CONTENTS

CHAPTER

ONE

INTRODUCTION TO PYSPARSE

PySparse extends the Python interpreter by a set of sparse matrix types holding double precision values. PySparse also includes modules that implement

- Iterative Krylov methods for solving linear systems of equations,
- Diagonal (Jacobi) and SSOR preconditioners,
-

CHAPTER TWO

SPARSE MATRIX FORMATS

This section describes the sparse matrix storage schemes available in Pysparse. It also covers sparse matrix creation,

CHAPTER
THREE

3.1.2 ll_mat objects

ll_mat

`export_mtx(`*fileName, precision=6*`)`

Exports the matrix `A` to file named `fileName`. The matrix is stored in MatrixMarket Coordinate format. Depending on the properties of the `ll_mat` object `A` the generated file either uses the symmetric or a general MatrixMarket Coordinate format. The optional parameter `precision` specifies the number of decimal digits that are used to express the non-zero entries in the output file.

`shift(`*sigma, M*`)`

Performs the daxpy operation `A` `A +` `M`. The parameter is expected to be a Python Float object. The parameter `M` is expected to an object of type `ll_mat` of compatible shape.

`copy()`

Returns a new `ll_mat` object that is a (deep) ll_mat object A.

`csr_mat` and `sss_mat` Object Attributes and Methods

```
L.put(-e[1:], din[:-1], din[1:])
return L
```

```
for
```

It is striking to see how slow the straightforward `poisson2d` version is in Matlab. As we see in the next section, the

CHAPTER

FOUR

PRECONDITIONERS

4.1 The `precon` Module

CHAPTER

FIVE

ITERATIVE SOLVERS

relres the relative residual at the approximate solution computed by the iterative method. What this actually is depends on the actual iterative method used.

The iterative solvers may accept additional parameters, which are passed as keyword arguments.

The Matlab solution (without preconditioner) may look as follows:

```
n = 300;
L = poisson2d_kron(n);
[x,flag,relres,iter] = pcg(L, ones(n*n,1), 1e-12, 2000, ...
                           [], [], zeros(n*n,1));
```

od[po]T/F319.9626TJ07.62175TdUpdatTheTheas..2

5.1.3 Performance comparison with Matlab and nativ[-5C.3

`nnz`

The `nnz` attribute holds the total number of nonzero entries stored in both the `L` and `U` factors.

`solve`

```
    def precon(self, x, y):
        self.LU.solve(x,y)


n = .00
A = poisson.poisson2d_sym_blk(n).to_csr()   # Convert right away
b = numpy.ones(n*n)
x = numpy.empty(n*n)

K = ILU_Precon(A)
info, niter, relres = itsolvers.pcg(A, b, x, 1e-12, 2000, K)
```

Note:

6.2.2 The

`solve(`*rhs, transpose=False*`)`

Solve the linear system `A x = rhs`, where A is the input matrix and `rhs` is a Numpy vector of appropriate dimension. The result is placed in the `sol` member of the class instance.

If the optional argument

scale string that specifies the scaling UMFPACK should use. Valid values are ‘none’,

PysparsolvePyspar(*rhs, x*Pyspar*)*PysparSolve the linear systemPysparA x = rhsPyspar. The result is placed in the

CHAPTER

SEVEN

EIGENVALUE SOLVER

7.1 The `jdsym` Module

The `jdsym` module provides an implementation of the JDSYM algorithm, that is conveniently callable from Python. JDSYM is an eigenvalue solver to compute eigenpairs of a generalised matrix eigenvalue problem of the form

$$Ax = \quad Mx \tag{7.1}$$

or a standard eigenvalue problem of the form

$$Ax = \quad x \tag{7.2}$$

where A 0(an)-25symmetric and M 0(an)-25symmetric positive definite.

The module exports a single function:

`jdsym`(*A, M, K, kmax, tau, jdtol, itmax, linsolver, **kwargs*)
    Implements Jacobi-Davidson iterative method to identify a given number of eigenvalues near a target value.

Parameters
- **A** the matrix A in (7.1) or (7.2). A must provide the `shape` attribute and the `matvec` and `matvec_transp` methods.
- **M** the matrix M in (7.1). M must provide the `shape` attribute and the `matvec` and `matvec_transp` methods. If the standard eigenvalue problem (7.2

blkwise is an integer that affects the convergence criterion if `blksize`

```
A = A.to_sss(); M = M.to_sss()
k_conv, lmbd, Q, it  = jdsym.jdsym(A, M, K, 5, tau,
                                  1e-10, 150, itsolvers.qmrs,
                                  jmin=5, jmax=10, clvl=1, strategy=1)
```

CHAPTER

EIGHT

HIGHER-LEVEL SPARSE MATRIX CLASSES

8.1 The `pysparseMatrix` module

class `PysparseMatrix(`***kwargs*`)`

Bases: `sparseMatrix.SparseMatrix`

```
>>> L = PysparseMatrix(size = 3)
```

8.1.1 Creating an Identity Matrix

class `PysparseIdentityMatrix`(*size*)

Bases: `pysparseMatrix.PysparseMatrix`

Represents a sparse identity matrix for pysparse.

CHAPTER

NINE

CHAPTER

TWELVE

INDICES AND TABLES

BIBLIOGRAPHY

[DEGLL99] J. W. Demmel, S. C. Eisenstat, J. R. Gilbert, X. S. Li and J. W. H. Liu, *A supernodal approach to sparse partial pivoting*, SIAM Journal on Matrix Analysis and Applications **20**(3), pp. 720-755, 1999.

[DGL99] J. W. Demmel, J. R. Gilbert and X. S. Li, *An Asynchronous Parallel Supernodal Algorithm for Sparse Gaussian Elimination*, SIAM Journal on Matrix Analysis and Applications **20**(4), pp. 915-952, 1999.

[LD03] X. S. Li and J. W. Demmel, *SuperLU_DIST: A Scacable Distributed-Memory Sparse Direct Solver for Unsymmetric Linear Systems*, ACM Transactions on Mathematical Software **29**(2), pp. 110-140, 2003.

[SLU] http://crd.lbl.gov/~xiaoye/SuperLU

[D04a] T. A. Davis, *A column pre-ordering strategy for the unsymmetric-pattern multifrontal method*, ACM Transactions on Mathematical Software, **30**(2), pp. 165-195, 2004. T. A. Davis, *Algorithm 832: UMFPACK, an unsymmetric-pattern multifrontal method*,
Mathematical Software, **30**(2), pp. 196. 2004.

[DD99] T. A. Davis and I. S. Duff, *A combined unifrontal/multifrontal method for unsymmetric s*
Transactions on Mathematical Software, **25**(1), pp. 1-19, 1999. T. A. Davis and I. S. Duff, *An unsymm*
Journal on Matrix Analysis and Application

[UMF] http://www.cise.ufl.edu/research/sparse/

INDEX