cyipopt Documentation

Release 1.2.0

cyipopt Developers

Oct 11, 2023

CONTENTS

cyipopt: Python wrapper for the Ipopt optimization package, written in Cython.

Ipopt (Interior Point Optimizer, pronounced ''Eye-Pea-Opt") is an open source software package for large-scale nonlinear optimization. It is designed to find (local) solutions of mathematical optimization problems of the form

$$\min_{x \in R^n} f(x)$$

subject to

$$g_L \leq g(x) \leq g_U$$
$$x_L \leq x \leq x_U$$

Where x are the optimization variables (possibly with upper and lower bounds), $f(x)$ is the objective function and $g(x)$ are the general nonlinear constraints. The constraints, $g(x)$, have lower and upper bounds. Note that equality constraints can be specified by setting $g_L^i = g_U^i$.

cyipopt is a python wrapper around Ipopt. It enables using Ipopt from the comfort of the Python programming language. cyipopt is available under the EPL (Eclipse Public License) open-source license.

Contents:

CHAPTER

ONE

INSTALLATION

1.1 Using conda

Conda is a cross platform package manager and provides the easiest mechanism to install cyipopt on Linux, Mac, and Windows. Once conda is installed, install cyipopt from the Conda Forge channel with:

```
$ conda install -c conda-forge cyipopt
```

The above command will install binary versions of all the necessary dependencies as well as cyipopt. Conda Forge supplies a basic build of Ipopt that is suitable for many use cases. You will have to install from source if you want a customized Ipopt installation.

1.2 From source

To begin installing from source you will need to install the following dependencies:

- C/C++ compiler
- pkg-config [only for Linux and Mac]
- Ipopt [>= 3.13 on Windows]
- Python 3.6+
- setuptools
- cython
- numpy
- scipy [optional]

The binaries and header files of the Ipopt package can be obtained from http://www.coin-or.org/download/binary/Ipopt/. These include a version compiled against the MKL library. Or you can build Ipopt from source. The remaining dependencies can be installed with conda or other package managers.

1.2.1 On Linux and Mac

For Linux and Mac, the `ipopt` executable should be in your path and discoverable by pkg-config, i.e. this command should return a valid result:

```
$ pkg-config --libs --cflags ipopt
```

You will need to install Ipopt in a system location or set `LD_LIBRARY_PATH` if pkg-config does not find the executable.

Once all the dependencies are installed, execute:

```
$ python setup.py install
```

to build and install the package.

1.2.2 From source on Windows

Install the dependencies with conda (Anaconda or Miniconda):

```
$ conda.exe install -c conda-forge numpy cython setuptools
```

Or alternatively with pip:

```
$ pip install numpy cython setuptools
```

Additionally, make sure you have a C compiler setup to compile Python C extensions, e.g. Visual C++. Build tools for VS2019 https://visualstudio.microsoft.com/downloads/#build-tools-for-visual-studio-2019 have been tested to work for conda Python 3.7 (see https://github.com/mechmotum/cyipopt/issues/52).

Download and extract the cyipopt source code from Github or PyPi.

Obtain IPOPT one of two ways:

1. Using official IPOPTs binaries:

Download the latest precompiled version of Ipopt that includes the DLL files from https://github.com/coin-or/Ipopt/releases. Note that the current setup only supports Ipopt >= 3.13.0. The build 3.13.3 of Ipopt has been confirmed to work and can be downloaded from Ipopt-3.13.3-win64-msvs2019-md.zip. After Ipopt is extracted, the `bin`, `lib` and `include` folders should be in the root cyipopt directory, i.e. adjacent to the `setup.py` file. Alternatively, you can set the environment variable `IPOPTWINDIR` to point to the Ipopt directory that contains the `bin`, `lib` and `include` directories.

2. Using Conda Forge's IPOPT binary:

If using conda, you can install an IPOPT binary from Conda Forge:

```
$ conda.exe install -c conda-forge ipopt
```

The environment variable `IPOPTWINDIR` should then be set to `USECONDAFORGEIPOPT`.

Finally, execute:

```
$ python setup.py install
```

NOTE: It is advised to use the Anaconda or Miniconda distributions and *not* the official python.org distribution. Even though it has been tested to work with the latest builds, it is well-known for causing issues. (see https://github.com/mechmotum/cyipopt/issues/52).

1.3 On Ubuntu 18.04 Using APT Dependencies

All of the dependencies can be installed with Ubuntu's package manager:

```
$ sudo apt install build-essential pkg-config python-dev cython python-numpy coinor-
↪libipopt1v5 coinor-libipopt-dev
```

The NumPy and IPOPT libs and headers are installed in standard locations, so you should not need to set `LD_LIBRARY_PATH` or `PKG_CONFIG_PATH`.

Now run `python setup.py build` to compile cyipopt. In the output of this command you should see two calls to `gcc` for compiling and linking. Make sure both of these are pointing to the correct libraries and headers. They will look something like this (formatted and commented for easy viewing here):

```
$ python setup.py build
...
x86_64-linux-gnu-gcc -pthread -DNDEBUG -g -fwrapv -O2 -Wall -Wstrict-prototypes -fno-
↪strict-aliasing
  -Wdate-time -D_FORTIFY_SOURCE=2 -g -fdebug-prefix-map=/build/python2.7-3hk45v/python2.
↪7-2.7.15~rc1=.
  -fstack-protector-strong -Wformat -Werror=format-security -fPIC
  -I/usr/local/include/coin  # points to IPOPT headers
  -I/usr/local/include/coin/ThirdParty  # points to IPOPT third party headers
  -I/usr/lib/python2.7/dist-packages/numpy/core/include  # points to NumPy headers
  -I/usr/include/python2.7  # points to Python 2.7 headers
  -c src/cyipopt.c -o build/temp.linux-x86_64-2.7/src/cyipopt.o
x86_64-linux-gnu-gcc -pthread -shared -Wl,-O1 -Wl,-Bsymbolic-functions -Wl,-Bsymbolic-
↪functions -Wl,-z,relro
  -fno-strict-aliasing -DNDEBUG -g -fwrapv -O2 -Wall -Wstrict-prototypes -Wdate-time -D_
↪FORTIFY_SOURCE=2 -g
  -fdebug-prefix-map=/build/python2.7-3hk45v/python2.7-2.7.15~rc1=. -fstack-protector-
↪strong -Wformat
  -Werror=format-security -Wl,-Bsymbolic-functions -Wl,-z,relro -Wdate-time -D_FORTIFY_
↪SOURCE=2 -g
  -fdebug-prefix-map=/build/python2.7-3hk45v/python2.7-2.7.15~rc1=. -fstack-protector-
↪strong -Wformat
  -Werror=format-security build/temp.linux-x86_64-2.7/src/cyipopt.o
  -L/usr/local/lib
  -L/lib/../lib
  -L/usr/lib/../lib
  -L/usr/lib/gcc/x86_64-linux-gnu/5
  -L/usr/lib/gcc/x86_64-linux-gnu/5/../../..
  -L/usr/lib/gcc/x86_64-linux-gnu/5/../../../../lib
  -L/usr/lib/gcc/x86_64-linux-gnu/5/../../../x86_64-linux-gnu
  -lipopt -llapack -lblas -lm -ldl -lcoinmumps -lblas -lgfortran -lm -lquadmath  #␣
↪linking to relevant libs
  -lcoinhsl -llapack -lblas -lgfortran -lm -lquadmath -lcoinmetis  # linking to relevant␣
↪libs
  -o build/lib.linux-x86_64-2.7/cyipopt.so
...
```

You can check that everything linked correctly with `ldd`:

```
$ ldd build/lib.linux-x86_64-2.7/cyipopt.so
linux-vdso.so.1 (0x00007ffc1677c000)
libipopt.so.0 => /usr/local/lib/libipopt.so.0 (0x00007fcdc8668000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007fcdc8277000)
libcoinmumps.so.0 => /usr/local/lib/libcoinmumps.so.0 (0x00007fcdc7eef000)
libcoinhsl.so.0 => /usr/local/lib/libcoinhsl.so.0 (0x00007fcdc7bb4000)
liblapack.so.3 => /usr/lib/x86_64-linux-gnu/liblapack.so.3 (0x00007fcdc732e000)
libblas.so.3 => /usr/lib/x86_64-linux-gnu/libblas.so.3 (0x00007fcdc70d3000)
libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007fcdc6ecf000)
libstdc++.so.6 => /usr/lib/x86_64-linux-gnu/libstdc++.so.6 (0x00007fcdc6b46000)
libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6 (0x00007fcdc67a8000)
/lib64/ld-linux-x86-64.so.2 (0x00007fcdc8d20000)
libgcc_s.so.1 => /lib/x86_64-linux-gnu/libgcc_s.so.1 (0x00007fcdc6590000)
libcoinmetis.so.0 => /usr/local/lib/libcoinmetis.so.0 (0x00007fcdc6340000)
libgfortran.so.3 => /usr/lib/x86_64-linux-gnu/libgfortran.so.3 (0x00007fcdc600f000)
libopenblas.so.0 => /usr/lib/x86_64-linux-gnu/libopenblas.so.0 (0x00007fcdc3d69000)
libgfortran.so.4 => /usr/lib/x86_64-linux-gnu/libgfortran.so.4 (0x00007fcdc398a000)
libquadmath.so.0 => /usr/lib/x86_64-linux-gnu/libquadmath.so.0 (0x00007fcdc374a000)
libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007fcdc352b000)
```

And finally install the package into Python's default package directory:

```
$ python setup.py install
```

Note that you may or may not want to install this package system wide, i.e. prepend `sudo` to the above command, but it is safest to install into your user space, i.e. what `pip install --user` does, or setup a virtual environment with tools like venv or conda. If you use virtual environments you will need to be careful about selecting headers and libraries for packages in or out of the virtual environments in the build step. Note that cython, and numpy could alternatively be installed using Python specific package managers, e.g. `pip install cython numpy`.

1.4 On Ubuntu 18.04 with Custom Compiled IPOPT

Install system wide dependencies:

```
$ sudo apt install pkg-config python-dev wget
$ sudo apt build-dep coinor-libipopt1v5
```

Install `pip` so all Python packages can be installed via `pip`:

```
$ sudo apt install python-pip
```

Then use `pip` to install the following packages:

```
$ pip install --user numpy cython
```

1.4.1 Compile Ipopt

The Ipopt compilation instructions are derived from https://www.coin-or.org/Ipopt/documentation/node14.html. If you get errors, start there for help.

Download Ipopt source code. Choose the version that you would like to have from <https://www.coin-or.org/download/source/Ipopt/>. For example:

```
$ cd ~
$ wget https://www.coin-or.org/download/source/Ipopt/Ipopt-3.12.11.tgz
```

Extract the Ipopt source code:

```
$ tar -xvf Ipopt-3.12.11.tgz
```

Create a temporary environment variable pointing to the Ipopt directory:

```
$ export IPOPTDIR=~/Ipopt-3.12.11
```

To use linear solvers other than the default mumps, e.g. `ma27, ma57, ma86` solvers, the `HSL` package are needed. `HSL` can be downloaded from its official website <http://www.hsl.rl.ac.uk/ipopt/>.

Extract `HSL` source code after you get it. Rename the extracted folder to `coinhsl` and copy it in the HSL folder: `Ipopt-3.12.11/ThirdParty/HSL`

Build Ipopt:

```
$ mkdir $IPOPTDIR/build
$ cd $IPOPTDIR/build
$ ../configure
$ make
$ make test
```

Add `make install` if you want a system wide install.

Set environment variables:

```
$ export IPOPT_PATH="~/Ipopt-3.12.11/build"
$ export PKG_CONFIG_PATH=$PKG_CONFIG_PATH:$IPOPT_PATH/lib/pkgconfig
$ export PATH=$PATH:$IPOPT_PATH/bin
```

Get help from this web-page if you get errors in setting environments:

https://stackoverflow.com/questions/13428910/how-to-set-the-environmental-variable-ld-library-path-in-linux

Now compile `cyipopt`. Download the `cyipopt` source code from PyPi, for example:

```
$ cd ~
$ wget https://files.pythonhosted.org/packages/05/57/
↪a7c5a86a8f899c5c109f30b8cdb278b64c43bd2ea04172cbfed721a98fac/ipopt-0.1.9.tar.gz
$ tar -xvf ipopt-0.1.8.tar.gz
$ cd ipopt
```

Compile `cyipopt`:

```
$ python setup.py build
```

If there is no error, then you have compiled `cyipopt` successfully

Check that everything linked correctly with `ldd`

```
$ ldd build/lib.linux-x86_64-2.7/cyipopt.so
linux-vdso.so.1 (0x00007ffe895e1000)
libipopt.so.1 => /home/<username>/Ipopt-3.12.11/build/lib/libipopt.so.1␣
↪(0x00007f74efc2a000)
libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f74ef839000)
libcoinmumps.so.1 => /home/<username>/Ipopt-3.12.11/build/lib/libcoinmumps.so.1␣
↪(0x00007f74ef4ae000)
libcoinhsl.so.1 => /home/<username>/Ipopt-3.12.11/build/lib/libcoinhsl.so.1␣
↪(0x00007f74ef169000)
liblapack.so.3 => /usr/lib/x86_64-linux-gnu/liblapack.so.3 (0x00007f74ee8cb000)
libblas.so.3 => /usr/lib/x86_64-linux-gnu/libblas.so.3 (0x00007f74ee65e000)
libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007f74ee45a000)
libstdc++.so.6 => /usr/lib/x86_64-linux-gnu/libstdc++.so.6 (0x00007f74ee0d1000)
libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6 (0x00007f74edd33000)
/lib64/ld-linux-x86-64.so.2 (0x00007f74f02c0000)
libgcc_s.so.1 => /lib/x86_64-linux-gnu/libgcc_s.so.1 (0x00007f74edb1b000)
libcoinmetis.so.1 => /home/<username>/Ipopt-3.12.11/build/lib/libcoinmetis.so.1␣
↪(0x00007f74ed8ca000)
libgfortran.so.4 => /usr/lib/x86_64-linux-gnu/libgfortran.so.4 (0x00007f74ed4eb000)
```

Install `cyipopt` (prepend `sudo` if you want a system wide install):

```
$ python setup.py install
```

To use `cyipopt` you will need to set the `LD_LIBRARY_PATH` to point to your Ipopt install if you did not install it to a standard location. For example:

```
$ export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:~/Ipopt-3.12.11/build/lib
```

You can add this to your shell's configuration file if you want it set every time you open your shell, for example the following line can it can be added to your `~/.bashrc`

```
$ echo 'export LD_LIBRARY_PATH=$LD_LIBRARY_PATH:$HOME/Ipopt-3.12.11/build/lib' >> ~/.
↪bashrc
```

Now you should be able to run a `cyipopt` example:

```
$ cd test
$ python -c "import cyipopt"
$ python examplehs071.py
```

If it could be run successfully, the optimization will start with the following descriptions:

```
******************************************************************************
This program contains Ipopt, a library for large-scale nonlinear optimization.
 Ipopt is released as open source code under the Eclipse Public License (EPL).
         For more information visit http://projects.coin-or.org/Ipopt
******************************************************************************

This is Ipopt version 3.12.11, running with linear solver ma27.
...
```

1.5 Conda Forge binaries with HSL

It is possible to use the HSL linear solvers with cyipopt installed via Conda Forge. To do so, first download the HSL source code tarball. The following explanation uses `coinhsl-2014.01.10.tar.gz` with conda installed on Ubuntu 20.04.

Create a conda environment with at least gfortran and cyipopt:

```
$ conda create -n hsl-test -c conda-forge gfortran cyipopt
$ conda activate hsl-test
```

You should now have an environment that includes ipopt. You can checked what ipopt is linked against like so:

```
(hsl-test) $ ldd ~/miniconda/envs/hsl-test/lib/libipopt.so
   linux-vdso.so.1 (0x00007ffcaf45b000)
   librt.so.1 => /lib/x86_64-linux-gnu/librt.so.1 (0x00007f8965748000)
   liblapack.so.3 => /home/<username>/miniconda/envs/hsl-test/lib/./liblapack.so.3␣
↪(0x00007f89635fe000)
   libdmumps_seq-5.2.1.so => /home/<username>/miniconda/envs/hsl-test/lib/./libdmumps_
↪seq-5.2.1.so (0x00007f89633d8000)
   libmumps_common_seq-5.2.1.so => /home/<username>/miniconda/envs/hsl-test/lib/./
↪libmumps_common_seq-5.2.1.so (0x00007f8963377000)
   libpord_seq-5.2.1.so => /home/<username>/miniconda/envs/hsl-test/lib/./libpord_seq-5.
↪2.1.so (0x00007f896335e000)
   libmpiseq_seq-5.2.1.so => /home/<username>/miniconda/envs/hsl-test/lib/./libmpiseq_
↪seq-5.2.1.so (0x00007f8963352000)
   libesmumps-6.so => /home/<username>/miniconda/envs/hsl-test/lib/./libesmumps-6.so␣
↪(0x00007f8963349000)
   libscotch-6.so => /home/<username>/miniconda/envs/hsl-test/lib/./libscotch-6.so␣
↪(0x00007f89632b1000)
   libscotcherr-6.so => /home/<username>/miniconda/envs/hsl-test/lib/./libscotcherr-6.so␣
↪(0x00007f89632ac000)
   libmetis.so => /home/<username>/miniconda/envs/hsl-test/lib/./libmetis.so␣
↪(0x00007f8963237000)
   libgfortran.so.5 => /home/<username>/miniconda/envs/hsl-test/lib/./libgfortran.so.5␣
↪(0x00007f896308e000)
   libdl.so.2 => /lib/x86_64-linux-gnu/libdl.so.2 (0x00007f8963088000)
   libstdc++.so.6 => /home/<username>/miniconda/envs/hsl-test/lib/./libstdc++.so.6␣
↪(0x00007f8962edb000)
   libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6 (0x00007f8962d8c000)
   libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f8962b9a000)
   /lib64/ld-linux-x86-64.so.2 (0x00007f8965a02000)
   libgcc_s.so.1 => /home/<username>/miniconda/envs/hsl-test/lib/./libgcc_s.so.1␣
↪(0x00007f8962b85000)
   libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007f8962b62000)
   libgomp.so.1 => /home/<username>/miniconda/envs/hsl-test/lib/./../libgomp.so.1␣
↪(0x00007f8962b2a000)
   libz.so.1 => /home/<username>/miniconda/envs/hsl-test/lib/./../libz.so.1␣
↪(0x00007f8962b10000)
   libquadmath.so.0 => /home/<username>/miniconda/envs/hsl-test/lib/./../libquadmath.so.0␣
↪(0x00007f8962ad6000)
```

Now navigate into the extracted HSL directory and configure HSL:

```
(hsl-test) $ cd /path/to/coinhsl-2014.01.10/
(hsl-test) $ ./configure \
   --prefix=/home/<username>/miniconda/envs/hsl-test/ \
   --with-blas="-L/home/<username>/miniconda/envs/hsl-test/lib/ -lblas" \
   LIBS="-llapack" \
   FC=/home/<username>/miniconda/envs/hsl-test/bin/gfortran \
   CC=/home/<username>/miniconda/envs/hsl-test/bin/gcc \
```

This tells HSL to install into your environment, link against the environment's blas and lapack libraries and to use the environment's gfortran and gcc compilers to build HSL. After configuring, build and install with:

```
(hsl-test) $ make
(hsl-test) $ make install
```

You should now find a shared HSL library in your environment. Check to make sure it is properly linked (especially blas):

```
(hsl-test) $ ldd ~/miniconda/envs/hsl-test/lib/libcoinhsl.so
   linux-vdso.so.1 (0x00007ffe2085a000)
   libopenblas.so.0 => /home/<username>/miniconda/envs/hsl-test/lib/libopenblas.so.0␣
↪(0x00007f72a1766000)
   libgfortran.so.5 => /home/<username>/miniconda/envs/hsl-test/lib/libgfortran.so.5␣
↪(0x00007f72a15bd000)
   libm.so.6 => /lib/x86_64-linux-gnu/libm.so.6 (0x00007f72a143f000)
   libgcc_s.so.1 => /home/<username>/miniconda/envs/hsl-test/lib/libgcc_s.so.1␣
↪(0x00007f72a142a000)
   libquadmath.so.0 => /home/<username>/miniconda/envs/hsl-test/lib/libquadmath.so.0␣
↪(0x00007f72a13f0000)
   libc.so.6 => /lib/x86_64-linux-gnu/libc.so.6 (0x00007f72a11fe000)
   libpthread.so.0 => /lib/x86_64-linux-gnu/libpthread.so.0 (0x00007f72a11d9000)
   /lib64/ld-linux-x86-64.so.2 (0x00007f72a39d4000)
```

Now, in your cyipopt script set the following two options:

```
problem.add_option('linear_solver', 'ma57')
problem.add_option('hsllib', 'libcoinhsl.so')
```

The various HSL solvers can be set with `linear_solver` and the `hsllib` name must be specified because the default name ipopt looks for is `libhsl.so`. Identify the shared library installed on your system and make sure the name provided for the `hsllib` option matches. For example, on macOS you may need `problem.add_option('hsllib', 'libcoinhsl.dylib')`.

CHAPTER

TWO

USAGE

2.1 SciPy Compatible Interface

For simple cases where you do not need the full power of sparse and structured Jacobians etc, `cyipopt` provides the function `minimize_ipopt` which has the same behaviour as `scipy.optimize.minimize`, for example:

```
>>> from scipy.optimize import rosen, rosen_der
>>> from cyipopt import minimize_ipopt
>>> x0 = [1.3, 0.7, 0.8, 1.9, 1.2]
>>> res = minimize_ipopt(rosen, x0, jac=rosen_der)
>>> print(res)
    fun: 2.1256746564022273e-18
   info: {'x': array([1., 1., 1., 1., 1.]), 'g': array([], dtype=float64), 'obj_val': 2.
↪1256746564022273e-18, 'mult_g': array([], dtype=float64), 'mult_x_L': array([0., 0., 0.
↪, 0., 0.]), 'mult_x_U': array([0., 0., 0., 0., 0.]), 'status': 0, 'status_msg': b
↪'Algorithm terminated successfully at a locally optimal point, satisfying the␣
↪convergence tolerances (can be specified by options).'}
 message: b'Algorithm terminated successfully at a locally optimal point, satisfying the␣
↪convergence tolerances (can be specified by options).'
   nfev: 200
    nit: 37
   njev: 39
 status: 0
success: True
      x: array([1., 1., 1., 1., 1.])
```

2.1.1 Algorithmic Differentation

Computing derivatives by hand can be quite error-prone. In case you don't provide the (exact) objective gradient or the jacobian of the constraint function, the scipy interface will approximate the missing derivatives by finite differences similar to `scipy.optimize.minimize`. However, finite differences are prone to truncation errors due to floating point arithmetic and computationally expensive especially for evaluating jacobians. A more efficient and accurate way to evaluate derivatives is algorithmic differentation (AD).

In this example we use AD by means of the JAX library to compute derivatives and we use cyipopt's scipy interface to

solve an example problem, namely number 71 from the Hock-Schittkowsky test suite[1],

$$\begin{aligned} \min_{x \in \mathbb{R}^4} \; & x_1 x_4 (x_1 + x_2 + x_3) + x_3 \\ s.t. \; & x_1 x_2 x_3 x_4 \geq 25 \\ & x_1^2 + x_2^2 + x_3^2 + x_4^2 = 40 \\ & 1 \leq x_1, x_2, x_3, x_4 \leq 5, \end{aligned}$$

with the starting point,

$$x_0 = (1,\ 5,\ 5,\ 1),$$

and the optimal solution,

$$x_* = (1.0,\ 4.743,\ 3.821,\ 1.379)$$

We start by importing all required libraries:

```
from jax.config import config

# Enable 64 bit floating point precision
config.update("jax_enable_x64", True)

# We use the CPU instead of GPU und mute all warnings if no GPU/TPU is found.
config.update('jax_platform_name', 'cpu')

import jax.numpy as np
from jax import jit, grad, jacfwd
from cyipopt import minimize_ipopt
```

Then we define the objective and constraint functions:

```
def objective(x):
    return x[0]*x[3]*np.sum(x[:3]) + x[2]

def eq_constraints(x):
    return np.sum(x**2) - 40

def ineq_constrains(x):
    return np.prod(x) - 25
```

Next, we build the derivatives and just-in-time (jit) compile the functions (more details regarding `jit`, `grad` and `jacfwd` can be found in the JAX autodiff cookbook):

```
# jit the functions
obj_jit = jit(objective)
con_eq_jit = jit(eq_constraints)
con_ineq_jit = jit(ineq_constrains)

# build the derivatives and jit them
obj_grad = jit(grad(obj_jit))  # objective gradient
obj_hess = jit(jacrev(jacfwd(obj_jit))) # objective hessian
```

(continues on next page)

[1] W. Hock and K. Schittkowski. Test examples for nonlinear programming codes. Lecture Notes in Economics and Mathematical Systems, 187, 1981.

(continued from previous page)

```
con_eq_jac = jit(jacfwd(con_eq_jit))  # jacobian
con_ineq_jac = jit(jacfwd(con_ineq_jit))  # jacobian
con_eq_hess = jacrev(jacfwd(con_eq_jit)) # hessian
con_eq_hessvp = jit(lambda x, v: con_eq_hess(x) * v[0]) # hessian vector-product
con_ineq_hess = jacrev(jacfwd(con_ineq_jit))  # hessian
con_ineq_hessvp = jit(lambda x, v: con_ineq_hess(x) * v[0]) # hessian vector-product
```

Finally, we can call `minimize_ipopt` similar to `scipy.optimize.minimize`:

```
# constraints
cons = [
    {'type': 'eq', 'fun': con_eq_jit, 'jac': con_eq_jac, 'hess': con_eq_hessvp},
    {'type': 'ineq', 'fun': con_ineq_jit, 'jac': con_ineq_jac, 'hess': con_ineq_hessvp}
 ]

# starting point
x0 = np.array([1.0, 5.0, 5.0, 1.0])

# variable bounds: 1 <= x[i] <= 5
bnds = [(1, 5) for _ in range(x0.size)]

# executing the solver
res = minimize_ipopt(obj_jit, jac=obj_grad, hess=obj_hess, x0=x0, bounds=bnds,
                     constraints=cons, options={'disp': 5})
```

2.2 Problem Interface

In this example we will use cyipopt problem class interface to solve the aforementioned test problem.

2.2.1 Getting started

Before you can use cyipopt, you have to import it:

```
import cyipopt
```

This problem will also make use of NumPy:

```
import numpy as np
```

2.2.2 Defining the problem

The first step is to define a class that computes the objective and its gradient, the constraints and its Jacobian, and the Hessian. The following methods can be defined on the class:

- `cyipopt.Problem.objective()`
- `cyipopt.Problem.gradient()`
- `cyipopt.Problem.constraints()`
- `cyipopt.Problem.jacobian()`

- `cyipopt.Problem.hessian()`

The `cyipopt.Problem.jacobian()` and `cyipopt.Problem.hessian()` methods should return the non-zero values of the respective matrices as flattened arrays. The hessian should return a flattened lower triangular matrix.

The Jacobian and Hessian can be dense or sparse. If sparse, you must also define:

- `cyipopt.Problem.jacobianstructure()`
- `cyipopt.Problem.hessianstructure()`

which should return a tuple of indices that indicate the location of the non-zero values of the Jacobian and Hessian matrices, respectively. If not defined then these matrices are assumed to be dense.

The `cyipopt.Problem.intermediate()` method is called every Ipopt iteration algorithm and can be used to perform any needed computation at each iteration.

Define the problem class:

```
class HS071():

    def objective(self, x):
        """Returns the scalar value of the objective given x."""
        return x[0] * x[3] * np.sum(x[0:3]) + x[2]

    def gradient(self, x):
        """Returns the gradient of the objective with respect to x."""
        return np.array([
            x[0]*x[3] + x[3]*np.sum(x[0:3]),
            x[0]*x[3],
            x[0]*x[3] + 1.0,
            x[0]*np.sum(x[0:3])
        ])

    def constraints(self, x):
        """Returns the constraints."""
        return np.array((np.prod(x), np.dot(x, x)))

    def jacobian(self, x):
        """Returns the Jacobian of the constraints with respect to x."""
        return np.concatenate((np.prod(x)/x, 2*x))

    def hessianstructure(self):
        """Returns the row and column indices for non-zero vales of the
        Hessian."""

        # NOTE: The default hessian structure is of a lower triangular matrix,
        # therefore this function is redundant. It is included as an example
        # for structure callback.

        return np.nonzero(np.tril(np.ones((4, 4))))

    def hessian(self, x, lagrange, obj_factor):
        """Returns the non-zero values of the Hessian."""

        H = obj_factor*np.array((
            (2*x[3], 0, 0, 0),
```

(continues on next page)

(continued from previous page)

```
            (x[3],   0, 0, 0),
            (x[3],   0, 0, 0),
            (2*x[0]+x[1]+x[2], x[0], x[0], 0)))

        H += lagrange[0]*np.array((
            (0, 0, 0, 0),
            (x[2]*x[3], 0, 0, 0),
            (x[1]*x[3], x[0]*x[3], 0, 0),
            (x[1]*x[2], x[0]*x[2], x[0]*x[1], 0)))

        H += lagrange[1]*2*np.eye(4)

        row, col = self.hessianstructure()

        return H[row, col]

    def intermediate(self, alg_mod, iter_count, obj_value, inf_pr, inf_du, mu,
                     d_norm, regularization_size, alpha_du, alpha_pr,
                     ls_trials):
        """Prints information at every Ipopt iteration."""

        msg = "Objective value at iteration #{:d} is - {:g}"

        print(msg.format(iter_count, obj_value))
```

Now define the lower and upper bounds of x and the constraints:

```
lb = [1.0, 1.0, 1.0, 1.0]
ub = [5.0, 5.0, 5.0, 5.0]

cl = [25.0, 40.0]
cu = [2.0e19, 40.0]
```

Define an initial guess:

```
x0 = [1.0, 5.0, 5.0, 1.0]
```

Define the full problem using the *cyipopt.Problem* class:

```
nlp = cyipopt.Problem(
   n=len(x0),
   m=len(cl),
   problem_obj=HS071(),
   lb=lb,
   ub=ub,
   cl=cl,
   cu=cu,
)
```

The constructor of the *cyipopt.Problem* class requires:

- `n`: the number of variables in the problem,
- `m`: the number of constraints in the problem,

- `lb` and `ub`: lower and upper bounds on the variables,
- `cl` and `cu`: lower and upper bounds of the constraints.
- `problem_obj` is an object whose methods implement `objective`, `gradient`, `constraints`, `jacobian`, and `hessian` of the problem.

2.2.3 Setting optimization parameters

Setting optimization parameters is done by calling the *cyipopt.Problem.add_option()* method, e.g.:

```
nlp.add_option('mu_strategy', 'adaptive')
nlp.add_option('tol', 1e-7)
```

The different options and their possible values are described in the ipopt documentation.

2.2.4 Executing the solver

The optimization algorithm is run by calling the *cyipopt.Problem.solve()* method, which accepts the starting point for the optimization as its only parameter:

```
x, info = nlp.solve(x0)
```

The method returns the optimal solution and an info dictionary that contains the status of the algorithm, the value of the constraints multipliers at the solution, and more.

2.3 Where to go from here

Once you feel sufficiently familiar with the basics, feel free to dig into the *reference*. For more examples, check the `examples/` subdirectory of the distribution.

CHAPTER

THREE

REFERENCE

This is the class and function reference of cyipopt. Please refer to the *tutorial* for further details, as the class and function raw specifications may not be enough to give full guidelines on their uses.

class cyipopt.**Problem**

Wrapper class for solving optimization problems using the C interface of the Ipopt package.

It can be used to solve general nonlinear programming problems of the form:

$$\min_{x \in R^n} f(x)$$

subject to

$$g_L \leq g(x) \leq g_U$$
$$x_L \leq x \leq x_U$$

Where x are the optimization variables (possibly with upper an lower bounds), $f(x)$ is the objective function and $g(x)$ are the general nonlinear constraints. The constraints, $g(x)$, have lower and upper bounds. Note that equality constraints can be specified by setting $g_L^i = g_U^i$.

Parameters

- **n** (*integer*) – Number of primal variables.
- **m** (*integer*) – Number of constraints.
- **problem_obj** (*object, optional (default=None)*) – An object holding the problem's callbacks. If None, cyipopt will use `self`, this is useful when subclassing `Problem`. The object is required to have the following attributes and methods (some are optional):

  –objective
  [function pointer] Callback function for evaluating objective function. The callback functions accepts one parameter: x (value of the optimization variables at which the objective is to be evaluated). The function should return the objective function value at the point x.

  –constraints
  [function pointer] Callback function for evaluating constraint functions. The callback functions accepts one parameter: x (value of the optimization variables at which the constraints are to be evaluated). The function should return the constraints values at the point x.

  –gradient
  [function pointer] Callback function for evaluating gradient of objective function. The callback functions accepts one parameter: x (value of the optimization variables at which the gradient is to be evaluated). The function should return the gradient of the objective function at the point x.

–jacobian
[function pointer] Callback function for evaluating Jacobian of constraint functions. The callback functions accepts one parameter: x (value of the optimization variables at which the Jacobian is to be evaluated). The function should return the values of the Jacobian as calculated using x. The values should be returned as a 1-dim numpy array (using the same order as you used when specifying the sparsity structure)

–jacobianstructure
[function pointer, optional (default=None)] Callback function that accepts no parameters and returns the sparsity structure of the Jacobian (the row and column indices only). If None, the Jacobian is assumed to be dense.

–hessian
[function pointer, optional (default=None)] Callback function for evaluating Hessian of the Lagrangian function. The callback functions accepts three parameters x (value of the optimization variables at which the Hessian is to be evaluated), lambda (values for the constraint multipliers at which the Hessian is to be evaluated) objective_factor the factor in front of the objective term in the Hessian. The function should return the values of the Hessian as calculated using x, lambda and objective_factor. The values should be returned as a 1-dim numpy array (using the same order as you used when specifying the sparsity structure). If `None`, the Hessian is calculated numerically.

–hessianstructure
[function pointer, optional (default=None)] Callback function that accepts no parameters and returns the sparsity structure of the Hessian of the lagrangian (the row and column indices only). If `None`, the Hessian is assumed to be dense.

–intermediate
[function pointer, optional (default=None)] Optional. Callback function that is called once per iteration (during the convergence check), and can be used to obtain information about the optimization status while Ipopt solves the problem. If this callback returns False, Ipopt will terminate with the `User_Requested_Stop status`. The information below corresponeds to the argument list passed to this callback:

alg_mod:
Algorithm phase: 0 is for regular, 1 is restoration.

iter_count:
The current iteration count.

obj_value:
The unscaled objective value at the current point

inf_pr:
The scaled primal infeasibility at the current point.

inf_du:
The scaled dual infeasibility at the current point.

mu:
The value of the barrier parameter.

d_norm:
The infinity norm (max) of the primal step.

regularization_size:
The value of the regularization term for the Hessian of the Lagrangian in the augmented system.

alpha_du:
The stepsize for the dual variables.

alpha_pr:
The stepsize for the primal variables.

ls_trials:
The number of backtracking line search steps.

more information can be found in the following link: https://coin-or.github.io/Ipopt/OUTPUT.html

- **lb** (*array-like, shape(n,)*) – Lower bounds on variables, where n is the dimension of x. To assume no lower bounds pass values lower then `10^-19`.
- **ub** (*array-like, shape(n,)*) – Upper bounds on variables, where n is the dimension of x. To assume no upper bounds pass values higher then `10^-19`.
- **cl** (*array-like, shape(m,)*) – Lower bounds on constraints, where m is the number of constraints. Equality constraints can be specified by setting `cl[i] = cu[i]`.
- **cu** (*array-like, shape(m,)*) – Upper bounds on constraints, where m is the number of constraints. Equality constraints can be specified by setting `cl[i] = cu[i]`.

addOption()

Add a keyword/value option pair to the problem.

Deprecated since version 1.0.0: *addOption()* will be removed in CyIpopt 1.1.0, it is replaced by *add_option()* because the latter complies with PEP8.

add_option()

Add a keyword/value option pair to the problem.

See the Ipopt documentaion for details on available options.

Parameters

- **keyword** (*str*) – Option name.
- **val** (*str, int or float*) – Value of the option. The type of val should match the option definition as described in the Ipopt documentation.

close()

Deallocate memory resources used by the Ipopt package.

Called implicitly by the *Problem* class destructor.

setProblemScaling()

Optional function for setting scaling parameters for the problem.

Deprecated since version 1.0.0: *setProblemScaling()* will be removed in CyIpopt 1.1.0, it is replaced by *set_problem_scaling()* because the latter complies with PEP8.

set_problem_scaling()

Optional function for setting scaling parameters for the problem.

To use the scaling parameters set the option `nlp_scaling_method` to `user-scaling`.

Parameters

- **obj_scaling** (*float*) – Determines, how Ipopt should internally scale the objective function. For example, if this number is chosen to be 10, then Ipopt solves internally an optimization problem that has 10 times the value of the original objective. In particular, if this value is negative, then Ipopt will maximize the objective function instead of minimizing it.

- **x_scaling** (*array-like, shape(n,)*) – The scaling factors for the variables. If None, no scaling is done.
- **g_scaling** (*array-like, shape(m,)*) – The scaling factors for the constrains. If None, no scaling is done.

solve()

Returns the optimal solution and an info dictionary.

Solves the posed optimization problem starting at point x.

Parameters
x (*array-like, shape(n,)*) – Initial guess.

Returns

- **x** (*array, shape(n,)*) – Optimal solution.
- **info** (*dictionary*) –

x: ndarray, shape(n,)
optimal solution

g: ndarray, shape(m,)
constraints at the optimal solution

obj_val: float
objective value at optimal solution

mult_g: ndarray, shape(m,)
final values of the constraint multipliers

mult_x_L: ndarray, shape(n,)
bound multipliers at the solution

mult_x_U: ndarray, shape(n,)
bound multipliers at the solution

status: integer
gives the status of the algorithm

status_msg: string
gives the status of the algorithm as a message

class cyipopt.**problem**(**args*, ***kwargs*)

Class to continue support for old API.

Deprecated since version 1.0.0: *problem* will be removed in CyIpopt 1.1.0, it is replaced by *Problem* because the latter complies with PEP8.

For full documentation of this class including its attributes and methods please see *Problem*.

This class acts as a wrapper to the new *Problem* class. It simply issues a FutureWarning to the user before passing all args and kwargs through to *Problem*.

Returns
Instance created with the *args* and *kwargs* parameters.

Return type
Problem

cyipopt.**minimize_ipopt**(*fun*, *x0*, *args=()*, *kwargs=None*, *method=None*, *jac=None*, *hess=None*, *hessp=None*, *bounds=None*, *constraints=()*, *tol=None*, *callback=None*, *options=None*)

Minimize a function using ipopt. The call signature is exactly like for `scipy.optimize.mimize`. In options, all options are directly passed to ipopt. Check [http://www.coin-or.org/Ipopt/documentation/node39.html] for details. The options `disp` and `maxiter` are automatically mapped to their ipopt-equivalents `print_level` and `max_iter`.

`cyipopt.`**`set_logging_level`**`()`

Set the logger verbosity to the specified level.

Parameters

`level` (*int*) – The verbosity of the logger. This threshold is used to determine which logging messages are logged by this module's `log()` function.

`cyipopt.`**`setLoggingLevel`**`()`

Function to continue support for old API.

Deprecated since version 1.0.0: *setLoggingLevel()* will be removed in CyIpopt 1.1.0, it is replaced by *set_logging_level()* because the latter complies with PEP8.

For full documentation of this function please see *set_logging_level()*.

This function acts as a wrapper to the new *set_logging_level()* function. It simply issues a `FutureWarning` to the user before passing all args and kwargs through to *set_logging_level()*.

CHAPTER

FOUR

DEVELOPMENT

4.1 Development Install

Clone the repository:

```
$ git clone git@github.com:mechmotum/cyipopt.git
$ cd cyipopt
```

Create a Conda environment with the dependencies:

```
$ conda env create -f conda/cyipopt-dev.yml
```

Activate the environment:

```
$ conda activate cyipopt-dev
```

Install a development version[1]:

```
(cyipopt-dev)$ python setup.py develop
```

4.2 Building the documentation

After installing the development version of cyipopt, navigate to a directory that contains the source code and execute the `Makefile`:

```
(cyipopt-dev)$ cd docs
(cyipopt-dev)$ make html
```

Once the build process finishes, direct your web browser to `build/html/index.html`.

[1] Changes to any of the Cython files require calling `python setup.py develop` to see effects of the changes.

4.3 Testing

You can test the installation by running each of the examples in the `examples/` directory and running the test suite. The tests can be run with:

```
(cyipopt-dev)$ pytest
```

CHAPTER

FIVE

INDICES AND TABLES

- genindex
- modindex
- search

CHAPTER

SIX

COPYRIGHT

INDEX