposes, or (2) anything designed or sold for incorporation
into a dwelling.  In determining whether a product is a consumer product,
doubtful cases shall be resolved in favor of coverage.  For a particular
product received by a particular user, "normally used" refers to a
typical or common use of that class of product, regardless of the status
of the particular user or of the way in which the particular user
actually uses, or expects or is expected to use, the product.  A product
is a consumer product regardless of whether the product has substantial
commercial, industrial or non-consumer uses, unless such uses represent
the only significant mode of use of the product.

  "Installation Information" for a User Product means any methods,
procedures, authorization keys, or other information required to install
and execute modified versions of a covered work in that User Product from
a modified version of its Corresponding Source.  The information must
suffice to ensure that the continued functioning of the modified object
code is in no case prevented or interfered with solely because
modification has been made.

  If you convey an object code work under this section in, or with, or
specifically for use in, a User Product, and the conveying occurs as
part of a transaction in which the right of possession and use of the
User Product is transferred to the recipient in perpetuity or for a
fixed term (regardless of how the transaction is characterized), the
Corresponding Source conveyed under this section must be accompanied
by the Installation Information.  But this requirement does not apply
if neither you nor any third party retains the ability to install
modified object code on the User Product (for example, the work has
been installed in ROM).

  The requirement to provide Installation Information does not include a
```

(continues on next page)

(continued from previous page)

```
requirement to continue to provide support service, warranty, or updates
for a work that has been modified or installed by the recipient, or for
the User Product in which it has been modified or installed.  Access to a
network may be denied when the modification itself materially and
adversely affects the operation of the network or violates the rules and
protocols for communication across the network.

  Corresponding Source conveyed, and Installation Information provided,
in accord with this section must be in a format that is publicly
documented (and with an implementation available to the public in
source code form), and must require no special password or key for
unpacking, reading or copying.

  7. Additional Terms.

  "Additional permissions" are terms that supplement the terms of this
License by making exceptions from one or more of its conditions.
Additional permissions that are applicable to the entire Program shall
be treated as though they were included in this License, to the extent
that they are valid under applicable law.  If additional permissions
apply only to part of the Program, that part may be used separately
under those permissions, but the entire Program remains governed by
this License without regard to the additional permissions.

  When you convey a copy of a covered work, you may at your option
remove any additional permissions from that copy, or from any part of
it.  (Additional permissions may be written to require their own
removal in certain cases when you modify the work.)  You may place
additional permissions on material, added by you to a covered work,
for which you have or can give appropriate copyright permission.

  Notwithstanding any other provision of this License, for material you
add to a covered work, you may (if authorized by the copyright holders of
that material) supplement the terms of this License with terms:

    a) Disclaiming warranty or limiting liability differently from the
    terms of sections 15 and 16 of this License; or

    b) Requiring preservation of specified reasonable legal notices or
    author attributions in that material or in the Appropriate Legal
    Notices displayed by works containing it; or

    c) Prohibiting misrepresentation of the origin of that material, or
    requiring that modified versions of such material be marked in
    reasonable ways as different from the original version; or

    d) Limiting the use for publicity purposes of names of licensors or
    authors of the material; or

    e) Declining to grant rights under trademark law for use of some
    trade names, trademarks, or service marks; or
```

(continues on next page)

(continued from previous page)

```
    f) Requiring indemnification of licensors and authors of that
    material by anyone who conveys the material (or modified versions of
    it) with contractual assumptions of liability to the recipient, for
    any liability that these contractual assumptions directly impose on
    those licensors and authors.

  All other non-permissive additional terms are considered "further
restrictions" within the meaning of section 10.  If the Program as you
received it, or any part of it, contains a notice stating that it is
governed by this License along with a term that is a further
restriction, you may remove that term.  If a license document contains
a further restriction but permits relicensing or conveying under this
License, you may add to a covered work material governed by the terms
of that license document, provided that the further restriction does
not survive such relicensing or conveying.

  If you add terms to a covered work in accord with this section, you
must place, in the relevant source files, a statement of the
additional terms that apply to those files, or a notice indicating
where to find the applicable terms.

  Additional terms, permissive or non-permissive, may be stated in the
form of a separately written license, or stated as exceptions;
the above requirements apply either way.

  8. Termination.

  You may not propagate or modify a covered work except as expressly
provided under this License.  Any attempt otherwise to propagate or
modify it is void, and will automatically terminate your rights under
this License (including any patent licenses granted under the third
paragraph of section 11).

  However, if you cease all violation of this License, then your
license from a particular copyright holder is reinstated (a)
provisionally, unless and until the copyright holder explicitly and
finally terminates your license, and (b) permanently, if the copyright
holder fails to notify you of the violation by some reasonable means
prior to 60 days after the cessation.

  Moreover, your license from a particular copyright holder is
reinstated permanently if the copyright holder notifies you of the
violation by some reasonable means, this is the first time you have
received notice of violation of this License (for any work) from that
copyright holder, and you cure the violation prior to 30 days after
your receipt of the notice.

  Termination of your rights under this section does not terminate the
licenses of parties who have received copies or rights from you under
this License.  If your rights have been terminated and not permanently
reinstated, you do not qualify to receive new licenses for the same
material under section 10.
```

(continues on next page)

(continued from previous page)

```
  9. Acceptance Not Required for Having Copies.

  You are not required to accept this License in order to receive or
run a copy of the Program.  Ancillary propagation of a covered work
occurring solely as a consequence of using peer-to-peer transmission
to receive a copy likewise does not require acceptance.  However,
nothing other than this License grants you permission to propagate or
modify any covered work.  These actions infringe copyright if you do
not accept this License.  Therefore, by modifying or propagating a
covered work, you indicate your acceptance of this License to do so.

  10. Automatic Licensing of Downstream Recipients.

  Each time you convey a covered work, the recipient automatically
receives a license from the original licensors, to run, modify and
propagate that work, subject to this License.  You are not responsible
for enforcing compliance by third parties with this License.

  An "entity transaction" is a transaction transferring control of an
organization, or substantially all assets of one, or subdividing an
organization, or merging organizations.  If propagation of a covered
work results from an entity transaction, each party to that
transaction who receives a copy of the work also receives whatever
licenses to the work the party's predecessor in interest had or could
give under the previous paragraph, plus a right to possession of the
Corresponding Source of the work from the predecessor in interest, if
the predecessor has it or can get it with reasonable efforts.

  You may not impose any further restrictions on the exercise of the
rights granted or affirmed under this License.  For example, you may
not impose a license fee, royalty, or other charge for exercise of
rights granted under this License, and you may not initiate litigation
(including a cross-claim or counterclaim in a lawsuit) alleging that
any patent claim is infringed by making, using, selling, offering for
sale, or importing the Program or any portion of it.

  11. Patents.

  A "contributor" is a copyright holder who authorizes use under this
License of the Program or a work on which the Program is based.  The
work thus licensed is called the contributor's "contributor version".

  A contributor's "essential patent claims" are all patent claims
owned or controlled by the contributor, whether already acquired or
hereafter acquired, that would be infringed by some manner, permitted
by this License, of making, using, or selling its contributor version,
but do not include claims that would be infringed only as a
consequence of further modification of the contributor version.  For
purposes of this definition, "control" includes the right to grant
patent sublicenses in a manner consistent with the requirements of
this License.
```

(continues on next page)

(continued from previous page)

```
  Each contributor grants you a non-exclusive, worldwide, royalty-free
patent license under the contributor's essential patent claims, to
make, use, sell, offer for sale, import and otherwise run, modify and
propagate the contents of its contributor version.

  In the following three paragraphs, a "patent license" is any express
agreement or commitment, however denominated, not to enforce a patent
(such as an express permission to practice a patent or covenant not to
sue for patent infringement).  To "grant" such a patent license to a
party means to make such an agreement or commitment not to enforce a
patent against the party.

  If you convey a covered work, knowingly relying on a patent license,
and the Corresponding Source of the work is not available for anyone
to copy, free of charge and under the terms of this License, through a
publicly available network server or other readily accessible means,
then you must either (1) cause the Corresponding Source to be so
available, or (2) arrange to deprive yourself of the benefit of the
patent license for this particular work, or (3) arrange, in a manner
consistent with the requirements of this License, to extend the patent
license to downstream recipients.  "Knowingly relying" means you have
actual knowledge that, but for the patent license, your conveying the
covered work in a country, or your recipient's use of the covered work
in a country, would infringe one or more identifiable patents in that
country that you have reason to believe are valid.

  If, pursuant to or in connection with a single transaction or
arrangement, you convey, or propagate by procuring conveyance of, a
covered work, and grant a patent license to some of the parties
receiving the covered work authorizing them to use, propagate, modify
or convey a specific copy of the covered work, then the patent license
you grant is automatically extended to all recipients of the covered
work and works based on it.

  A patent license is "discriminatory" if it does not include within
the scope of its coverage, prohibits the exercise of, or is
conditioned on the non-exercise of one or more of the rights that are
specifically granted under this License.  You may not convey a covered
work if you are a party to an arrangement with a third party that is
in the business of distributing software, under which you make payment
to the third party based on the extent of your activity of conveying
the work, and under which the third party grants, to any of the
parties who would receive the covered work from you, a discriminatory
patent license (a) in connection with copies of the covered work
conveyed by you (or copies made from those copies), or (b) primarily
for and in connection with specific products or compilations that
contain the covered work, unless you entered into that arrangement,
or that patent license was granted, prior to 28 March 2007.

  Nothing in this License shall be construed as excluding or limiting
any implied license or other defenses to infringement that may
```

(continues on next page)

(continued from previous page)

```
otherwise be available to you under applicable patent law.

  12. No Surrender of Others' Freedom.

  If conditions are imposed on you (whether by court order, agreement or
otherwise) that contradict the conditions of this License, they do not
excuse you from the conditions of this License.  If you cannot convey a
covered work so as to satisfy simultaneously your obligations under this
License and any other pertinent obligations, then as a consequence you may
not convey it at all.  For example, if you agree to terms that obligate you
to collect a royalty for further conveying from those to whom you convey
the Program, the only way you could satisfy both those terms and this
License would be to refrain entirely from conveying the Program.

  13. Use with the GNU Affero General Public License.

  Notwithstanding any other provision of this License, you have
permission to link or combine any covered work with a work licensed
under version 3 of the GNU Affero General Public License into a single
combined work, and to convey the resulting work.  The terms of this
License will continue to apply to the part which is the covered work,
but the special requirements of the GNU Affero General Public License,
section 13, concerning interaction through a network will apply to the
combination as such.

  14. Revised Versions of this License.

  The Free Software Foundation may publish revised and/or new versions of
the GNU General Public License from time to time.  Such new versions will
be similar in spirit to the present version, but may differ in detail to
address new problems or concerns.

  Each version is given a distinguishing version number.  If the
Program specifies that a certain numbered version of the GNU General
Public License "or any later version" applies to it, you have the
option of following the terms and conditions either of that numbered
version or of any later version published by the Free Software
Foundation.  If the Program does not specify a version number of the
GNU General Public License, you may choose any version ever published
by the Free Software Foundation.

  If the Program specifies that a proxy can decide which future
versions of the GNU General Public License can be used, that proxy's
public statement of acceptance of a version permanently authorizes you
to choose that version for the Program.

  Later license versions may give you additional or different
permissions.  However, no additional obligations are imposed on any
author or copyright holder as a result of your choosing to follow a
later version.

  15. Disclaimer of Warranty.
```

(continues on next page)

(continued from previous page)

```
  THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY
APPLICABLE LAW.  EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT
HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY
OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO,
THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR
PURPOSE.  THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM
IS WITH YOU.  SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF
ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

  16. Limitation of Liability.

  IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING
WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS
THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY
GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE
USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF
DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD
PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS),
EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF
SUCH DAMAGES.

  17. Interpretation of Sections 15 and 16.

  If the disclaimer of warranty and limitation of liability provided
above cannot be given local legal effect according to their terms,
reviewing courts shall apply local law that most closely approximates
an absolute waiver of all civil liability in connection with the
Program, unless a warranty or assumption of liability accompanies a
copy of the Program in return for a fee.

                     END OF TERMS AND CONDITIONS

            How to Apply These Terms to Your New Programs

  If you develop a new program, and you want it to be of the greatest
possible use to the public, the best way to achieve this is to make it
free software which everyone can redistribute and change under these terms.

  To do so, attach the following notices to the program.  It is safest
to attach them to the start of each source file to most effectively
state the exclusion of warranty; and each file should have at least
the "copyright" line and a pointer to where the full notice is found.

    <one line to give the program's name and a brief idea of what it does.>
    Copyright (C) <year>  <name of author>

    This program is free software: you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation, either version 3 of the License, or
    (at your option) any later version.
```

(continues on next page)

(continued from previous page)

```
    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.

    You should have received a copy of the GNU General Public License
    along with this program.  If not, see <http://www.gnu.org/licenses/>.

Also add information on how to contact you by electronic and paper mail.

  If the program does terminal interaction, make it output a short
notice like this when it starts in an interactive mode:

    <program>  Copyright (C) <year>  <name of author>
    This program comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
    This is free software, and you are welcome to redistribute it
    under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate
parts of the General Public License.  Of course, your program's commands
might be different; for a GUI interface, you would use an "about box".

  You should also get your employer (if you work as a programmer) or school,
if any, to sign a "copyright disclaimer" for the program, if necessary.
For more information on this, and how to apply and follow the GNU GPL, see
<http://www.gnu.org/licenses/>.

  The GNU General Public License does not permit incorporating your program
into proprietary programs.  If your program is a subroutine library, you
may consider it more useful to permit linking proprietary applications with
the library.  If this is what you want to do, use the GNU Lesser General
Public License instead of this License.  But first, please read
<http://www.gnu.org/philosophy/why-not-lgpl.html>.
```

1.10.2 CONMIN

```
CONMIN - CONstrained function MINimization
Copyright (c) 1972, Gary N. Vanderplaats
Modified for pyOpt - 2008, Ruben E. Perez (Ruben.Perez@rmc.ca)

This program is free software; you can redistribute it and/or modify
it under the terms of the GNU General Public License as published by
the Free Software Foundation; either version 2 of the License, or
(at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
GNU General Public License for more details.
```

(continues on next page)

(continued from previous page)

```
You should have received a copy of the GNU General Public License
along with this program; if not, write to the Free Software
Foundation, Inc., 59 Temple Place, Suite 330, Boston,
MA 02111-1307 USA.

------------------------------------------------------------------------

                    GNU GENERAL PUBLIC LICENSE
                       Version 2, June 1991

 Copyright (C) 1989, 1991 Free Software Foundation, Inc.,
 51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.

                            Preamble

  The licenses for most software are designed to take away your
freedom to share and change it.  By contrast, the GNU General Public
License is intended to guarantee your freedom to share and change free
software--to make sure the software is free for all its users.  This
General Public License applies to most of the Free Software
Foundation's software and to any other program whose authors commit to
using it.  (Some other Free Software Foundation software is covered by
the GNU Lesser General Public License instead.)  You can apply it to
your programs, too.

  When we speak of free software, we are referring to freedom, not
price.  Our General Public Licenses are designed to make sure that you
have the freedom to distribute copies of free software (and charge for
this service if you wish), that you receive source code or can get it
if you want it, that you can change the software or use pieces of it
in new free programs; and that you know you can do these things.

  To protect your rights, we need to make restrictions that forbid
anyone to deny you these rights or to ask you to surrender the rights.
These restrictions translate to certain responsibilities for you if you
distribute copies of the software, or if you modify it.

  For example, if you distribute copies of such a program, whether
gratis or for a fee, you must give the recipients all the rights that
you have.  You must make sure that they, too, receive or can get the
source code.  And you must show them these terms so they know their
rights.

  We protect your rights with two steps: (1) copyright the software, and
(2) offer you this license which gives you legal permission to copy,
distribute and/or modify the software.

  Also, for each author's protection and ours, we want to make certain
that everyone understands that there is no warranty for this free
software.  If the software is modified by someone else and passed on, we
```

(continues on next page)

(continued from previous page)

```
want its recipients to know that what they have is not the original, so
that any problems introduced by others will not reflect on the original
authors' reputations.

  Finally, any free program is threatened constantly by software
patents.  We wish to avoid the danger that redistributors of a free
program will individually obtain patent licenses, in effect making the
program proprietary.  To prevent this, we have made it clear that any
patent must be licensed for everyone's free use or not licensed at all.

  The precise terms and conditions for copying, distribution and
modification follow.

                    GNU GENERAL PUBLIC LICENSE
   TERMS AND CONDITIONS FOR COPYING, DISTRIBUTION AND MODIFICATION

  0. This License applies to any program or other work which contains
a notice placed by the copyright holder saying it may be distributed
under the terms of this General Public License.  The "Program", below,
refers to any such program or work, and a "work based on the Program"
means either the Program or any derivative work under copyright law:
that is to say, a work containing the Program or a portion of it,
either verbatim or with modifications and/or translated into another
language.  (Hereinafter, translation is included without limitation in
the term "modification".)  Each licensee is addressed as "you".

Activities other than copying, distribution and modification are not
covered by this License; they are outside its scope.  The act of
running the Program is not restricted, and the output from the Program
is covered only if its contents constitute a work based on the
Program (independent of having been made by running the Program).
Whether that is true depends on what the Program does.

  1. You may copy and distribute verbatim copies of the Program's
source code as you receive it, in any medium, provided that you
conspicuously and appropriately publish on each copy an appropriate
copyright notice and disclaimer of warranty; keep intact all the
notices that refer to this License and to the absence of any warranty;
and give any other recipients of the Program a copy of this License
along with the Program.

You may charge a fee for the physical act of transferring a copy, and
you may at your option offer warranty protection in exchange for a fee.

  2. You may modify your copy or copies of the Program or any portion
of it, thus forming a work based on the Program, and copy and
distribute such modifications or work under the terms of Section 1
above, provided that you also meet all of these conditions:

    a) You must cause the modified files to carry prominent notices
    stating that you changed the files and the date of any change.
```

(continues on next page)

(continued from previous page)

```
    b) You must cause any work that you distribute or publish, that in
    whole or in part contains or is derived from the Program or any
    part thereof, to be licensed as a whole at no charge to all third
    parties under the terms of this License.

    c) If the modified program normally reads commands interactively
    when run, you must cause it, when started running for such
    interactive use in the most ordinary way, to print or display an
    announcement including an appropriate copyright notice and a
    notice that there is no warranty (or else, saying that you provide
    a warranty) and that users may redistribute the program under
    these conditions, and telling the user how to view a copy of this
    License.  (Exception: if the Program itself is interactive but
    does not normally print such an announcement, your work based on
    the Program is not required to print an announcement.)

These requirements apply to the modified work as a whole.  If
identifiable sections of that work are not derived from the Program,
and can be reasonably considered independent and separate works in
themselves, then this License, and its terms, do not apply to those
sections when you distribute them as separate works.  But when you
distribute the same sections as part of a whole which is a work based
on the Program, the distribution of the whole must be on the terms of
this License, whose permissions for other licensees extend to the
entire whole, and thus to each and every part regardless of who wrote it.

Thus, it is not the intent of this section to claim rights or contest
your rights to work written entirely by you; rather, the intent is to
exercise the right to control the distribution of derivative or
collective works based on the Program.

In addition, mere aggregation of another work not based on the Program
with the Program (or with a work based on the Program) on a volume of
a storage or distribution medium does not bring the other work under
the scope of this License.

  3. You may copy and distribute the Program (or a work based on it,
under Section 2) in object code or executable form under the terms of
Sections 1 and 2 above provided that you also do one of the following:

    a) Accompany it with the complete corresponding machine-readable
    source code, which must be distributed under the terms of Sections
    1 and 2 above on a medium customarily used for software interchange; or,

    b) Accompany it with a written offer, valid for at least three
    years, to give any third party, for a charge no more than your
    cost of physically performing source distribution, a complete
    machine-readable copy of the corresponding source code, to be
    distributed under the terms of Sections 1 and 2 above on a medium
    customarily used for software interchange; or,

    c) Accompany it with the information you received as to the offer
```

(continues on next page)

(continued from previous page)

```
    to distribute corresponding source code.  (This alternative is
    allowed only for noncommercial distribution and only if you
    received the program in object code or executable form with such
    an offer, in accord with Subsection b above.)

The source code for a work means the preferred form of the work for
making modifications to it.  For an executable work, complete source
code means all the source code for all modules it contains, plus any
associated interface definition files, plus the scripts used to
control compilation and installation of the executable.  However, as a
special exception, the source code distributed need not include
anything that is normally distributed (in either source or binary
form) with the major components (compiler, kernel, and so on) of the
operating system on which the executable runs, unless that component
itself accompanies the executable.

If distribution of executable or object code is made by offering
access to copy from a designated place, then offering equivalent
access to copy the source code from the same place counts as
distribution of the source code, even though third parties are not
compelled to copy the source along with the object code.

  4. You may not copy, modify, sublicense, or distribute the Program
except as expressly provided under this License.  Any attempt
otherwise to copy, modify, sublicense or distribute the Program is
void, and will automatically terminate your rights under this License.
However, parties who have received copies, or rights, from you under
this License will not have their licenses terminated so long as such
parties remain in full compliance.

  5. You are not required to accept this License, since you have not
signed it.  However, nothing else grants you permission to modify or
distribute the Program or its derivative works.  These actions are
prohibited by law if you do not accept this License.  Therefore, by
modifying or distributing the Program (or any work based on the
Program), you indicate your acceptance of this License to do so, and
all its terms and conditions for copying, distributing or modifying
the Program or works based on it.

  6. Each time you redistribute the Program (or any work based on the
Program), the recipient automatically receives a license from the
original licensor to copy, distribute or modify the Program subject to
these terms and conditions.  You may not impose any further
restrictions on the recipients' exercise of the rights granted herein.
You are not responsible for enforcing compliance by third parties to
this License.

  7. If, as a consequence of a court judgment or allegation of patent
infringement or for any other reason (not limited to patent issues),
conditions are imposed on you (whether by court order, agreement or
otherwise) that contradict the conditions of this License, they do not
excuse you from the conditions of this License.  If you cannot
```

(continues on next page)

(continued from previous page)

```
distribute so as to satisfy simultaneously your obligations under this
License and any other pertinent obligations, then as a consequence you
may not distribute the Program at all.  For example, if a patent
license would not permit royalty-free redistribution of the Program by
all those who receive copies directly or indirectly through you, then
the only way you could satisfy both it and this License would be to
refrain entirely from distribution of the Program.

If any portion of this section is held invalid or unenforceable under
any particular circumstance, the balance of the section is intended to
apply and the section as a whole is intended to apply in other
circumstances.

It is not the purpose of this section to induce you to infringe any
patents or other property right claims or to contest validity of any
such claims; this section has the sole purpose of protecting the
integrity of the free software distribution system, which is
implemented by public license practices.  Many people have made
generous contributions to the wide range of software distributed
through that system in reliance on consistent application of that
system; it is up to the author/donor to decide if he or she is willing
to distribute software through any other system and a licensee cannot
impose that choice.

This section is intended to make thoroughly clear what is believed to
be a consequence of the rest of this License.

  8. If the distribution and/or use of the Program is restricted in
certain countries either by patents or by copyrighted interfaces, the
original copyright holder who places the Program under this License
may add an explicit geographical distribution limitation excluding
those countries, so that distribution is permitted only in or among
countries not thus excluded.  In such case, this License incorporates
the limitation as if written in the body of this License.

  9. The Free Software Foundation may publish revised and/or new versions
of the General Public License from time to time.  Such new versions will
be similar in spirit to the present version, but may differ in detail to
address new problems or concerns.

Each version is given a distinguishing version number.  If the Program
specifies a version number of this License which applies to it and "any
later version", you have the option of following the terms and conditions
either of that version or of any later version published by the Free
Software Foundation.  If the Program does not specify a version number of
this License, you may choose any version ever published by the Free Software
Foundation.

  10. If you wish to incorporate parts of the Program into other free
programs whose distribution conditions are different, write to the author
to ask for permission.  For software which is copyrighted by the Free
Software Foundation, write to the Free Software Foundation; we sometimes
```

(continues on next page)

(continued from previous page)

```
make exceptions for this.  Our decision will be guided by the two goals
of preserving the free status of all derivatives of our free software and
of promoting the sharing and reuse of software generally.

                            NO WARRANTY

  11. BECAUSE THE PROGRAM IS LICENSED FREE OF CHARGE, THERE IS NO WARRANTY
FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW.  EXCEPT WHEN
OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES
PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED
OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF
MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE.  THE ENTIRE RISK AS
TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU.  SHOULD THE
PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING,
REPAIR OR CORRECTION.

  12. IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING
WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MAY MODIFY AND/OR
REDISTRIBUTE THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES,
INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING
OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED
TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY
YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER
PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE
POSSIBILITY OF SUCH DAMAGES.

                     END OF TERMS AND CONDITIONS

            How to Apply These Terms to Your New Programs

  If you develop a new program, and you want it to be of the greatest
possible use to the public, the best way to achieve this is to make it
free software which everyone can redistribute and change under these terms.

  To do so, attach the following notices to the program.  It is safest
to attach them to the start of each source file to most effectively
convey the exclusion of warranty; and each file should have at least
the "copyright" line and a pointer to where the full notice is found.

    <one line to give the program's name and a brief idea of what it does.>
    Copyright (C) <year>  <name of author>

    This program is free software; you can redistribute it and/or modify
    it under the terms of the GNU General Public License as published by
    the Free Software Foundation; either version 2 of the License, or
    (at your option) any later version.

    This program is distributed in the hope that it will be useful,
    but WITHOUT ANY WARRANTY; without even the implied warranty of
    MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
    GNU General Public License for more details.
```

(continues on next page)

(continued from previous page)

```
    You should have received a copy of the GNU General Public License along
    with this program; if not, write to the Free Software Foundation, Inc.,
    51 Franklin Street, Fifth Floor, Boston, MA 02110-1301 USA.

Also add information on how to contact you by electronic and paper mail.

If the program is interactive, make it output a short notice like this
when it starts in an interactive mode:

    Gnomovision version 69, Copyright (C) year name of author
    Gnomovision comes with ABSOLUTELY NO WARRANTY; for details type `show w'.
    This is free software, and you are welcome to redistribute it
    under certain conditions; type `show c' for details.

The hypothetical commands `show w' and `show c' should show the appropriate
parts of the General Public License.  Of course, the commands you use may
be called something other than `show w' and `show c'; they could even be
mouse-clicks or menu items--whatever suits your program.

You should also get your employer (if you work as a programmer) or your
school, if any, to sign a "copyright disclaimer" for the program, if
necessary.  Here is a sample; alter the names:

  Yoyodyne, Inc., hereby disclaims all copyright interest in the program
  `Gnomovision' (which makes passes at compilers) written by James Hacker.

  <signature of Ty Coon>, 1 April 1989
  Ty Coon, President of Vice

This General Public License does not permit incorporating your program into
proprietary programs.  If your program is a subroutine library, you may
consider it more useful to permit linking proprietary applications with the
library.  If this is what you want to do, use the GNU Lesser General
Public License instead of this License.
```

1.10.3 NSGA2

```
NSGA2 - Non Sorting Genetic Algorithm II
Copyright (c) 2005, Kalyanmoy Deb (deb@iitk.ac.in)
Modified for pyOpt - 2008, Ruben E. Perez (Ruben.Perez@rmc.ca)

Permission is hereby granted, free of charge, to any person obtaining a
copy of this software and associated documentation files (the
"Software"), to deal in the Software without restriction, including
without limitation the rights to use, copy, modify, merge, publish,
copies of the Software, and to permit persons to whom the Software
is furnished to do so, subject to the following conditions:

Distribution, sublicense, and/or sell of this code for any commercial
purpose is permissible ONLY BY DIRECT ARRANGEMENT WITH THE COPYRIGHT OWNER.
```

(continues on next page)

(continued from previous page)

```
The above copyright notice and this permission notice shall be included
in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

1.10.4 PSQP

```
PSQP - Preconditioned Sequential Quadratic Programming
Copyright (c) 2007, Ladislav Luksan (luksan@uivt.cas.cz)
Modified for pyOpt - 2010, Ruben E. Perez (Ruben.Perez@rmc.ca)

This program is free software: you can redistribute it and/or modify
it under the terms of the GNU Lesser General Public License as
published by the Free Software Foundation, either version 3 of
the License, or (at your option) any later version.

This program is distributed in the hope that it will be useful,
but WITHOUT ANY WARRANTY; without even the implied warranty of
MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE.  See the
GNU Lesser General Public License for more details.

You should have received a copy of the GNU Lesser General
Public License along with this program.  If not, see
<http://www.gnu.org/licenses/lgpl.html>.

------------------------------------------------------------------------

                   GNU LESSER GENERAL PUBLIC LICENSE
                       Version 3, 29 June 2007

 Copyright (C) 2007 Free Software Foundation, Inc. <http://fsf.org/>
 Everyone is permitted to copy and distribute verbatim copies
 of this license document, but changing it is not allowed.


  This version of the GNU Lesser General Public License incorporates
the terms and conditions of version 3 of the GNU General Public
License, supplemented by the additional permissions listed below.

  0. Additional Definitions.

  As used herein, "this License" refers to version 3 of the GNU Lesser
General Public License, and the "GNU GPL" refers to version 3 of the GNU
```

(continues on next page)

(continued from previous page)

```
General Public License.

  "The Library" refers to a covered work governed by this License,
other than an Application or a Combined Work as defined below.

  An "Application" is any work that makes use of an interface provided
by the Library, but which is not otherwise based on the Library.
Defining a subclass of a class defined by the Library is deemed a mode
of using an interface provided by the Library.

  A "Combined Work" is a work produced by combining or linking an
Application with the Library.  The particular version of the Library
with which the Combined Work was made is also called the "Linked
Version".

  The "Minimal Corresponding Source" for a Combined Work means the
Corresponding Source for the Combined Work, excluding any source code
for portions of the Combined Work that, considered in isolation, are
based on the Application, and not on the Linked Version.

  The "Corresponding Application Code" for a Combined Work means the
object code and/or source code for the Application, including any data
and utility programs needed for reproducing the Combined Work from the
Application, but excluding the System Libraries of the Combined Work.

  1. Exception to Section 3 of the GNU GPL.

  You may convey a covered work under sections 3 and 4 of this License
without being bound by section 3 of the GNU GPL.

  2. Conveying Modified Versions.

  If you modify a copy of the Library, and, in your modifications, a
facility refers to a function or data to be supplied by an Application
that uses the facility (other than as an argument passed when the
facility is invoked), then you may convey a copy of the modified
version:

   a) under this License, provided that you make a good faith effort to
   ensure that, in the event an Application does not supply the
   function or data, the facility still operates, and performs
   whatever part of its purpose remains meaningful, or

   b) under the GNU GPL, with none of the additional permissions of
   this License applicable to that copy.

  3. Object Code Incorporating Material from Library Header Files.

  The object code form of an Application may incorporate material from
a header file that is part of the Library.  You may convey such object
code under terms of your choice, provided that, if the incorporated
material is not limited to numerical parameters, data structure
```

(continues on next page)

(continued from previous page)

```
layouts and accessors, or small macros, inline functions and templates
(ten or fewer lines in length), you do both of the following:

   a) Give prominent notice with each copy of the object code that the
   Library is used in it and that the Library and its use are
   covered by this License.

   b) Accompany the object code with a copy of the GNU GPL and this license
   document.

  4. Combined Works.

  You may convey a Combined Work under terms of your choice that,
taken together, effectively do not restrict modification of the
portions of the Library contained in the Combined Work and reverse
engineering for debugging such modifications, if you also do each of
the following:

   a) Give prominent notice with each copy of the Combined Work that
   the Library is used in it and that the Library and its use are
   covered by this License.

   b) Accompany the Combined Work with a copy of the GNU GPL and this license
   document.

   c) For a Combined Work that displays copyright notices during
   execution, include the copyright notice for the Library among
   these notices, as well as a reference directing the user to the
   copies of the GNU GPL and this license document.

   d) Do one of the following:

       0) Convey the Minimal Corresponding Source under the terms of this
       License, and the Corresponding Application Code in a form
       suitable for, and under terms that permit, the user to
       recombine or relink the Application with a modified version of
       the Linked Version to produce a modified Combined Work, in the
       manner specified by section 6 of the GNU GPL for conveying
       Corresponding Source.

       1) Use a suitable shared library mechanism for linking with the
       Library.  A suitable mechanism is one that (a) uses at run time
       a copy of the Library already present on the user's computer
       system, and (b) will operate properly with a modified version
       of the Library that is interface-compatible with the Linked
       Version.

   e) Provide Installation Information, but only if you would otherwise
   be required to provide such information under section 6 of the
   GNU GPL, and only to the extent that such information is
   necessary to install and execute a modified version of the
   Combined Work produced by recombining or relinking the
```

(continues on next page)

(continued from previous page)

```
   Application with a modified version of the Linked Version. (If
   you use option 4d0, the Installation Information must accompany
   the Minimal Corresponding Source and Corresponding Application
   Code. If you use option 4d1, you must provide the Installation
   Information in the manner specified by section 6 of the GNU GPL
   for conveying Corresponding Source.)

  5. Combined Libraries.

  You may place library facilities that are a work based on the
Library side by side in a single library together with other library
facilities that are not Applications and are not covered by this
License, and convey such a combined library under terms of your
choice, if you do both of the following:

   a) Accompany the combined library with a copy of the same work based
   on the Library, uncombined with any other library facilities,
   conveyed under the terms of this License.

   b) Give prominent notice with the combined library that part of it
   is a work based on the Library, and explaining where to find the
   accompanying uncombined form of the same work.

  6. Revised Versions of the GNU Lesser General Public License.

  The Free Software Foundation may publish revised and/or new versions
of the GNU Lesser General Public License from time to time. Such new
versions will be similar in spirit to the present version, but may
differ in detail to address new problems or concerns.

  Each version is given a distinguishing version number. If the
Library as you received it specifies that a certain numbered version
of the GNU Lesser General Public License "or any later version"
applies to it, you have the option of following the terms and
conditions either of that published version or of any later version
published by the Free Software Foundation. If the Library as you
received it does not specify a version number of the GNU Lesser
General Public License, you may choose any version of the GNU Lesser
General Public License ever published by the Free Software Foundation.

  If the Library as you received it specifies that a proxy can decide
whether future versions of the GNU Lesser General Public License shall
apply, that proxy's public statement of acceptance of any version is
permanent authorization for you to choose that version for the
Library.
```

1.10.5 SLSQP

```
SLSQP - Sequential Least Squares Programming
Copyright (c) 1988, Dieter Kraft (kraft@hm.edu)
Copyright (c) 1994, Association for Computing Machinery
Patched for pyOpt - 2008, Ruben E. Perez (Ruben.Perez@rmc.ca)

Permission is hereby granted, free of charge, to any person obtaining a
copy of this software and associated documentation files (the
"Software"), to deal in the Software without restriction, including
without limitation the rights to use, copy, modify, merge, publish,
distribute, sublicense, and/or sell copies of the Software, and to
permit persons to whom the Software is furnished to do so, subject to
the following conditions:

The above copyright notice and this permission notice shall be included
in all copies or substantial portions of the Software.

THE SOFTWARE IS PROVIDED "AS IS", WITHOUT WARRANTY OF ANY KIND, EXPRESS
OR IMPLIED, INCLUDING BUT NOT LIMITED TO THE WARRANTIES OF
MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE AND NONINFRINGEMENT.
IN NO EVENT SHALL THE AUTHORS OR COPYRIGHT HOLDERS BE LIABLE FOR ANY
CLAIM, DAMAGES OR OTHER LIABILITY, WHETHER IN AN ACTION OF CONTRACT,
TORT OR OTHERWISE, ARISING FROM, OUT OF OR IN CONNECTION WITH THE
SOFTWARE OR THE USE OR OTHER DEALINGS IN THE SOFTWARE.
```

1.11 Optimization

class pyoptsparse.pyOpt_optimization.**Optimization**(*name*, *objFun*, *comm=None*, *sens=None*)

The main purpose of this class is to describe the structure and potentially, sparsity pattern of an optimization problem.

Parameters

name
[str] Name given to optimization problem.

objFun
[Python function handle] Function handle used to evaluate the objective function.

comm
[MPI intra communication] The communicator this problem will be solved on. This is required for both analysis when the objective is computed in parallel as well as to use the internal parallel gradient computations. Defaults to MPI.COMM_WORLD if not given.

sens
[str or python Function.] Specify method to compute sensitivities.

addCon(*name*, **args*, ***kwargs*)

Convenience function. See addConGroup() for more information

addConGroup(*name*, *nCon*, *lower=None*, *upper=None*, *scale=1.0*, *linear=False*, *wrt=None*, *jac=None*)

Add a group of variables into a variable set. This is the main function used for adding variables to pyOptSparse.

Parameters

name
[str] Constraint name. All names given to constraints must be unique

nCon
[int] The number of constraints in this group

lower
[scalar or array] The lower bound(s) for the constraint. If it is a scalar, it is applied to all nCon constraints. If it is an array, the array must be the same length as nCon.

upper
[scalar or array] The upper bound(s) for the constraint. If it is a scalar, it is applied to all nCon constraints. If it is an array, the array must be the same length as nCon.

scale
[scalar or array] A scaling factor for the constraint. It is generally advisable to have most optimization constraint around the same order of magnitude.

linear
[bool] Flag to specify if this constraint is linear. If the constraint is linear, both the `wrt` and `jac` keyword arguments must be given to specify the constant portion of the constraint Jacobian.

The intercept term of linear constraints must be supplied as part of the bound information. The linear constraint $g_L \leq Ax + b \leq g_U$ is equivalent to $g_L - b \leq Ax \leq g_U - b$, and pyOptSparse requires the latter form. In this case, the arguments should be:

```
jac = {"dvName" : A, ...}, lower = gL - b, upper = gU - b
```

wrt
[iterable (list, set, OrderedDict, array etc)] 'wrt' stand for stands for 'With Respect To'. This specifies for what dvs have non-zero Jacobian values for this set of constraints. The order is not important.

jac
[dictionary] For linear and sparse non-linear constraints, the constraint Jacobian must be passed in. The structure of jac dictionary is as follows:

```
{'dvName1':matrix1, 'dvName2', matrix1, ...}
```

They keys of the Jacobian must correspond to the dvGroups given in the wrt keyword argument. The dimensions of each "chunk" of the constraint Jacobian must be consistent. For example, `matrix1` must have a shape of (nCon, nDvs) where nDVs is the total number of design variables in dvName1. `matrix1` may be a dense numpy array or it may be scipy sparse matrix. However, it is *HIGHLY* recommended that sparse constraints are supplied to pyOptSparse using the pyOptSparse's simplified sparse matrix format. The reason for this is that it is *impossible* for force scipy sparse matrices to keep a fixed sparsity pattern; if the sparsity pattern changes during an optimization, *IT WILL FAIL*.

The three simplified pyOptSparse sparse matrix formats are summarized below:

```
mat = {'coo':[row, col, data], 'shape':[nrow, ncols]} # A coo matrix
mat = {'csr':[rowp, colind, data], 'shape':[nrow, ncols]} # A csr␣
↪matrix
mat = {'coo':[colp, rowind, data], 'shape':[nrow, ncols]} # A csc␣
↪matrix
```

Note that for nonlinear constraints (linear=False), the values themselves in the matrices in jac do not matter, but the sparsity structure **does** matter. It is imperative that entries that will at some point have non-zero entries have non-zero entries in jac argument. That is, we do not let the sparsity structure of the Jacobian change throughout the optimization. This stipulation is automatically checked internally.

`addObj`(*name*, **args*, ***kwargs*)

Add Objective into Objectives Set

`addVar`(*name*, **args*, ***kwargs*)

This is a convenience function. It simply calls addVarGroup() with nVars=1. Variables added with addVar() are returned as *scalars*.

`addVarGroup`(*name*, *nVars*, *varType='c'*, *value=0.0*, *lower=None*, *upper=None*, *scale=1.0*, *offset=0.0*, *choices=[]*, ***kwargs*)

Add a group of variables into a variable set. This is the main function used for adding variables to pyOptSparse.

Parameters

name
[str] Name of variable group. This name should be unique across all the design variable groups

nVars
[int] Number of design variables in this group.

varType
[str.] String representing the type of variable. Suitable values for type are: 'c' for continuous variables, 'i' for integer values and 'd' for discrete selection.

value
[scalar or array.] Starting value for design variables. If it is a a scalar, the same value is applied to all 'nVars' variables. Otherwise, it must be iterable object with length equal to 'nVars'.

lower
[scalar or array.] Lower bound of variables. Scalar/array usage is the same as value keyword

upper
[scalar or array.] Upper bound of variables. Scalar/array usage is the same as value keyword

scale
[scalar or array. Define a user supplied scaling] variable for the design variable group. This is often necessary when design variables of widely varying magnitudes are used within the same optimization. Scalar/array usage is the same as value keyword.

offset
[scalar or array. Define a user supplied offset] variable for the design variable group. This is often necessary when design variable has a large magnitude, but only changes a little about this value.

choices
[list] Specify a list of choices for discrete design variables

Notes

Calling addVar() and addVarGroup(..., nVars=1, ...) are **NOT** equivalent! The variable added with addVar() will be returned as scalar, while variable returned from addVarGroup will be an array of length 1.

It is recommended that the addVar() and addVarGroup() calls follow the examples above by including all the keyword arguments. This make it very clear the intent of the script's author. The type, value, lower, upper and scale should be given for all variables even if the default value is used.

Examples

```
>>> # Add a single design variable 'alpha'
>>> optProb.addVar('alpha', varType='c', value=2.0, lower=0.0, upper=10.0,␣
↪scale=0.1)
>>> # Add 10 unscaled variables of 0.5 between 0 and 1 with name 'y'
>>> optProb.addVarGroup('y', varType='c', value=0.5, lower=0.0, upper=1.0,␣
↪scale=1.0)
```

checkConName(*conName*)

Check if the desired constraint name has already been added. If it is has already been added, return a mangled name (a number appended) that *is* valid. This is intended to be used by classes that automatically add constraints to pyOptSparse.

Parameters

conName
[str] Constraint name to check validity on

Returns

validName
[str] A valid constraint name. May be the same as conName it that was, in fact, a valid name.

checkVarName(*varName*)

Check if the desired variable name varName if has already been added. If it is has already been added, return a mangled name (a number appended) that *is* valid. This is intended to be used by classes that automatically add variables to pyOptSparse

Parameters

varName
[str] Variable name to check validity on

Returns

validName
[str] A valid variable name. May be the same as varName it that was, in fact, a valid name.

delVar(*name*)

Delete a variable or variable group

Parameters

name
[str] Name of variable or variable group to remove

`evaluateLinearConstraints`(*x*, *fcon*)

This function is required for optimizers that do not explicitly treat the linear constraints. For those optimizers, we will evaluate the linear constraints here. We place the values of the linear constraints in the fcon dictionary such that it appears as if the user evaluated these constraints.

Parameters

x
[array] This must be the processed x-vector from the optimizer

fcon
[dict] Dictionary of the constraints. The linear constraints are to be added to this dictionary.

`finalize`()

This is a helper function which will only finalize the optProb if it's not already finalized.

`getDVConIndex`(*startIndex=1*, *printIndex=True*)

Return the index of a scalar DV/constraint, or the beginning and end index (inclusive) of a DV/constraint array. This is useful for looking at SNOPT gradient check output, and the default startIndex=1 is for that purpose

`getDVs`()

Return a dictionary of the design variables. In most common usage, this function is not required.

Returns

outDVs
[dict] The dictionary of variables. This is the same as 'x' that would be used to call the user objective function.

`getOrdering`(*conOrder*, *oneSided*, *noEquality=False*)

Internal function that is used to produce a index list that reorders the constraints the way a particular optimizer needs.

Parameters

conOrder
[list] This must contain the following 4 strings: 'ni', 'li', 'ne', 'le' which stand for nonlinear inequality, linear inequality, nonlinear equality and linear equality. This defines the order that the optimizer wants the constraints

oneSided
[bool] Flag to do all constraints as one-sided instead of two sided. Most optimizers need this but some can deal with the two-sided constraints properly (snopt and ipopt for example)

noEquality
[bool] Flag to split equality constraints into two inequality constraints. Some optimizers (CONMIN for example) can't do equality constraints explicitly.

`printSparsity`(*verticalPrint=False*)

This function prints an (ASCII) visualization of the Jacobian sparsity structure. This helps the user visualize what pyOptSparse has been given and helps ensure it is what the user expected. It is highly recommended this function be called before the start of every optimization to verify the optimization problem setup.

Parameters

verticalPrint
[bool] True if the design variable names in the header should be printed vertically instead of horizontally. If true, this will make the constraint Jacobian print out more narrow and taller.

> **Warning:** This function is **collective** on the optProb comm. It is therefore necessary to call this function on **all** processors of the optProb comm.

`processConstraintJacobian`(*gcon*)

This generic function is used to assemble the entire constraint Jacobian. The order of the constraint Jacobian is in 'natural' ordering, that is the order the constraints have been added (mostly; since it can be different when constraints are added on different processors).

The input is gcon, which is dict or an array. The array format should only be used when the pyOpt_gradient class is used since this results in a dense (and correctly oriented) Jacobian. The user should NEVER return a dense Jacobian since this extremely fickle and easy to break. The dict 'gcon' must contain only the non-linear constraints Jacobians; the linear ones will be added automatically.

Parameters

gcon
[array or dict] Constraint gradients. Either a complete 2D array or a nested dictionary of gradients given with respect to the variables.

Returns

gcon
[dict with csr data] Return the Jacobian in a sparse csr format. can be easily converted to csc, coo or dense format as required by individual optimizers

> **Warning:** This function should not need to be called by the user

`processContoDict`(*fcon_in*, *scaled=True*, *dtype='d'*, *natural=False*, *multipliers=False*)

Parameters

fcon_in
[array] Array of constraint values to be converted into a dictionary

scaled
[bool] Flag specifying if the returned array should be scaled by the pyOpt scaling. The only type this is not true is when the automatic derivatives are used

dtype
[str] String specifying the data type to return. Normally this is 'd' for a float. The complex-step derivative computations will call this function with 'D' to ensure that the complex perturbations pass through correctly.

natural
[bool] Flag to specify if the input data is in the natural ordering. This is only used when computing gradient automatically with FD/CS.

multipliers
[bool] Flag that indicates whether this deprocessing is for the multipliers or the constraint values. In the case of multipliers, no constraint offset should be applied.

> **Warning:** This function should not need to be called by the user

`processContoVec`(*fcon_in*, *scaled=True*, *dtype='d'*, *natural=False*)

Parameters

fcon_in
[dict] Dictionary of constraint values

scaled
[bool] Flag specifying if the returned array should be scaled by the pyOpt scaling. The only type this is not true is when the automatic derivatives are used

dtype
[str] String specifying the data type to return. Normally this is 'd' for a float. The complex-step derivative computations will call this function with 'D' to ensure that the complex perturbations pass through correctly.

natural
[bool] Flag to specify if the data should be returned in the natural ordering. This is only used when computing gradient automatically with FD/CS.

Warning: This function should not need to be called by the user

`processObjectiveGradient`(*funcsSens*)

This generic function is used to assemble the objective gradient(s)

Parameters

funcsSens
[dict] Dictionary of all function gradients. Just extract the objective(s) we need here.

Warning: This function should not need to be called by the user

`processObjtoDict`(*fobj_in*, *scaled=True*)

This function converts the objective in array form to the corresponding dictionary form.

Parameters

fobj_in
[float or ndarray] The objective in array format. In the case of a single objective, a float can also be accepted.

scaled
[bool] Flag specifying if the returned dictionary should be scaled by the pyOpt scaling.

Returns

fobj
[dictionary] The dictionary form of fobj_in, which is just a key:value pair for each objective.

`processObjtoVec`(*funcs*, *scaled=True*)

This is currently just a stub-function. It is here since it the future we may have to deal with multiple objectives so this function will deal with that

Parameters

funcs
[dictionary of function values]

Returns

obj
[float or array] Processed objective(s).

> **Warning:** This function should not need to be called by the user

`processXtoDict`(*x*)

Take the flattened array of variables in 'x' and return a dictionary of variables keyed on the name of each variable.

Parameters

x
[array] Flattened array from optimizer

> **Warning:** This function should not need to be called by the user

`processXtoVec`(*x*)

Take the dictionary form of x and convert back to flattened array.

Parameters

x
[dict] Dictionary form of variables

Returns

x_array
[array] Flattened array of variables

> **Warning:** This function should not need to be called by the user

`setDVs`(*inDVs*)

Set one or more groups of design variables from a dictionary. In most common usage, this function is not required.

Parameters

inDVs
[dict] The dictionary of variables. The keys are the names of the variable groups, and the values are the desired design variable values for each variable group.

`setDVsFromHistory`(*histFile*, *key=None*)

Set optimization variables from a previous optimization. This is like a cold start, but some variables may have been added or removed from the previous optimization. This will try to set all variables it can.

Parameters

histFile
[str] Filename of the history file to read

key
[str] Key of the history file to use for the x values. The default is None which will use the last x-value stored in the dictionary.

1.12 Optimizer

class pyoptsparse.pyOpt_optimizer.**Optimizer**(**args*, ***kwargs*)

This is the base optimizer class that all optimizers inherit from. We define common methods here to avoid code duplication.

Parameters

name
[str] Optimizer name

category
[str] Typically local or global

defaultOptions
[dictionary] A dictionary containing the default options

informs
[dict] Dictionary of the inform codes

getInform(*infocode=None*)

Get optimizer result inform code at exit

Parameters

infocode
[int] Integer information code

getOption(*name*)

Return the optimizer option value for name

Parameters

name
[str] name of option for which to retrieve value

Returns

value
[varies] value of option for 'name'

setOption(*name*, *value=None*)

Generic routine for all option setting. The routine does error checking on the type of the value.

Parameters

name
[str] Name of the option to set

value
[varies] Variable value to set.

pyoptsparse.pyOpt_optimizer.**OPT**(*optName*, **args*, ***kwargs*)

This is a simple utility function that enables creating an optimizer based on the 'optName' string. This can be useful for doing optimization studies with respect to optimizer since you don't need massive if-statements.

Parameters

optName
[str] String identifying the optimizer to create

***args, **kwargs**
[varies] Passed to optimizer creation.

Returns

opt
[pyOpt_optimizer inherited optimizer] The desired optimizer

1.13 Gradient

class pyoptsparse.pyOpt_gradient.**Gradient**(*optProb*, *sensType*, *sensStep=None*, *sensMode=''*, *comm=None*)

Gradient class for automatically computing gradients with finite difference or complex step.

Parameters

optProb
[Optimization instance] This is the complete description of the optimization problem.

sensType
[str]

- `FD` for forward difference
- `CD` for central difference
- `FDR` for forward difference with relative step size
- `CDR` for central difference with relative step size
- `CS` for complex step

sensStep
[float] Step size to use for differencing

sensMode
[str] Flag to compute gradients in parallel.

1.14 Variable

class pyoptsparse.pyOpt_variable.**Variable**(*name*, *varType*, *value*, *lower*, *upper*, *scale*, *offset*, *scalar=False*, *choices=[]*)

This class holds the representation of a single pyOptSparse variable

See also:

pyoptsparse.pyOpt_optimization.Optimization.addVarGroup
for the full documentation

1.15 Constraint

class pyoptsparse.pyOpt_constraint.**Constraint**(*name*, *nCon*, *linear*, *wrt*, *jac*, *lower*, *upper*, *scale*)

This class holds the representation of a pyOptSparse constraint group

See also:

pyoptsparse.pyOpt_optimization.Optimization.addConGroup
for the full documentation

finalize(*variables*, *dvOffset*, *index*)

After the design variables have been finalized and the order is known we can check the constraint for consistency.

Parameters

variables
[OrderedDict] The pyOpt variable list after they have been finalized.

dvOffset
[dict] Design variable offsets from pyOpt_optimization

index
[int] The starting index of this constraint in natural order

Warning: This function should not need to be called by the user

1.16 Objective

class pyoptsparse.pyOpt_objective.**Objective**(*name*, *scale=1.0*)

This class holds the representation of a pyOptSparse objective.

Parameters

name
[str] Name of this objective

scale
[float] Scaling factor for objective. This does not change the actual optimization problem, but may be used to give a more human-meaningful value

See also:

pyoptsparse.pyOpt_optimization.Optimization.addObj
for the full documentation

1.17 Solution

class pyoptsparse.pyOpt_solution.**Solution**(*optProb*, *xStar*, *fStar*, *lambdaStar*, *optInform*, *info*)

This class is used to describe the solution of an optimization problem. This class inherits from Optimization which enables a solution to be used as an input to a subsequent optimization problem.

Parameters

optProb
[Optimization problem class] Optimization problem used to create solution

xStar
[dict] The final design variables

fStar
[dict] The final objective(s)

lambdaStar
[dict] The final Lagrange multipliers

optInform
[int] The inform code returned by the optimizer

info
[dict] A dictionary containing timing and call counter info to be stored in the Solution object.

1.18 History

Suppose we have an optimization problem with one DV group `xvars`, one constraint `con`, and the objective is called `obj`. In this case, the history file would have the following layout:

```
├── metadata
│   ├── optName
│   ├── optOptions
│   ├── nprocs
│   ├── startTime
│   ├── endTime
│   ├── optTime
│   ├── optimizer
│   ├── version
│   └── optVersion
├── optProb
├── varInfo
│   └── xvars
│       ├── lower
│       ├── upper
│       └── scale
├── conInfo
│   └── con
│       ├── lower
│       ├── upper
│       └── scale
├── objInfo
│   └── obj
```

(continues on next page)

(continued from previous page)

```
│       └── scale
├── 0
│   ├── xuser
│   │   └── xvars
│   ├── funcs
│   │   ├── obj
│   │   └── con
│   ├── iter
│   ├── fail
│   ├── isMajor
│   └── time
├── 1
│   ├── xuser
│   │   └── xvars
│   ├── funcsSens
│   │   ├── obj
│   │   │   └── xvars
│   │   └── con
│   │       └── xvars
│   ├── iter
│   ├── fail
│   ├── isMajor
│   └── time
└── last
```

The main optimization history is indexed via call counters, in this example `0` and `1`. Note that they do not match the major/minor iterations of a given optimizer, since gradient evaluations are stored separate from the function evaluation.

For SNOPT, a number of other values can be requested and stored in each major iteration, such as the feasibility and optimality from the SNOPT print out file.

1.18.1 API

class pyoptsparse.pyOpt_history.**History**(*fileName*, *optProb=None*, *temp=False*, *flag='r'*)

This class is essentially a thin wrapper around a SqliteDict dictionary to facilitate operations with pyOptSparse

Parameters

fileName
[str] File name for history file

optProb
[pyOpt_Optimization] The optimization object

temp
[bool] Flag to signify that the file should be deleted after it is closed

flag
[str] String specifying the mode. Similar to what was used in shelve. `n` for a new database and `r` to read an existing one.

close()

Close the underlying database. This should only be used in write mode. In read mode, we close the db during initialization.

`getCallCounters()`

Returns a list of all call counters stored in the history file.

Returns

list
a list of strings, each entry being a call counter.

`getConInfo(key=None)`

Returns the *ConInfo*, for all keys by default. A *key* parameter can also be supplied, to retrieve *ConInfo* corresponding to specific constraints.

Parameters

key
[str or list of str, optional] Specifies for which constraint to extract *ConInfo*.

Returns

dict
A dictionary containing ConInfo. For a single key, the return is one level deeper.

`getConNames()`

Returns the names of constraints.

Returns

list of str
A list containing the names of constraints.

`getDVInfo(key=None)`

Returns the *DVInfo*, for all keys by default. A *key* parameter can also be supplied, to retrieve *DVInfo* corresponding to specific DVs.

Parameters

key
[str or list of str, optional] Specifies for which DV to extract *DVInfo*.

Returns

dict
A dictionary containing DVInfo. For a single key, the return is one level deeper.

`getDVNames()`

Returns the names of the DVs.

Returns

list of str
A list containing the names of DVs.

`getExtraFuncsNames()`

Returns extra funcs names. These are extra key: value pairs stored in the `funcs` dictionary for each iteration, which are not used by the optimizer.

Returns

list of str
A list containing the names of extra funcs keys.

`getIterKeys()`

Returns the keys available at each optimization iteration. This function is useful for inspecting the history file, to determine what information is saved at each iteration.

Returns

list of str
A list containing the names of keys stored at each optimization iteration.

`getMetadata()`

Returns a copy of the metadata stored in the history file.

Returns

dict
A dictionary containing the metadata.

`getObjInfo(`*key=None*`)`

Returns the *ObjInfo*, for all keys by default. A *key* parameter can also be supplied, to retrieve *ObjInfo* corresponding to specific keys.

Parameters

key
[str or list of str, optional] Specifies for which obj to extract *ObjInfo*.

Returns

dict
A dictionary containing ObjInfo. For a single key, the return is one level deeper.

Notes

Recall that for the sake of generality, pyOptSparse allows for multiple objectives to be added. This feature is not used currently, but does make *ObjInfo* follow the same structure as *ConInfo* and *DVInfo*. Because of this, it is recommended that this function be accessed using the optional *key* argument.

`getObjNames()`

Returns the names of the objectives.

Returns

list of str
A list containing the names of objectives.

Notes

Recall that for the sake of generality, pyOptSparse allows for multiple objectives to be added. This feature is not used currently, but does make *ObjNames* follow the same structure as *ConNames* and *DVNames*.

`getOptProb()`

Returns a copy of the optProb associated with the optimization.

Returns

optProb
[pyOpt_optimization object] The optProb associated with the optimization. This is taken from the history file, and therefore has the `comm` and `objFun` fields removed.

getValues(*names=None*, *callCounters=None*, *major=True*, *scale=False*, *stack=False*, *allowSens=False*)

Parses an existing history file and returns a data dictionary used to post-process optimization results, containing the requested optimization iteration history.

Parameters

names
[list or str] the values of interest, can be the name of any DV, objective or constraint, or a list of them. If None, all values are returned. This includes the DVs, funcs, and any values stored by the optimizer.

callCounters
[list of ints, can also contain 'last'] a list of callCounters to extract information from. If the callCounter is invalid, i.e. outside the range or is a funcsSens evaluation, then it is skipped. 'last' represents the last major iteration. If None, values from all callCounters are returned.

major
[bool] flag to specify whether to include only major iterations.

scale
[bool] flag to specify whether to apply scaling for the values. True means to return values that are scaled the same way as the actual optimization.

stack
[bool] flag to specify whether the DV should be stacked into a single numpy array with the key *xuser*, or retain their separate DVGroups.

allowSens: bool
flag to specify whether gradient evaluation iterations are allowed. If true, it is up to the user to ensure that the callCounters specified contain the information requested.

Returns

dict
a dictionary containing the information requested. The keys of the dictionary correspond to the *names* requested. Each value is a numpy array with the first dimension equal to the number of callCounters requested.

Notes

Regardless of the major flag, failed function evaluations are not returned.

Examples

First we can request DV history over all major iterations:

```
>>> hist.getValues(names='xvars', major=True)
{'xvars': array([[-2.00000000e+00,  1.00000000e+00],
    [-1.00000000e+00,  9.00000000e-01],
    [-5.00305827e-17,  4.21052632e-01],
    [ 1.73666171e-06,  4.21049838e-01],
    [ 9.08477459e-06,  4.21045261e-01],
    [ 5.00000000e-01,  2.84786405e-01],
    [ 5.00000000e-01,  5.57279939e-01],
    [ 5.00000000e-01,  2.00000000e+00]])}
```

Next we can look at DV and optimality for the first and last iteration only:

```
>>> hist.getValues(names=['xvars','optimality'],callCounters=[0,'last'])
{'optimality': array([1.27895528, 0. ]),
'xvars': array([[-2. , 1. ],
                [ 0.5, 2. ]])}
```

pointExists(*callCounter*)

Determine if callCounter is in the database

Parameters

callCounter
[int or str of int]

Returns

bool
True if the callCounter exists in the history file. False otherwise.

read(*key*)

Read data for an arbitrary key. Returns None if key is not found. If passing in a callCounter, it should be verified by calling pointExists() first.

Parameters

key
[str or int] generic key[str] or callCounter[int]

Returns

dict
The value corresponding to *key* is returned. If the key is not found, *None* is returned instead.

write(*callCounter*, *data*)

This is the main to write data. Basically, we just pass in the callCounter, the integer forming the key, and a dictionary which will be written to the key

Parameters

callCounter
[int] the callCounter to write to

data
[dict] the dictionary corresponding to the callCounter

writeData(*key*, *data*)

Write arbitrary *key:data* value to db.

Parameters

key
[str] The key to be added to the history file

data
The data corresponding to the key. It can be anything as long as it is serializable in *sqlitedict*.

1.19 Utils

pyOptSparse_utils holds a minimal set of sparse-matrix type routines for pyOptSparse. This is designed to replace the SciPy sparse matrix formats, which have no way to enforce a constant sparsity structure as required by the optimizers. We use a very simple dictionary format to represent the three most common forms of sparse matrices:

```
mat = {'coo':[row,  col,    data], 'shape':[nrow, ncols]} # A coo matrix
mat = {'csr':[rowp, colind, data], 'shape':[nrow, ncols]} # A csr matrix
mat = {'csc':[colp, rowind, data], 'shape':[nrow, ncols]} # A csc matrix
```

pyoptsparse.pyOpt_utils.**convertToCOO**(*mat*)

Take a pyoptsparse sparse matrix definition of a COO, CSR or CSC matrix or numpy array or scipy sparse matrix and return the same matrix in COO format.

Parameters

mat
[dict or numpy array] A sparse matrix representation or numpy array

Returns

newMat
[dict] A coo representation of the same matrix

pyoptsparse.pyOpt_utils.**convertToCSC**(*mat*)

Take a pyoptsparse sparse matrix definition of a COO, CSR or CSC matrix or numpy array and return the same matrix in CSR format

Parameters

mat
[dict or numpy array] A sparse matrix representation or numpy array

Returns

newMat
[dict] A coo representation of the same matrix

pyoptsparse.pyOpt_utils.**convertToCSR**(*mat*)

Take a pyoptsparse sparse matrix definition of a COO, CSR or CSC matrix or numpy array and return the same matrix in CSR format

Parameters

mat
[dict or numpy array] A sparse matrix representation or numpy array

Returns

newMat
[dict] A coo representation of the same matrix

pyoptsparse.pyOpt_utils.**convertToDense**(*mat*)

Take a pyopsparse sparse matrix definition and convert back to a dense format. This is typically the final step for optimizers with dense constraint jacibians.

Parameters

mat
[dict] A sparse matrix representation. Should be in CSR format for best efficiency

Returns

newMat
[array] A dense numpy array of the same matrix

pyoptsparse.pyOpt_utils.**extractRows**(*mat*, *indices*)

Extract the rows defined by 'indices' and return a new CSR matrix.

Parameters

mat
[dict] pyoptsparse matrix CSR format

indices
[list/array of integer] The rows the user wants to extract

Returns

newMat
[dic] pyoptsparse CSR matrix

pyoptsparse.pyOpt_utils.**mapToCSC**(*mat*)

Given a pyoptsparse matrix definition, return a tuple containing a map of the matrix to the CSC format.

Parameters

mat
[dict] A sparse matrix representation.

Returns

tup
[tuple of numpy arrays]

tup[0]
[numpy array (size=nnz)] An array that holds the row index of each element in the CSC representation of the data.

tup[1]
[numpy array (size=num_cols+1)] An array that holds the indices in the CSC representation and data at which each column begins. The last index of contains the number of nonzero elements in the sparse array.

tup[2]
[numpy array] An indexing array which maps the elements in the data array to elements in the CSC data array.

pyoptsparse.pyOpt_utils.**mapToCSR**(*mat*)

Given a pyoptsparse matrix definition, return a tuple containing a map of the matrix to the CSR format.

Parameters

mat
[dict] A sparse matrix representation.

Returns

tup
[tuple of numpy arrays]

tup[0]
[numpy array (size=num_rows+1)] An array that holds the indices in col_idx and data at which each row begins. The last index of contains the number of nonzero elements in the sparse array.

tup[1]
[numpy array (size=nnz)] An array of the column indices of each element in data.

tup[2]
[numpy array (size=nnz)] An indexing array which maps the elements in the data array to elements in the CSR data array.

pyoptsparse.pyOpt_utils.**scaleColumns**(*mat*, *factor*)

d= Scale the columns of the matrix. Must be CSR format

pyoptsparse.pyOpt_utils.**scaleRows**(*mat*, *factor*)

Scale the rows of the matrix. Must be CSR format

1.20 SNOPT

SNOPT is a sparse nonlinear optimizer that is particularly useful for solving large-scale constrained problems with smooth objective functions and constraints. The algorithm consists of a sequential quadratic programming (SQP) algorithm that uses a smooth augmented Lagrangian merit function, while making explicit provision for infeasibility in the original problem and in the quadratic programming subproblems. The Hessian of the Lagrangian is approximated using the BFGS quasi-Newton update.

1.20.1 Installation

SNOPT is available for purchase here. Upon purchase, you should receive a zip file. Within the zip file, there is a folder called `src`. To use SNOPT with pyoptsparse, paste all files from `src` except snopth.f into `pyoptsparse/pySNOPT/source`.

From v2.0 onwards, only SNOPT v7.7.x is officially supported. To use pyOptSparse with previous versions of SNOPT, please checkout release v1.2. We currently test v7.7.7 and v7.7.1.

1.20.2 Options

Please refer to the SNOPT user manual for a complete listing of options and their default values. The following are a list of

- options which have values changed from the defaults within SNOPT
- options unique to pyOptSparse, implemented in the Python wrapper and not found in SNOPT

Table 1: SNOPT Default Options

Name	Type	Default value	Description
`iPrint`	int	18	Print File Output Unit (override internally in snopt?)
`iSumm`	int	19	Summary File Output Unit (override internally in snopt?)
`Print file`	str	`SNOPT_print.out`	Print file name
`Summary file`	str	`SNOPT_summary.out`	Summary file name
`Minor print level`	int	0	Minor iterations print level
`Problem Type`	str	`Minimize`	This specifies the problem type for SNOPT. • `Minimize`: minimization problem. • `Maximize`: maximization problem. • `Feasible point`: compute a feasible point only.
`Start`	str	`Cold`	This value is directly passed to the SNOPT kernel, and will be overwritten if another option, e.g. `Cold start` is supplied, in accordance with SNOPT options precedence. • `Cold`: Cold start • `Hot`: Hot start
`Derivative level`	int	3	The SNOPT derivative level. Only "3" is tested, where all derivatives are provided to SNOPT.
`Iterations limit`	int	10000000	The limit on the total number of minor iterations, summed over all major iterations. This option is set to a very large number to prevent premature termination of SNOPT.
`Minor iterations limit`	int	10000	The limit on the number of minor iterations *for each major iteration*. This option is set to a very large number to prevent premature termination of SNOPT.
`Proximal iterations limit`	int	10000	The iterations limit for solving the proximal point problem. We set this by default to a very large value in order to fully solve the proximal point problem to optimality
`Total character`	int	None	The total character

1.20.3 Informs

Table 2: SNOPT Informs

Code	Description
0	finished successfully
1	optimality conditions satisfied
2	feasible point found
3	requested accuracy could not be achieved
4	weak QP minimizer
10	the problem appears to be infeasible
11	infeasible linear constraints
12	infeasible linear equalities
13	nonlinear infeasibilities minimized
14	infeasibilities minimized
15	infeasible linear constraints in QP subproblem
20	the problem appears to be unbounded
21	unbounded objective
22	constraint violation limit reached
30	resource limit error
31	iteration limit reached
32	major iteration limit reached
33	the superbasics limit is too small
40	terminated after numerical difficulties
41	current point cannot be improved
42	singular basis
43	cannot satisfy the general constraints
44	ill-conditioned null-space basis
50	error in the user-supplied functions
51	incorrect objective derivatives
52	incorrect constraint derivatives
53	the QP Hessian is indefinite
54	incorrect second derivatives
55	incorrect derivatives
56	irregular or badly scaled problem functions
60	undefined user-supplied functions
61	undefined function at the first feasible point
62	undefined function at the initial point
63	unable to proceed into undefined region
70	user requested termination
71	terminated during function evaluation
72	terminated during constraint evaluation
73	terminated during objective evaluation
74	terminated from monitor routine
80	insufficient storage allocated
81	work arrays must have at least 500 elements
82	not enough character storage
83	not enough integer storage
84	not enough real storage
90	input arguments out of range
91	invalid input argument

continues on next page

Table 2 – continued from previous page

Code	Description
92	basis file dimensions do not match this problem
93	the QP Hessian is indefinite
100	finished successfully
101	SPECS file read
102	Jacobian structure estimated
103	MPS file read
104	memory requirements estimated
105	user-supplied derivatives appear to be correct
106	no derivatives were checked
107	some SPECS keywords were not recognized
110	errors while processing MPS data
111	no MPS file specified
112	problem-size estimates too small
113	fatal error in the MPS file
120	errors while estimating Jacobian structure
121	cannot find Jacobian structure at given point
130	fatal errors while reading the SP
131	no SPECS file (iSpecs le 0 or iSpecs gt 99)
132	End-of-file while looking for a BEGIN
133	End-of-file while reading SPECS file
134	ENDRUN found before any valid SPECS
140	system error
141	wrong no of basic variables
142	error in basis package

1.20.4 API

class pyoptsparse.pySNOPT.pySNOPT.**SNOPT**(**args*, ***kwargs*)

SNOPT Optimizer Class - Inherited from Optimizer Abstract Class

SNOPT Optimizer Class Initialization

__call__(*optProb*, *sens=None*, *sensStep=None*, *sensMode=None*, *storeHistory=None*, *hotStart=None*, *storeSens=True*, *timeLimit=None*, *restartDict=None*)

This is the main routine used to solve the optimization problem.

Parameters

optProb
[Optimization or Solution class instance] This is the complete description of the optimization problem to be solved by the optimizer

sens
[str or python Function.] Specify method to compute sensitivities. The default is None which will use SNOPT's own finite differences which are vastly superior to the pyOptSparse implementation. To explicitly use pyOptSparse gradient class to do the derivatives with finite differences use 'FD'. 'sens' may also be 'CS' which will cause pyOptSpare to compute the derivatives using the complex step method. Finally, 'sens' may be a python function handle which is expected to compute the sensitivities directly. For expensive function evaluations and/or problems with large numbers of design variables this is the preferred method.

sensStep
[float] Set the step size to use for design variables. Defaults to 1e-6 when sens is 'FD' and 1e-40j when sens is 'CS'.

sensMode
[str] Use 'pgc' for parallel gradient computations. Only available with mpi4py and each objective evaluation is otherwise serial

storeHistory
[str] File name of the history file into which the history of this optimization will be stored

hotStart
[str] File name of the history file to "replay" for the optimization. The optimization problem used to generate the history file specified in 'hotStart' must be **IDENTICAL** to the currently supplied 'optProb'. By identical we mean, **EVERY SINGLE PARAMETER MUST BE IDENTICAL**. As soon as he requested evaluation point from SNOPT does not match the history, function and gradient evaluations revert back to normal evaluations.

storeSens
[bool] Flag specifying if sensitivities are to be stored in hist. This is necessary for hotstarting only.

timeLimit
[float] Specify the maximum amount of time for optimizer to run. Must be in seconds. This can be useful on queue systems when you want an optimization to cleanly finish before the job runs out of time.

restartDict
[dict] A dictionary containing the necessary information for hot-starting SNOPT. This is typically the same dictionary returned by this function on a previous invocation.

Returns

sol
[Solution object] The optimization solution

restartDict
[dict] If 'Return work arrays' is True, a dictionary of arrays is also returned

1.21 IPOPT

IPOPT (Interior Point OPTimizer) is an open source interior point optimizer, designed for large-scale nonlinear optimization. The source code can be found here. The latest version we support is 3.13.2.

1.21.1 Installation

IPOPT must be installed separately, then linked to pyOptSparse when building. For the full installation instructions, please see their documentation. OpenMDAO also has a very helpful script which can be used to install IPOPT with other linear solvers. Here we explain a basic setup using MUMPS as the linear solver, together with METIS adapted from the OpenMDAO script.

1. Download the tarball and extract it to `$IPOPT_DIR` which could be set to for example `$HOME/packages/Ipopt`.
2. Install METIS, which can be used to improve the performance of the MUMPS linear solver.

```
# build METIS
cd $IPOPT_DIR
git clone https://github.com/coin-or-tools/ThirdParty-Metis.git
cd ThirdParty-Metis
./get.Metis
./configure --prefix=$IPOPT_DIR
make
make install
```

3. Install MUMPS

```
# build MUMPS
cd $IPOPT_DIR
git clone https://github.com/coin-or-tools/ThirdParty-Mumps.git
cd ThirdParty-Mumps
./get.Mumps
./configure --with-metis --with-metis-lflags="-L${IPOPT_DIR}/lib -lcoinmetis" \
     --with-metis-cflags="-I${IPOPT_DIR}/include -I${IPOPT_DIR}/include/coin-or -I$
↪{IPOPT_DIR}/include/coin-or/metis" \
     --prefix=$IPOPT_DIR CFLAGS="-I${IPOPT_DIR}/include -I${IPOPT_DIR}/include/coin-
↪or -I${IPOPT_DIR}/include/coin-or/metis" \
     FCFLAGS="-I${IPOPT_DIR}/include -I${IPOPT_DIR}/include/coin-or -I${IPOPT_DIR}/
↪include/coin-or/metis"
make
make install
```

4. Build IPOPT

```
# build IPOPT
cd $IPOPT_DIR
mkdir build
cd build
../configure --prefix=${IPOPT_DIR} --disable-java --with-mumps --with-mumps-lflags=
↪"-L${IPOPT_DIR}/lib -lcoinmumps" \
     --with-mumps-cflags="-I${IPOPT_DIR}/include/coin-or/mumps"
make
make install
```

5. You must add the IPOPT library path to the `LD_LIBRARY_PATH` variable for things to work right. This could be done for example by adding the following to your `.bashrc`:

```
export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$IPOPT_DIR/lib
```

   Furthermore, the environment variable `$IPOPT_DIR` must be set correctly in order to link to pyOptSparse. Alternatively, you can manually define the variables `$IPOPT_LIB` and `$IPOPT_INC` for the lib and include paths separately.

6. Now clean build pyOptSparse. Verify that IPOPT works by running the relevant tests.

Note: To get IPOPT working with pyOptSparse when using another linear solver, several things must be changed.

1. The `setup.py` file located in `pyoptsparse/pyIPOPT` must be updated accordingly. In particular, the `libraries=` line must be changed to reflect the alternate linear solver. For example, for HSL you need to replace `coinmumps` and `coinmetis` with `coinhsl`.

2. The option `linear_solver` in the options dictionary must be changed. The default value can be changed in `pyIPOPT.py` so that this option does not need to be manually set in every run script.

1.21.2 Options

Please refer to the IPOPT website for complete listing of options. The following are the options which are set by default within pyOptSparse. All other options take the default value with IPOPT unless specified by the user.

Table 3: IPOPT Default Options

Name	Type	Default value	Description
`print_level`	int	0	Printing level
`file_print_level`	int	5	Printing level for the output file
`sb`	str	`yes`	This is an undocumented option which suppresses the IPOPT header from being printed to screen every time.
`print_user_options`	str	`yes`	Whether to print the user-modified options
`output_file`	str	`IPOPT.out`	The name of the output file from IPOPT
`linear_solver`	str	`mumps`	The linear solver used.

1.21.3 Informs

Table 4: IPOPT Informs

Code	Description
`0`	Solve Succeeded
`1`	Solved To Acceptable Level
`2`	Infeasible Problem Detected
`3`	Search Direction Becomes Too Small
`4`	Diverging Iterates
`5`	User Requested Stop
`6`	Feasible Point Found
`-1`	Maximum Iterations Exceeded
`-2`	Restoration Failed
`-3`	Error In Step Computation
`-4`	Maximum CpuTime Exceeded
`-10`	Not Enough Degrees Of Freedom
`-11`	Invalid Problem Definition
`-12`	Invalid Option
`-13`	Invalid Number Detected
`-100`	Unrecoverable Exception
`-101`	NonIpopt Exception Thrown
`-102`	Insufficient Memory
`-199`	Internal Error

1.21.4 API

class pyoptsparse.pyIPOPT.pyIPOPT.**IPOPT**(**args*, ***kwargs*)

IPOPT Optimizer Class - Inherited from Optimizer Abstract Class

IPOPT Optimizer Class Initialization

__call__(*optProb*, *sens=None*, *sensStep=None*, *sensMode=None*, *storeHistory=None*, *hotStart=None*, *storeSens=True*)

This is the main routine used to solve the optimization problem.

Parameters

optProb
[Optimization or Solution class instance] This is the complete description of the optimization problem to be solved by the optimizer

sens
[str or python Function.] Specifiy method to compute sensitivities. To explictly use pyOptSparse gradient class to do the derivatives with finite differenes use 'FD'. 'sens' may also be 'CS' which will cause pyOptSpare to compute the derivatives using the complex step method. Finally, 'sens' may be a python function handle which is expected to compute the sensitivities directly. For expensive function evaluations and/or problems with large numbers of design variables this is the preferred method.

sensStep
[float] Set the step size to use for design variables. Defaults to 1e-6 when sens is 'FD' and 1e-40j when sens is 'CS'.

sensMode
[str] Use 'pgc' for parallel gradient computations. Only available with mpi4py and each objective evaluation is otherwise serial

storeHistory
[str] File name of the history file into which the history of this optimization will be stored

hotStart
[str] File name of the history file to "replay" for the optimziation. The optimization problem used to generate the history file specified in 'hotStart' must be **IDENTICAL** to the currently supplied 'optProb'. By identical we mean, **EVERY SINGLE PARAMETER MUST BE IDENTICAL**. As soon as he requested evaluation point does not match the history, function and gradient evaluations revert back to normal evaluations.

storeSens
[bool] Flag sepcifying if sensitivities are to be stored in hist. This is necessay for hotstarting only.

1.22 SLSQP

SLSQP optimizer is a sequential least squares programming algorithm which uses the Han-Powell quasi-Newton method with a BFGS update of the B-matrix and an L1-test function in the step-length algorithm. The optimizer uses a slightly modified version of Lawson and Hanson's NNLS nonlinear least-squares solver.

The version provided is the original source code from 1991 by Dieter Kraft.

1.22.1 Options

Table 5: SLSQP Default Options

Name	Type	Default value	Description
ACC	float	1e-06	Convergence Accurancy
MAXIT	int	500	Maximum Iterations
IPRINT	int	1	Output Level (<0 - None, 0 - Screen, 1 - File)
IOUT	int	60	Output Unit Number
IFILE	str	SLSQP.out	Output File Name

1.22.2 Informs

Table 6: SLSQP Informs

Code	Description
-1	Gradient evaluation required (g & a)
0	Optimization terminated successfully.
1	Function evaluation required (f & c)
2	More equality constraints than independent variables
3	More than 3*n iterations in LSQ subproblem
4	Inequality constraints incompatible
5	Singular matrix E in LSQ subproblem
6	Singular matrix C in LSQ subproblem
7	Rank-deficient equality constraint subproblem HFTI
8	Positive directional derivative for linesearch
9	Iteration limit exceeded

1.22.3 API

class pyoptsparse.pySLSQP.pySLSQP.**SLSQP**(**args*, ***kwargs*)

SLSQP Optimizer Class - Inherited from Optimizer Abstract Class

This is the base optimizer class that all optimizers inherit from. We define common methods here to avoid code duplication.

Parameters

name
[str] Optimizer name

category
[str] Typically local or global

defaultOptions
[dictionary] A dictionary containing the default options

informs
[dict] Dictionary of the inform codes

__call__(*optProb*, *sens=None*, *sensStep=None*, *sensMode=None*, *storeHistory=None*, *hotStart=None*, *storeSens=True*)

This is the main routine used to solve the optimization problem.

Parameters

optProb
[Optimization or Solution class instance] This is the complete description of the optimization problem to be solved by the optimizer

sens
[str or python Function.] Specifiy method to compute sensitivities. To explictly use pyOptSparse gradient class to do the derivatives with finite differenes use 'FD'. 'sens' may also be 'CS' which will cause pyOptSpare to compute the derivatives using the complex step method. Finally, 'sens' may be a python function handle which is expected to compute the sensitivities directly. For expensive function evaluations and/or problems with large numbers of design variables this is the preferred method.

sensStep
[float] Set the step size to use for design variables. Defaults to 1e-6 when sens is 'FD' and 1e-40j when sens is 'CS'.

sensMode
[str] Use 'pgc' for parallel gradient computations. Only available with mpi4py and each objective evaluation is otherwise serial

storeHistory
[str] File name of the history file into which the history of this optimization will be stored

hotStart
[str] File name of the history file to "replay" for the optimziation. The optimization problem used to generate the history file specified in 'hotStart' must be **IDENTICAL** to the currently supplied 'optProb'. By identical we mean, **EVERY SINGLE PARAMETER MUST BE IDENTICAL**. As soon as he requested evaluation point from SLSQP does not match the history, function and gradient evaluations revert back to normal evaluations.

storeSens
[bool] Flag sepcifying if sensitivities are to be stored in hist. This is necessay for hotstarting only.

1.23 NLPQLP

NLPQLP is a sequential quadratic programming (SQP) method which solves problems with smooth continuously differentiable objective function and constraints. The algorithm uses a quadratic approximation of the Lagrangian function and a linearization of the constraints. To generate a search direction a quadratic subproblem is formulated and solved. The line search can be performed with respect to two alternative merit functions, and the Hessian approximation is updated by a modified BFGS formula.

NLPQLP is a proprietary software, which can be obtained here. The latest version supported is v4.2.2.

1.23.1 Options

Table 7: NLPQLP Default Options

Name	Type	Default value	Description
accuracy	float	1e-06	Convergence accuracy
accuracyQP	float	1e-14	Convergence accuracy for QP
stepMin	float	1e-06	Minimum step length
maxFun	int	20	Maximum Number of Function Calls During Line Search
maxIt	int	500	Maximum Number of Iterations
maxNM	int	1	Maximum stack size for non-monotone line search
rho	float	1.0	Factor scaling identify for IFAIL=2
iPrint	int	2	Output Level • 2: Major • 0: None • 1: Final • 3: Major/Minor • 4: Full
mode	int	0	Mode (0 - Normal Execution, 1 to 18 - See Manual)
iOut	int	6	Output Unit Number
lMerit	bool	True	Merit Function Type (True - L2 Augmented Penalty, False - L1 Penalty)
lQl	bool	False	QP Subproblem Solver (True - Quasi-Newton, False - Cholesky)
iFile	str	NLPQLP.out	Output File Name

1.23.2 Informs

Table 8: NLPQLP Informs

Code	Description
-2	Compute gradient values w.r.t. the variables stored in first column of X, and store them in DF and DG. Only derivatives for active constraints ACTIVE(J)=.TRUE. need to be computed.
-1	Compute objective fn and all constraint values subjectthe variables found in the first L columns of X, and store them in F and G.
0	The optimality conditions are satisfied.
1	The algorithm has been stopped after MAXIT iterations.
2	The algorithm computed an uphill search direction.
3	Underflow occurred when determining a new approximation matrix for the Hessian of the Lagrangian.
4	The line search could not be terminated successfully.
5	Length of a working array is too short. More detailed error information is obtained with IPRINT>0
6	There are false dimensions, for example M>MMAX, N>=NMAX, or MNN2<>M+N+N+2.
7	The search direction is close to zero, but the current iterate is still infeasible.
8	The starting point violates a lower or upper bound.
9	Wrong input parameter, i.e., MODE, LDL decomposition in D and C (in case of MODE=1), IPRINT, IOUT
10	Internal inconsistency of the quadratic subproblem, division by zero.
11	More than MAXFUN successive non-evaluable function calls.
100	The solution of the quadratic programming subproblem has been terminated with an error message and IFAIL is set to IFQL+100, where IFQL denotes the index of an inconsistent constraint.

1.23.3 API

class pyoptsparse.pyNLPQLP.pyNLPQLP.**NLPQLP**(**args*, ***kwargs*)

NLPQL Optimizer Class - Inherited from Optimizer Abstract Class

This is the base optimizer class that all optimizers inherit from. We define common methods here to avoid code duplication.

Parameters

name
[str] Optimizer name

category
[str] Typically local or global

defaultOptions
[dictionary] A dictionary containing the default options

informs
[dict] Dictionary of the inform codes

__call__(*optProb*, *sens=None*, *sensStep=None*, *sensMode=None*, *storeHistory=None*, *hotStart=None*, *storeSens=True*)

This is the main routine used to solve the optimization problem.

Parameters

optProb
[Optimization or Solution class instance] This is the complete description of the optimization problem to be solved by the optimizer

sens
[str or python Function.] Specify method to compute sensitivities. To explicitly use pyOptSparse gradient class to do the derivatives with finite differences use 'FD'. 'sens' may also be 'CS' which will cause pyOptSpare to compute the derivatives using the complex step method. Finally, 'sens' may be a python function handle which is expected to compute the sensitivities directly. For expensive function evaluations and/or problems with large numbers of design variables this is the preferred method.

sensStep
[float] Set the step size to use for design variables. Defaults to 1e-6 when sens is 'FD' and 1e-40j when sens is 'CS'.

sensMode
[str] Use 'pgc' for parallel gradient computations. Only available with mpi4py and each objective evaluation is otherwise serial

storeHistory
[str] File name of the history file into which the history of this optimization will be stored

hotStart
[str] File name of the history file to "replay" for the optimization. The optimization problem used to generate the history file specified in 'hotStart' must be **IDENTICAL** to the currently supplied 'optProb'. By identical we mean, **EVERY SINGLE PARAMETER MUST BE IDENTICAL**. As soon as he requested evaluation point from NLPQL does not match the history, function and gradient evaluations revert back to normal evaluations.

storeSens
[bool] Flag specifying if sensitivities are to be stored in hist. This is necessary for hotstarting only.

1.24 NSGA2

This optimizer is a non-dominating sorting genetic algorithm that solves non-convex and non-smooth single and multiobjective optimization problems. The algorithm attempts to perform global optimization, while enforcing constraints using a tournament selection-based strategy

Warning: Currently, the Python wrapper does not catch exceptions. If there is **any** error in the user-supplied function, you will get a seg-fault and no idea where it happened. Please make sure the objective is without errors before trying to use nsga2.

1.24.1 Options

Table 9: NSGA2 Default Options

Name	Type	Default value	Description
PopSize	int	100	Population size
maxGen	int	1000	Maximum number of generations
pCross_real	float	0.6	Probability of crossover of real variables
pMut_real	float	0.2	Probability of mutation of real variables
eta_c	float	10.0	Distribution index for crossover
eta_m	float	20.0	Distribution index for mutation
pCross_bin	float	0.0	Probability of crossover of binary variable
pMut_bin	float	0.0	Probability of mutation of binary variables
PrintOut	int	1	Flag to turn on output to filename
seed	int	0	Random Number Seed (0 - Auto-Seed based on time clock)
xinit	int	0	Use Initial Solution Flag (0 - random population, 1 - use given solution)

1.24.2 API

class pyoptsparse.pyNSGA2.pyNSGA2.**NSGA2**(**args*, ***kwargs*)

NSGA2 Optimizer Class - Inherited from Optimizer Abstract Class

This is the base optimizer class that all optimizers inherit from. We define common methods here to avoid code duplication.

Parameters

name
[str] Optimizer name

category
[str] Typically local or global

defaultOptions
[dictionary] A dictionary containing the default options

informs
[dict] Dictionary of the inform codes

__call__(*optProb*, *storeHistory=None*, *hotStart=None*, ***kwargs*)

This is the main routine used to solve the optimization problem.

Parameters

optProb
[Optimization or Solution class instance] This is the complete description of the optimization problem to be solved by the optimizer

storeHistory
[str] File name of the history file into which the history of this optimization will be stored

hotStart
[str] File name of the history file to "replay" for the optimziation. The optimization problem used to generate the history file specified in 'hotStart' must be **IDENTICAL** to the currently supplied 'optProb'. By identical we mean, **EVERY SINGLE PARAMETER MUST BE IDENTICAL**. As soon as he requested evaluation point from NSGA2 does not match the history, function and gradient evaluations revert back to normal evaluations.

Notes

The kwargs are there such that the sens= argument can be supplied (but ignored here in nsga2)

1.25 PSQP

This optimizer implements a sequential quadratic programming method with a BFGS variable metric update

1.25.1 Options

Table 10: PSQP Default Options

Name	Type	Default value	Description
XMAX	float	1e+16	Maximum Stepsize
TOLX	float	1e-16	Variable Change Tolerance
TOLC	float	1e-06	Constraint Violation Tolerance
TOLG	float	1e-06	Lagrangian Gradient Tolerance
RPF	float	0.0001	Penalty Coefficient
MIT	int	1000	Maximum Number of Iterations
MFV	int	2000	Maximum Number of Function Evaluations
MET	int	2	Variable Metric Update (1 - BFGS, 2 - Hoshino)
MEC	int	2	Negative Curvature Correction (1 - None, 2 - Powell's Correction)
IPRINT	int	2	Output Level (0 - None, 1 - Final, 2 - Iter)
IOUT	int	6	Output Unit Number
IFILE	str	PSQP.out	Output File Name

1.25.2 Informs

Table 11: PSQP Informs

Code	Description
1	Change in design variable was less than or equal to tolerance
2	Change in objective function was less than or equal to tolerance
3	Objective function less than or equal to tolerance
4	Maximum constraint value is less than or equal to tolerance
11	Maximum number of iterations exceeded
12	Maximum number of function evaluations exceeded
13	Maximum number of gradient evaluations exceeded
-6	Termination criterion not satisfied, but obtained point is acceptable
-7	Positive directional derivative in line search
-8	Interpolation error in line search
-10	Optimization failed

1.25.3 API

class pyoptsparse.pyPSQP.pyPSQP.**PSQP**(**args*, ***kwargs*)

PSQP Optimizer Class - Inherited from Optimizer Abstract Class

This is the base optimizer class that all optimizers inherit from. We define common methods here to avoid code duplication.

Parameters

name
[str] Optimizer name

category
[str] Typically local or global

defaultOptions
[dictionary] A dictionary containing the default options

informs
[dict] Dictionary of the inform codes

__call__(*optProb*, *sens=None*, *sensStep=None*, *sensMode=None*, *storeHistory=None*, *hotStart=None*, *storeSens=True*)

This is the main routine used to solve the optimization problem.

Parameters

optProb
[Optimization or Solution class instance] This is the complete description of the optimization problem to be solved by the optimizer

sens
[str or python Function.] Specifiy method to compute sensitivities. To explictly use pyOptSparse gradient class to do the derivatives with finite differenes use 'FD'. 'sens' may also be 'CS' which will cause pyOptSpare to compute the derivatives using the complex step method. Finally, 'sens' may be a python function handle which is expected to compute the sensitivities directly. For expensive function evaluations and/or problems with large numbers of design variables this is the preferred method.

sensStep
[float] Set the step size to use for design variables. Defaults to 1e-6 when sens is 'FD' and 1e-40j when sens is 'CS'.

sensMode
[str] Use 'pgc' for parallel gradient computations. Only available with mpi4py and each objective evaluation is otherwise serial

storeHistory
[str] File name of the history file into which the history of this optimization will be stored

hotStart
[str] File name of the history file to "replay" for the optimziation. The optimization problem used to generate the history file specified in 'hotStart' must be **IDENTICAL** to the currently supplied 'optProb'. By identical we mean, **EVERY SINGLE PARAMETER MUST BE IDENTICAL**. As soon as he requested evaluation point from PSQP does not match the history, function and gradient evaluations revert back to normal evaluations.

storeSens
[bool] Flag sepcifying if sensitivities are to be stored in hist. This is necessay for hotstarting only.

1.26 ParOpt

ParOpt is a nonlinear interior point optimizer that is designed for large parallel design optimization problems with structured sparse constraints. ParOpt is open source and can be downloaded at https://github.com/smdogroup/paropt. Documentation and examples for ParOpt can be found at https://smdogroup.github.io/paropt/. The version of ParOpt supported is v2.0.2.

1.26.1 Installation

Please follow the instructions here to install ParOpt as a separate Python package. Make sure that the package is named `paropt` and the installation location can be found by Python, so that `from paropt import ParOpt` works within the pyOptSparse folder. This typically requires installing it in a location which is already present under `$PYTHONPATH` environment variable, or you can modify the `.bashrc` file and manually append the path.

1.26.2 Options

Please see the ParOpt documentation for all available options.

1.26.3 API

class pyoptsparse.pyParOpt.ParOpt.**ParOpt**(**args*, ***kwargs*)

ParOpt optimizer class

ParOpt has the capability to handle distributed design vectors. This is not replicated here since pyOptSparse does not have the capability to handle this type of design problem.

This is the base optimizer class that all optimizers inherit from. We define common methods here to avoid code duplication.

Parameters

name
[str] Optimizer name

category
[str] Typically local or global

defaultOptions
[dictionary] A dictionary containing the default options

informs
[dict] Dictionary of the inform codes

__call__(*optProb*, *sens=None*, *sensStep=None*, *sensMode=None*, *storeHistory=None*, *hotStart=None*, *storeSens=True*)

This is the main routine used to solve the optimization problem.

Parameters

optProb
[Optimization or Solution class instance] This is the complete description of the optimization problem to be solved by the optimizer

sens
[str or python Function.] Specifiy method to compute sensitivities. To explictly use pyOptSparse gradient class to do the derivatives with finite differenes use 'FD'. 'sens' may also be 'CS' which will cause pyOptSpare to compute the derivatives using the complex step method. Finally, 'sens' may be a python function handle which is expected to compute the sensitivities directly. For expensive function evaluations and/or problems with large numbers of design variables this is the preferred method.

sensStep
[float] Set the step size to use for design variables. Defaults to 1e-6 when sens is 'FD' and 1e-40j when sens is 'CS'.

sensMode
[str] Use 'pgc' for parallel gradient computations. Only available with mpi4py and each objective evaluation is otherwise serial

storeHistory
[str] File name of the history file into which the history of this optimization will be stored

hotStart
[str] File name of the history file to "replay" for the optimziation. The optimization problem used to generate the history file specified in 'hotStart' must be **IDENTICAL** to the currently supplied 'optProb'. By identical we mean, **EVERY SINGLE PARAMETER MUST BE IDENTICAL**. As soon as he requested evaluation point from ParOpt does not match the history, function and gradient evaluations revert back to normal evaluations.

storeSens
[bool] Flag sepcifying if sensitivities are to be stored in hist. This is necessay for hotstarting only.

1.27 CONMIN

CONstrained function MINimization (CONMIN) is a gradient-based optimizer that uses the methods of feasible directions.

1.27.1 Options

Table 12: CONMIN Default Options

Name	Type	Default value	Description
`ITMAX`	int	10000	Maximum Number of Iterations
`DELFUN`	float	1e-06	Objective Relative Tolerance
`DABFUN`	float	1e-06	Objective Absolute Tolerance
`ITRM`	int	5	None
`NFEASCT`	int	20	None
`IPRINT`	int	4	Print Control (0 - None, 1 - Final, 2,3,4 - Debug)
`IOUT`	int	6	Output Unit Number
`IFILE`	str	`CONMIN.out`	Output File Name

1.27.2 API

class pyoptsparse.pyCONMIN.pyCONMIN.**CONMIN**(**args*, ***kwargs*)

CONMIN Optimizer Class - Inherited from Optimizer Abstract Class

This is the base optimizer class that all optimizers inherit from. We define common methods here to avoid code duplication.

Parameters

name
[str] Optimizer name

category
[str] Typically local or global

defaultOptions
[dictionary] A dictionary containing the default options

informs
[dict] Dictionary of the inform codes

__call__(*optProb*, *sens=None*, *sensStep=None*, *sensMode=None*, *storeHistory=None*, *hotStart=None*, *storeSens=True*)

This is the main routine used to solve the optimization problem.

Parameters

optProb
[Optimization or Solution class instance] This is the complete description of the optimization problem to be solved by the optimizer

sens
[str or python Function.] Specifiy method to compute sensitivities. To explictly use pyOptSparse gradient class to do the derivatives with finite differenes use 'FD'. 'sens' may also be 'CS' which will cause pyOptSpare to compute the derivatives using the complex step method. Finally, 'sens' may be a python function handle which is expected to compute the sensitivities directly. For expensive function evaluations and/or problems with large numbers of design variables this is the preferred method.

sensStep
[float] Set the step size to use for design variables. Defaults to 1e-6 when sens is 'FD' and 1e-40j when sens is 'CS'.

sensMode
[str] Use 'pgc' for parallel gradient computations. Only available with mpi4py and each objective evaluation is otherwise serial

storeHistory
[str] File name of the history file into which the history of this optimization will be stored

hotStart
[str] File name of the history file to "replay" for the optimziation. The optimization problem used to generate the history file specified in 'hotStart' must be **IDENTICAL** to the currently supplied 'optProb'. By identical we mean, **EVERY SINGLE PARAMETER MUST BE IDENTICAL**. As soon as he requested evaluation point from CONMIN does not match the history, function and gradient evaluations revert back to normal evaluations.

storeSens
[bool] Flag sepcifying if sensitivities are to be stored in hist. This is necessay for hotstarting only.

1.28 ALPSO

Augmented Lagrangian Particle Swarm Optimizer (ALPSO) is a PSO method that uses the augmented Lagrangian approach to handle constraints.

1.28.1 Options

Table 13: ALPSO Default Options

Name	Type	Default value	Description
SwarmSize	int	40	Number of Particles (Depends on Problem dimensions)
maxOuterIter	int	200	Maximum Number of Outer Loop Iterations (Major Iterations)
maxInnerIter	int	6	Maximum Number of Inner Loop Iterations (Minor Iterations)
minInnerIter	int	6	Minimum Number of Inner Loop Iterations (Dynamic Inner Iterations)
dynInnerIter	int	0	Dynamic Number of Inner Iterations Flag
stopCriteria	int	1	Stopping Criteria Flag (0 - maxIters, 1 - convergence)
stopIters	int	5	Consecutive Number of Iterations for which the Stopping Criteria must be Satisfied
etol	float	0.001	Absolute Tolerance for Equality constraints
itol	float	0.001	Absolute Tolerance for Inequality constraints
rtol	float	0.01	Relative Tolerance for Lagrange Multipliers
atol	float	0.01	Absolute Tolerance for Lagrange Function
dtol	float	0.1	Relative Tolerance in Distance of All Particles to Terminate (GCPSO)
printOuterIters	int	0	Number of Iterations Before Print Outer Loop Information
printInnerIters	int	0	Number of Iterations Before Print Inner Loop Information
rinit	float	1.0	Initial Penalty Factor
xinit	int	0	Initial Position Flag (0 - no position, 1 - position given)

continues on next page

Table 13 – continued from previous page

Name	Type	Default value	Description
vinit	float	1.0	Initial Velocity of Particles in Normalized [-1, 1] Design Space
vmax	float	2.0	Maximum Velocity of Particles in Normalized [-1, 1] Design Space
c1	float	2.0	Cognitive Parameter
c2	float	1.0	Social Parameter
w1	float	0.99	Initial Inertia Weight
w2	float	0.55	Final Inertia Weight
ns	int	15	Number of Consecutive Successes in Finding New Best Position of Best Particle Before Search Radius will be Increased (GCPSO)
nf	int	5	Number of Consecutive Failures in Finding New Best Position of Best Particle Before Search Radius will be Increased (GCPSO)
dt	float	1.0	Time step
vcrazy	float	0.0001	Craziness Velocity (Added to Particle Velocity After Updating the Penalty Factors and Langangian Multipliers)
fileout	int	1	Flag to Turn On Output to filename
filename	str	ALPSO.out	We could probably remove fileout flag if filename or fileinstance is given
seed	int	0	Random Number Seed (0 - Auto-Seed based on time clock)
HoodSize	int	40	Number of Neighbours of Each Particle
HoodModel	str	gbest	Neighbourhood Model (dl/slring - Double/Single Link Ring, wheel - Wheel, Spatial - based on spatial distance, sfrac - Spatial Fraction)
HoodSelf	int	1	Selfless Neighbourhood Model (0 - Include Particle i in NH i, 1 - Don't Include Particle i)

continues on next page

Table 13 – continued from previous page

Name	Type	Default value	Description
`Scaling`	int	1	Design Variables Scaling Flag (0 - no scaling, 1 - scaling between [-1, 1])
`parallelType`	str	`None`	Type of parallelization • `None`: No parallel function evaluations • `EXT`: Use parallel function evaluations

1.28.2 API

class pyoptsparse.pyALPSO.pyALPSO.**ALPSO**(**args*, ***kwargs*)

ALPSO Optimizer Class - Inherited from Optimizer Abstract Class

*Keyword arguments:**

- pll_type -> STR: ALPSO Parallel Implementation (None, SPM- Static, DPM- Dynamic, POA-Parallel Analysis), *Default* = None

This is the base optimizer class that all optimizers inherit from. We define common methods here to avoid code duplication.

Parameters

name
[str] Optimizer name

category
[str] Typically local or global

defaultOptions
[dictionary] A dictionary containing the default options

informs
[dict] Dictionary of the inform codes

__call__(*optProb*, *storeHistory=None*, *hotStart=None*, ***kwargs*)

This is the main routine used to solve the optimization problem.

Parameters

optProb
[Optimization or Solution class instance] This is the complete description of the optimization problem to be solved by the optimizer

storeHistory
[str] File name of the history file into which the history of this optimization will be stored

hotStart
[str] File name of the history file to "replay" for the optimization. The optimization problem used to generate the history file specified in 'hotStart' must be **IDENTICAL** to the currently supplied 'optProb'. By identical we mean, **EVERY SINGLE PARAMETER MUST BE IDENTICAL**. As soon as he requested evaluation point from ALPSO does not match the history and function evaluations revert back to normal evaluations.

Notes

The kwargs are there such that the sens= argument can be supplied (but ignored here in alpso)

PYTHON MODULE INDEX

p

INDEX

Symbols

A

C

D

E

F

G