PyOrder Documentation

Release 0.1.0

Dominique Orban

July 23, 2009

CONTENTS

1 Introduction **3**
- 1.1 A Few Concepts about Sparse Matrices . . . 3
- 1.2 Availability . . . 3

2 Bringing Nonzero Elements to the Diagonal **5**
- 2.1 The `pymc21` Module . . . 5
- 2.2 Examples . . . 5

3 Profile, Wavefront, Bandwidth Reduction **7**
- 3.1 The `pymc60` Module . . . 7
- 3.2 Examples . . . 8

4 Input / Output **11**
- 4.1 The `hrb` Module . . . 11

5 Indices and tables **13**

Module Index **15**

Index **17**

Contents:

CHAPTER

ONE

INTRODUCTION

PyOrder is a Python package that provides access to several means to reorder sparse matrices. Typically, sparse matrices are reordered prior to factorization so as to preserve sparsity of the factors, ensure that there are as many nonzero elements as possible on the diagonal, or decrease the envelope, wavefront, profile or semi-bandwidth.

Note: Orderings designed to promote sparsity of the factors of a matrix are different in nature from the other types of orderings mentioned in the previous paragraph and are not considered in PyOrder. If you would like to see such orderings (e.g., AMD, CAMD, COLAMD, etc.) included in future releases of PyOrder, please let me know.

1.1 A Few Concepts about Sparse Matrices

1.2 Availability

PyOrder is essentially a set of interfaces to subroutines from the HSL http://www.cse.scitech.ac.uk/nag/hsl (formerly the Harwell Subroutine Library). Subroutines from the HSL may be obtained free of charge under certain conditions. See http://hsl.rl.ac.uk/hsl2007/hsl20074researchers.html to decide whether the terms apply to you. The HSL subroutines relevant to PyOrder are not packaged together with the Python interfaces and should be obtained separately.

Note: At this time, only double precision real data is supported. Please let me know if you would like support for single precision and/or complex data.

CHAPTER

TWO

BRINGING NONZERO ELEMENTS TO THE DIAGONAL

2.1 The `pymc21` Module

Compute a row permutation so as to bring nonzero elements on the diagonal of a square sparse matrix in compressed sparse row (csr) format. This module provides an interface to the HSL subroutine MC21.

`nonzerodiag`(*nrow, colind, rowptr*)

Given the sparsity pattern of a square sparse matrix in compressed row (csr) format, attempt to find a *row* permutation so the row-permuted matrix has a nonzero diagonal, if this is possible. This function assumes that the matrix indexing is 0-based.

Parameters

- **nrow** The number of rows of the input matrix.
- **colind** An integer array (or list) of length nnz giving the column indices of the nonzero elements in each row.
- **rowptr** An integer array (or list) of length nrow+1 giving the indices of the first element of each row in colind.

Returns values

- **perm** An integer array of length nrow giving the variable permutation. If irow and jcol are two integer arrays describing the pattern of the input matrix in triple format, perm[irow] and jcol describe the permuted matrix.
- **nzdiag** The number of nonzeros on the diagonal of the permuted matrix.

2.2 Examples

2.2.1 Basic Usage

This first example calls the Fortran subroutine directly with the matrix data in compressed sparse column format.

> **Warning:** Keep in mind that when calling the Fortran subroutines directly, all indices in the matrix data must be 1-based, i.e., row indices range from 1 through nrow and column indices range from 1 through ncol.

```
import numpy as np
from pyorder.pymc21 import mc21module
n = 4
icn = np.array([1,4,3,4,1,4,2,4], dtype=np.int32)
```

```
ip = np.array([1,3,5,7], dtype=np.int32)
lenr = np.array([2,2,2,2], dtype=np.int32)
iperm,numnz = mc21module.mc21ad(icn,ip,lenr)
print 'iperm = ', iperm, ' (1-based)'
print 'numnz = ', numnz
```

2.2.2 Python Interface

In this second example, Python usage is intended. Matrix indices must therefore be 0-based. The permutation vector returned by `nonzerdiag()` is now also 0-based.

```
"""
Illustrate usage of the pymc21 module, using an input matrix in Harwell-Boeing
or Rutherford-Boeing format. Supply a file name as input argument on the command
line and uncomment below as appropriate.
"""

import sys
import numpy as np
from pyorder.tools.hrb import HarwellBoeingMatrix, RutherfordBoeingData
from pyorder.pymc21 import nonzerodiag
from sparsetools import FastSpy
import pylab

if len(sys.argv) < 2:
    sys.stderr.write('Supply input matrix as argument\n')
    sys.exit(1)

fname = sys.argv[1]
M = HarwellBoeingMatrix(fname, patternOnly=True, readRhs=False)
#M = RutherfordBoeingData(fname, patternOnly=True, readRhs=False)

if M.nrow != M.ncol:
    sys.stderr.write('Input matrix must be square\n')
    sys.exit(1)

perm, nzdiag = nonzerodiag(M.nrow, M.ind, M.ip)

(irow, jcol) = M.find()
left = pylab.subplot(121)
FastSpy(M.nrow, M.ncol, irow, jcol, sym=M.issym, ax=left.get_axes())

right = pylab.subplot(122)
FastSpy(M.nrow, M.ncol, perm[irow], jcol, sym=M.issym, ax=right.get_axes())
pylab.show()
```

CHAPTER

THREE

PROFILE, WAVEFRONT, BANDWIDTH REDUCTION

3.1 The `pymc60` Module

Profile/wavefront/semi-bandwidth reduction by way of Sloan's and the reverse Cuthill-McKee algorithms. This module provides an interface to the HSL subroutine MC60.

sloan (*n, rowind, colptr, icntl=, [0, 6], weight=, [2, 1]*)

Apply Sloan's algorithm to reduce the profile and wavefront of a sparse symmetric matrix. Either the lower or the upper triangle of the input matrix should be given in compressed sparse column (csc) or compressed sparse row (csr) format. This includes the diagonal of the matrix. A set of weights can be supplied to define the priority function in Sloan's method.

Parameters

- **n** The order of the input matrix.
- **rowind** An integer array (or list) of length nnz giving the row indices of the nonzero elements in each column.
- **colptr** An integer array (or list) of length n+1 giving the indices of the first element of each column in rowind.

Note that since either triangle can be given in either csc or csr format, the words 'row' and 'column' may be swapped in the description above. The indexing in rowind and colptr should be zero-based.

Keywords

- **icntl** An integer array (or list) of length two of control parameters used during the first phase, where the input data is checked. The method terminates if duplicates of out-of-range indices are discovered (icntl[0]=0) or ignores them (icntl[0]=1). No diagnostic messages will be output if icntl[1]=0. If icntl[1] is > 0, it gives the unit number (in the Fortran sense) where diagonostic messages are output.
- **weight** An integer array (or list) of length two giving the weights in Sloan's priority function. Reid and Scott (1999) recommend to apply the method twice, with either [2,1] and [16,1], or with [1,2] and [16,1], and to retain the best result.

Returns values

- **perm** An integer array of length n giving the variable permutation. If irow and jcol are two integer arrays describing the pattern of the input matrix in triple format, perm[irow] and perm[jrow] describe the permuted matrix.
- **rinfo** A real array of length 4 giving statistics on the permuted matrix. rinfo[0] = profile rinfo[1] = maximum wavefront rinfo[2] = semi-bandwidth rinfo[3] = root-mean-square wavefront.

rcmk (*n, rowind, colptr, icntl=, [0, 6]*)

Apply the reverse Cuthill-McKee algorithm to reduce the bandwidth of a sparse symmetric matrix. Either the lower or the upper triangle of the input matrix should be given in compressed sparse column (csc) or compressed sparse row (csr) format. This includes the diagonal of the matrix.

Parameters

- **n** The order of the input matrix.
- **rowind** An integer array (or list) of length nnz giving the row indices of the nonzero elements in each column.
- **colptr** An integer array (or list) of length n+1 giving the indices of the first element of each column in rowind.

Note that since either triangle can be given in either csc or csr format, the words 'row' and 'column' may be swapped in the description above. The indexing in rowind and colptr should be zero-based.

Keywords

- **icntl** An integer array (or list) of length two of control parameters used during the first phase, where the input data is checked. The method terminates if duplicates of out-of-range indices are discovered (icntl[0]=0) or ignores them (icntl[0]=1). No diagnostic messages will be output if icntl[1]=0. If icntl[1] is > 0, it gives the unit number (in the Fortran sense) where diagonostic messages are output.

Returns values

- **perm** An integer array of length n giving the variable permutation. If irow and jcol are two integer arrays describing the pattern of the input matrix in triple format, perm[irow] and perm[jcol] describe the permuted matrix.
- **rinfo** A real array of length 4 giving statistics on the permuted matrix. rinfo[0] = profile rinfo[1] = maximum wavefront rinfo[2] = semi-bandwidth rinfo[3] = root-mean-square wavefront.

reorder_matrix(*n, rowind, colptr, icntl=, [0, 6], jcntl=, [0, 0], weight=, [2, 1]*)

Helper function called by *sloan* and *rcm* performing the bulk of the work when applying Sloan's method or the reverse Cuthill-McKee algorithm to a symmetric sparse matrix.

3.2 Examples

3.2.1 Basic Usage

The first example is the one from the documentation of the HSL subroutine MC60. In it, we call the Fortran subroutines directly. As before, this means that all indices must be 1-based. The permutation vector and indices of the supervariables are also 1-based.

```python
"MC60 demo from the HSL MC60 spec sheet"
import numpy as np
from pyorder.pymc60 import mc60module

icntl = np.array([0,6], dtype=np.int32) # Abort on error
jcntl = np.array([0,0], dtype=np.int32) # Sloan's alg with auto choice
weight = np.array([2,1])                # Weights in Sloan's alg

# Store lower triangle of symmetric matrix in csr format (1-based)
n = 5
icptr = np.array([1,6,8,9,10,11], dtype=np.int32)  # nnz = 10
irn = np.empty(2*(icptr[-1]-1), dtype=np.int32)
irn[:icptr[-1]-1] = np.array([1,2,3,4,5,2,3,3,4,5], dtype=np.int32)

# Check data
info = mc60module.mc60ad(irn, icptr, icntl)

```

```
# Compute supervariables
nsup, svar, vars = mc60module.mc60bd(irn, icptr)
print 'The number of supervariables is ', nsup

# Permute reduced matrix
permsv = np.empty(nsup, dtype=np.int32)
pair = np.empty((2, nsup/2), dtype=np.int32)
info = mc60module.mc60cd(n,irn,icptr[:nsup+1],vars[:nsup],jcntl,permsv,weight,pair)

# Compute profile and wavefront
rinfo = mc60module.mc60fd(n, irn, icptr[:nsup+1], vars[:nsup], permsv)

# Obtain variable permutation from supervariable permutation
perm, possv = mc60module.mc60dd(svar, vars[:nsup], permsv)
print 'The variable permutation is ', perm    # 1-based
print 'The profile is ', rinfo[0]
print 'The maximum wavefront is ', rinfo[1]
print 'The semibandwidth is ', rinfo[2]
print 'The root-mean-square wavefront is ', rinfo[3]
```

3.2.2 Python Interface

The Python interface provides smoother and more intuitive application of Sloan's and the reverse Cuthill-McKee methods by way of the `sloan()` and `rcmk()` functions. At the same time, we use input data in either Harwell-Boeing or Rutherford-Boeing format.

```
"MC60 demo with input matrix in HB or RB format"

import numpy as np
from pyorder.pymc60 import sloan, rcmk
from pyorder.tools.hrb import HarwellBoeingMatrix, RutherfordBoeingData
from sparsetools import FastSpy
import pylab
import sys

if len(sys.argv) < 2:
    sys.stderr.write('Data file name must be supplied\n')
    sys.exit(1)

fname = sys.argv[1]
#M = HarwellBoeingMatrix(fname, patternOnly=True, readRhs=False)
M = RutherfordBoeingData(fname, patternOnly=True, readRhs=False)

if M.nrow != M.ncol or not M.issym:
    sys.stderr.write('Input matrix must be square and symmetric\n')
    sys.exit(1)

# Compute reverse Cuthill-McKee ordering
perm, rinfo = rcmk(M.nrow, M.ind, M.ip)

# Or: Compute Sloan's ordering
#perm, rinfo = sloan(M.nrow, M.ind, M.ip)

# Plot original matrix
(irow, jcol) = M.find()
left = pylab.subplot(121)
```

```
FastSpy(M.nrow, M.ncol, irow, jcol, sym=M.issym,
        ax=left.get_axes(), title='Original')

# Apply permutation and plot reordered matrix
right = pylab.subplot(122)
FastSpy(M.nrow, M.ncol, perm[irow], perm[jcol], sym=M.issym,
        ax=right.get_axes(), title='Reordered')
pylab.show()
```

CHAPTER

FOUR

INPUT / OUTPUT

4.1 The `hrb` Module

Provides access to sparse linear systems described in Harwell-Boeing or Rutherford-Boeing format. This module exposes the two classes HarwellBoeingMatrix and RutherfordBoeingData. For more information, see the references below.

4.1.1 References

Dominique Orban, GERAD and Ecole Polytechnique de Montreal, 2007 <dominique.orban@gerad.ca>

class `HarwellBoeingMatrix`(*fname, **kwargs*)

Imports a sparse matrix from a file in Harwell-Boeing format. The matrix is stored in compressed sparse row format in (self.ind, self.ip, self.val). Right-hand sides, if any, are stored in self.rhs. Right-hand sides can be stored as dense vectors, in which case self.rhs has shape (nrow, nrhs), as sparse vectors, in which case they are stored in compressed sparse column format in (self.rhsptr, self.rhsind, self.rhs), or in elemental format (typically when the matrix itself is stored in finite-element format), in which case self.rhs has shape (nnzero, nrhs).

Note that the matrix indices are zero-based, i.e., row indices range from 0 through nrow-1 and column indices range from 0 through ncol-1.

The matrix can be subsequently converted to triple format with (row, col) = self.find()

Keywords **patternOnly** do not read matrix element values (False)

readRhs read right-hand sides, if any (False)

readGuess read starting guess, if any (False)

realSol read solution vector, if any (False)

`find`()

`fortranRead`(*stream, format*)

`readArray`(*fp, which, nelm, format*)

`readMatrix`(*fp, **kwargs*)

class `RutherfordBoeingData`(*fname, **kwargs*)

Bases: `pyorder.tools.hrb.HarwellBoeingMatrix`

Imports data from a file in Rutherford-Boeing format. The data is held in (self.ind, self.ip, self.val). If the data represents a sparse matrix, the three arrays represent the matrix stored in compressed sparse row format.

Otherwise, the three arrays represent the supplementary data. Refer to the Rutherford-Boeing documentation for more information (reference [4] in the docstring for the present module.)

Note that the matrix indices are zero-based, i.e., row indices range from 0 through nrow-1 and column indices range from 0 through ncol-1.

The data can be subsequently converted to triple format with (row, col) = self.find()

Currently accepted keyword arguments are:

patternOnly	do not read data values	(False)

find()

fortranRead(*stream, format*)

readArray(*fp, which, nelm, format*)

readMatrix(*fp, **kwargs*)

CHAPTER

FIVE

INDICES AND TABLES

- *Index*
- *Module Index*
- *Search Page*

MODULE INDEX

P

pyorder.pymc21, 5
pyorder.pymc60, 7
pyorder.tools.hrb, 11

INDEX

F

find() (pyorder.tools.hrb.HarwellBoeingMatrix method), 11
find() (pyorder.tools.hrb.RutherfordBoeingData method), 12
fortranRead() (pyorder.tools.hrb.HarwellBoeingMatrix method), 11
fortranRead() (pyorder.tools.hrb.RutherfordBoeingData method), 12

H

HarwellBoeingMatrix (class in pyorder.tools.hrb), 11

N

nonzerodiag() (in module pyorder.pymc21), 5

P

pyorder.pymc21 (module), 5
pyorder.pymc60 (module), 7
pyorder.tools.hrb (module), 11

R

rcmk() (in module pyorder.pymc60), 7
readArray() (pyorder.tools.hrb.HarwellBoeingMatrix method), 11
readArray() (pyorder.tools.hrb.RutherfordBoeingData method), 12
readMatrix() (pyorder.tools.hrb.HarwellBoeingMatrix method), 11
readMatrix() (pyorder.tools.hrb.RutherfordBoeingData method), 12
reorder_matrix() (in module pyorder.pymc60), 8
RutherfordBoeingData (class in pyorder.tools.hrb), 11

S

sloan() (in module pyorder.pymc60), 7