Cheatography

Convex Optimization Cheat Sheet

by ThanhTrungK15 via cheatography.com/137592/cs/28908/

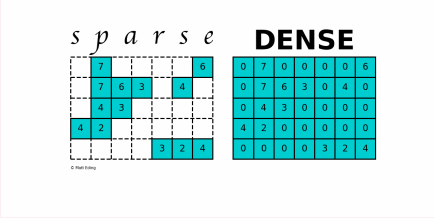
CVXOPT

The **CVXOPT** is a free software package for convex optimization based on the Python programming language. Its main purpose is to make the development of software for convex optimization applications straightforward by building on Python's extensive standard library and on the strengths of Python as a high-level programming language.

Use the following import convention:

```
>>> import cvxopt
```

Dense and Sparse Matrices



CVXOPT extends the built-in Python objects with two matrix objects: ***spmatrix*** for sparse matrix and ***matrix*** for dense matrix

Dense Matrices

```
--- The function matrix ---
>>> from cvxopt import matrix
>>> A = matrix(1, (1, 4))
>>> print(A)
   [ 1 1 1 1]
>>> A = matrix(1.0, (1, 4))
>>> print(A)
   [ 1.00e+00 1.00e+00 1.00e+00 1.00e+00]
>>> A = matrix(1 + 1j)
>>> print(A)
   [ 1.00e+00+j1.00e+00]
--- Several ways to define integer matrix ---
>>> A = matrix([0, 1, 2, 3], (2,2))
>>> A = matrix((0, 1, 2, 3), (2,2))
>>> A = matrix(range(4), (2,2))
>>> from array import array
>>> A = matrix(array('i', [0,1,2,3]), (2,2))
>>> print(A)
   [ 0 2]
   [ 1 3]
--- NumPy arrays can be converted to matrices ---
```

Dense Matrices (cont)

```
>>> from numpy import array
>>> x = array([[1., 2., 3.], [4., 5., 6.]])
>>> print(x)
   array([[ 1. 2. 3.]
          [ 4. 5. 6.]])
>>> print(matrix(x))
   [ 1.00e+00 2.00e+00 3.00e+00]
   [ 4.00e+00 5.00e+00 6.00e+00]
--- Another ways to create dense matrix ---
>>> print(matrix([[1., 2.], [3., 4.], [5., 6.]]))
   [ 1.00e+00 3.00e+00 5.00e+00]
   [ 2.00e+00 4.00e+00 6.00e+00]
>>> B1 = matrix([6, 7, 8, 9, 10, 11], (2,3))
>>> B2 = matrix([12, 13, 14, 15, 16, 17], (2,3))
>>> B3 = matrix([18, 19, 20], (1,3))
>>> D = matrix([B1, B2, B3])
>>> print(D)
   [ 6 8 10]
   [ 7 9 11]
   [ 12 14 16]
   [ 13 15 17]
   [ 18 19 20]
```

Sparse Matrices

```
>>> from cvxopt import matrix, spmatrix, sparse,
spdiag
--- The function spmatrix ---
>>> A = spmatrix(1.0, range(2), range(2))
>>> print(A)
   [ 1.00e+00 0 ]
   [ 0 1.00e+00]
>>> A = spmatrix([1, 2, 3, 4], [0, 0, 1, 1], [0,
1, 0, 1])
>>> print(A)
   [ 1.00e+00 2.00e+00]
   [ 3.00e+00 4.00e+00]
--- The function sparse ---
>>> A = matrix([[1, 2], [5, 6]])
>>> print(A)
```

Sparse Matrices (cont)

```
   [ 1 5]
   [ 2 6]
>>> B = spmatrix([], [], [], (2, 2))
>>> print(B)
   [0 0]
   [0 0]
>>> C = spmatrix([4, 2, 1, 9], [0, 0, 1, 1], [0, 1,
1, 0])
>>> print(C)
    [ 4.00e+00 2.00e+00]
    [ 9.00e+00 1.00e+00]
>>> D = sparse([[A, B], [B, C]])
>>> print(D)
   [ 1.00e+00 5.00e+00 0 0 ]
   [ 2.00e+00 6.00e+00 0 0 ]
   [ 0 0 4.00e+00 2.00e+00]
   [ 0 0 9.00e+00 1.00e+00]
>>> D = sparse([A, C])
>>> print(D)
   [ 1.00e+00 5.00e+00]
   [ 2.00e+00 6.00e+00]
   [ 4.00e+00 2.00e+00]
   [ 9.00e+00 1.00e+00]
--- The function spdiag ---
>>> A = 3.0
>>> print(A)
   3.0
>>> B = matrix([[1, 2], [4, 3]])
>>> print(B)
   [ 1 4]
   [ 2 3]
>>> C = spmatrix([4, 5, 6, 7], [0, 0, 1, 1], [0, 1,
0, 1])
>>> print(C)
   [ 4.00e+00 5.00e+00]
   [ 6.00e+00 7.00e+00]
>>> D = spdiag([A, B, C])
>>> print(D)
```

Sparse Matrices (cont)

```
   [ 3.00e+00 0 0 0 0 ]
   [ 0 1.00e+00 4.00e+00 0 0 ]
   [ 0 2.00e+00 3.00e+00 0 0 ]
   [ 0 0 0 4.00e+00 5.00e+00]
   [ 0 0 0 6.00e+00 7.00e+00]
```

Arithmetic Operations

Unary plus/minus	`+A, -A`
Addition	`A + B, A + c, c + A`
Subtraction	`A - B, A - c, c - A`
Matrix multiplication	`A * B`
Scalar multiplication and division	`c A, A c, A / c`
Remainder after division	`D % c`
Elementwise exponentiation	`D**e`
In-place addition	`A += B, A += c`
In-place subtraction	`A -= B, A -= c`
In-place scalar multiplication and division	`A *= c, A /= c`
In-place remainder	`A %= c`

`A` and `B` are dense or sparse matrices of compatible dimensions.
`c` is a scalar (a Python number or a dense 1 by 1 matrix)
`D` is a dense matrix.
`e` is a Python number