PETSc for Python

Release 3.18.0

Lisandro Dalcin

Sep 27, 2022

Contents

Abstract

This document describes petsc4py, a Python port to the PETSc libraries.

PETSc (the Portable, Extensible Toolkit for Scientific Computation) is a suite of data structures and routines for the scalable (parallel) solution of scientific applications modeled by partial differential equations. It employs the MPI standard for all message-passing communication.

This package provides an important subset of PETSc functionalities and uses NumPy to efficiently manage input and output of array data.

A *good friend* of petsc4py is:

- mpi4py: Python bindings for MPI, the *Message Passing Interface*.

Other projects depends on petsc4py:

- slepc4py: Python bindings for SLEPc, the *Scalable Library for Eigenvalue Problem Computations*.

1 Overview

PETSc is a suite of data structures and routines for the scalable (parallel) solution of scientific applications modeled by partial differential equations. It employs the MPI standard for all message-passing communication.

PETSc is intended for use in large-scale application projects [petsc-efficient], and several ongoing computational science projects are built around the PETSc libraries. With strict attention to component interoperability, PETSc facilitates the integration of independently developed application modules, which often most naturally employ different coding styles and data structures.

PETSc is easy to use for beginners [petsc-user-ref]. Moreover, its careful design allows advanced users to have detailed control over the solution process. PETSc includes an expanding suite of parallel linear and nonlinear equation solvers that are easily used in application codes written in C, C++, and Fortran. PETSc provides many of the mechanisms needed within parallel application codes, such as simple parallel matrix and vector assembly routines that allow the overlap of communication and computation.

1.1 Components

PETSc is designed with an object-oriented style. Almost all user-visible types are abstract interfaces with implementations that may be chosen at runtime. Those objects are managed through handles to opaque data structures which are created, accessed and destroyed by calling appropriate library routines.

PETSc consists of a variety of components. Each component manipulates a particular family of objects and the operations one would like to perform on these objects. These components provide the functionality required for many parallel solutions of PDEs.

Vec Provides the vector operations required for setting up and solving large-scale linear and nonlinear problems. Includes easy-to-use parallel scatter and gather operations, as well as special-purpose code for handling ghost points for regular data structures.

Mat A large suite of data structures and code for the manipulation of parallel sparse matrices. Includes four different parallel matrix data structures, each appropriate for a different class of problems.

PC A collection of sequential and parallel preconditioners, including (sequential) ILU(k), LU, and (both sequential and parallel) block Jacobi, overlapping additive Schwarz methods and (through BlockSolve95) ILU(0) and ICC(0).

KSP Parallel implementations of many popular Krylov subspace iterative methods, including GMRES, CG, CGS, Bi-CG-Stab, two variants of TFQMR, CR, and LSQR. All are coded so that they are immediately usable with any preconditioners and any matrix data structures, including matrix-free methods.

SNES Data-structure-neutral implementations of Newton-like methods for nonlinear systems. Includes both line search and trust region techniques with a single interface. Employs by default the above data structures and linear solvers. Users can set custom monitoring routines, convergence criteria, etc.

TS Code for the time evolution of solutions of PDEs. In addition, provides pseudo-transient continuation techniques for computing steady-state solutions.

2 Installation

2.1 Using pip

You can use **pip** to install petsc4py and its dependencies (mpi4py is optional but highly recommended):

```
$ python -m pip install [--user] numpy mpi4py  (or pip install [--user] numpy mpi4py)
$ python -m pip install [--user] petsc petsc4py (or pip install [--user] petsc petsc4py)
```

2.2 Using setuptools

You can also install dependencies manually and then invoke setuptools from the *petsc4py* source directory:

$ python setup.py build $ python setup.py install

You may use the *–install-lib* argument to the *install* command to alter the *site-packages* directory where the package is to be installed.

If you are cross-compiling, and the *numpy* module cannot be loaded on your build host, then before invoking *setup.py*, set *NUMPY_INCLUDE* environment variable to the path that would be returned by *import numpy; numpy.get_include()*:

$ export NUMPY_INCLUDE=/usr/lib/pythonX/site-packages/numpy/core/include

2.3 From PETSc source

If you already have downloaded PETSc source and have installed the dependencies of *petsc4py*, then to build the *petsc4py* module along with PETSc, add the *–with-petsc4py=1* argument to the configure command when building PETSc:

$ ./configure –with-petsc4py=1 $ make $ make install

This will install PETSc and the *petsc4py* module into the PETSc directory under the prefix specified to the PETSc configure command.

If you wish to make the module importable without having to set the *PYTHONPATH* environment variable, you may add a shortcut to the system-wide *site-packages* directory creating a special *.pth* file with exactly one line of Python code. This can be done by the following command, where the system-wide path is assumed to be */usr/lib/pythonX/site-packages* (replace *X* with your python version):

$ echo “import sys, os;” “p = os.getenv(‘PETSC_DIR’);” “a = os.getenv(‘PETSC_ARCH’) or ‘’;” “p = p and os.path.join(p, a, ‘lib’);” “p and (p in sys.path or sys.path.append(p))” > /usr/lib/pythonX/site-packages/petsc4py.pth

If you are cross-compiling, and *numpy* cannot be loaded on your build host, then pass *–have-numpy=1 –with-numpy-include=PATH*, where *PATH* is the path that would be returned by *import numpy; print(numpy.get_include())*. This will suppress autodetection of the include path on the build host.

3 Tutorial

XXX To be written … Any contribution welcome!

4 Citations

If PETSc for Python been significant to a project that leads to an academic publication, please acknowledge that fact by citing the project.

- L. Dalcin, P. Kler, R. Paz, and A. Cosimo, *Parallel Distributed Computing using Python*, Advances in Water Resources, 34(9):1124-1139, 2011. http://dx.doi.org/10.1016/j.advwatres.2011.04.013
- S. Balay, S. Abhyankar, M. Adams, S. Benson, J. Brown, P. Brune, K. Buschelman, E. Constantinescu, L. Dalcin, A. Dener, V. Eijkhout, W. Gropp, V. Hapla, T. Isaac, J. Faibussowitsch, P. Jolivet, D. Karpeyev, D. Kaushik, M. Knepley, F. Kong, S. Kruger, D. May, L. Curfman McInnes, R. Mills, L. Mitchell, T. Munson, J. Roman, K. Rupp, P. Sanan, J Sarich, B. Smith, S. Zampini, H. Zhang, and H. Zhang, J. Zhang, *PETSc Users Manual*, ANL-21/39 - Revision 3.18, 2022. https://petsc.org/release/docs/manual/manual.pdf

References

[petsc-user-ref] S. Balay, S. Abhyankar, M. Adams, S. Benson, J. Brown, P. Brune, K. Buschelman, E. Constantinescu, L. Dalcin, A. Dener, V. Eijkhout, W. Gropp, V. Hapla, T. Isaac, J. Faibussowitsch, P. Jolivet, D. Karpeyev, D. Kaushik, M. Knepley, F. Kong, S. Kruger, D. May, L. Curfman McInnes, R. Mills, L. Mitchell, T. Munson, J. Roman, K. Rupp, P. Sanan, J Sarich, B. Smith, S. Zampini, H. Zhang, and H. Zhang, J. Zhang, *PETSc Users Manual*, ANL-21/39 - Revision 3.18, 2022. https://petsc.org/release/docs/manual/manual.pdf

[petsc-efficient] Satish Balay, Victor Eijkhout, William D. Gropp, Lois Curfman McInnes and Barry F. Smith. Efficient Management of Parallelism in Object Oriented Numerical Software Libraries. Modern Software Tools in Scientific Computing. E. Arge, A. M. Bruaset and H. P. Langtangen, editors. 163–202. Birkhauser Press. 1997.