Sparse Matrices

Linear algebra

The linear algebra module contains a lot of matrix related functions, including linear equation solving, eigenvalue solvers, matrix functions (for example matrix-exponentiation), a number of different decompositions (SVD, LU, cholesky), etc.

Detailed documetation is available at: http://docs.scipy.org/doc/scipy/reference/linalg.html (http://docs.scipy.org/doc/scipy/reference/linalg.html)

Here we will look at how to use some of these functions:

Linear equation systems

Linear equation systems on the matrix form

$$Ax = b$$

where A is a matrix and x, b are vectors can be solved like:

In [1]:
```
from scipy.linalg import solve, norm, eigvals, eig, inv, det
import numpy as np
```

In [2]:
```
A = np.array([[3,2,0],[1,-1,0],[0,5,1]])
b = np.array([2,4,-1])
```

In [3]:
```
x = solve(A, b)

x
```

Out[3]:
```
array([ 2., -2.,  9.])
```

Matrix operations

In [4]:
```
# the matrix inverse
inv(A)
```

Out[4]:
```
array([[ 0.2,  0.4,  0. ],
       [ 0.2, -0.6,  0. ],
       [-1. ,  3. ,  1. ]])
```

In [5]:
```
# determinant
det(A)
```

Out[5]:
```
-5.0
```

In [6]:
```
# norms of various orders
norm(A, ord=2), norm(A, ord=np.Inf)
```

Out[6]:
```
(5.652162851117261, 6.0)
```

Sparse matrices

Sparse matrices are often useful in numerical simulations dealing with large systems, if the problem can be described in matrix form where the matrices or vectors mostly contains zeros.

There are many possible strategies for storing sparse matrices in an efficient way. Some of the most common are the so-called coordinate form (COO), list of list (LIL) form, and compressed-sparse column CSC (and row, CSR). Each format has some advantanges and disadvantages. Most computational algorithms (equation solving, matrix-matrix multiplication, etc) can be efficiently implemented using CSR or CSC formats, but they are not so intuitive and not so easy to initialize. So often a sparse matrix is initially created in COO or LIL format (where we can efficiently add elements to the sparse matrix data), and then converted to CSC or CSR before used in real calcalations.

For more information about these sparse formats, see e.g. http://en.wikipedia.org/wiki/Sparse_matrix (http://en.wikipedia.org/wiki/Sparse_matrix)

When we create a sparse matrix we have to choose which format it should be stored in. For example,

In [7]:
```
from scipy.sparse import *
```

In [8]:
```
# dense matrix
M = np.array([[1,0,0,0], [0,3,0,0], [0,1,1,0], [1,0,0,1]]); M
```

Out[8]:
```
array([[1, 0, 0, 0],
       [0, 3, 0, 0],
       [0, 1, 1, 0],
       [1, 0, 0, 1]])
```

In [9]:
```
# convert from dense to sparse
A = csr_matrix(M); A
```

Out[9]:
```
<4x4 sparse matrix of type '<class 'numpy.intc'>'
        with 6 stored elements in Compressed Sparse Row format>
```

In [10]:
```
# convert from sparse to dense
A.todense()
```

Out[10]:
```
matrix([[1, 0, 0, 0],
        [0, 3, 0, 0],
        [0, 1, 1, 0],
        [1, 0, 0, 1]], dtype=int32)
```

More efficient way to create sparse matrices: create an empty matrix and populate with using matrix indexing (avoids creating a potentially large dense matrix)

In [11]:
```
A = lil_matrix((4,4)) # empty 4x4 sparse matrix
A[0,0] = 1
A[1,1] = 3
A[2,2] = A[2,1] = 1
A[3,3] = A[3,0] = 1
A
```

Out[11]:
```
<4x4 sparse matrix of type '<class 'numpy.float64'>'
        with 6 stored elements in List of Lists format>
```

In [12]:
```
A.todense()
```

Out[12]:
```
matrix([[1., 0., 0., 0.],
        [0., 3., 0., 0.],
        [0., 1., 1., 0.],
        [1., 0., 0., 1.]])
```

Converting between different sparse matrix formats:

In [13]:
```
A
```

Out[13]:
```
<4x4 sparse matrix of type '<class 'numpy.float64'>'
        with 6 stored elements in List of Lists format>
```

In [14]:
```
A = csr_matrix(A); A
```

Out[14]:
```
<4x4 sparse matrix of type '<class 'numpy.float64'>'
        with 6 stored elements in Compressed Sparse Row format>
```

In [15]:
```
A = csc_matrix(A); A
```

Out[15]:
```
<4x4 sparse matrix of type '<class 'numpy.float64'>'
        with 6 stored elements in Compressed Sparse Column format>
```

We can compute with sparse matrices like with dense matrices:

In [16]:
```
A.todense()
```

Out[16]:
```
matrix([[1., 0., 0., 0.],
        [0., 3., 0., 0.],
        [0., 1., 1., 0.],
        [1., 0., 0., 1.]])
```

In [17]:
```
(A * A).todense()
```

Out[17]:
```
matrix([[1., 0., 0., 0.],
        [0., 9., 0., 0.],
        [0., 4., 1., 0.],
        [2., 0., 0., 1.]])
```

In [18]:
```
v = np.array([1,2,3,4])[:,np.newaxis]; v
```

Out[18]:
```
array([[1],
       [2],
       [3],
       [4]])
```

In [19]:
```
# sparse matrix - dense vector multiplication
A * v
```

Out[19]:
```
array([[1.],
       [6.],
       [5.],
       [5.]])
```

In [20]:
```
# same result with dense matrix - dense vector multiplcation
A.todense() * v
```

Out[20]:
```
matrix([[1.],
        [6.],
        [5.],
        [5.]])
```

Original versions of these notebooks created by J.R. Johansson (robert@riken.jp (mailto:robert@riken.jp)) http://dml.riken.jp/~rob/ (http://dml.riken.jp/~rob/), modifications have been made by Dr. Derek Riley