MatPy — Matrix package for Python

Huaiyu Zhu, <hzhu@users.sourceforge.net>
http://MatPy.sourceforge.net/

Version 0.3.1, March 23, 2002

Abstract

This document describes **MatPy**, a **Python** package for numerical computation with **Matlab**-like interface, implemented as wrappers around the packages Numerical, Gnuplot and SpecialFuncs.

It differs from the **NumPy** package in the user interface, which is more oriented toward linear algebra and for interactive use.

It differs from **Matlab** in the availability and utilization of **Python**'s excellent object-oriented features.

Contents

1 Introduction

This is an attempt to combine the elegance **`Python`** as an object-oriented programming language and the convenience of **`Matlab`** expressions for numerical computation.

1.1 Some highlights (as compared to `Numeric` package)

- Row vector $= 1 \times n$ matrix, col vector $= n \times 1$ matrix
- Slicing like **`A[:,0], A[0,:]`** gives col and row vectors
- Assembles block matrices from submatrices
- **`A+B, A-B, A*B, A/B, A**n`** are as in linear algebra
- Matrix functions **`expm, sqrtm`**, ...
- Complex matrices behave correctly
- Allow both matrix and elementwise operations

1.2 Previously added features

- Easy interface to **`gnuplot`**. Also include change axis, holdon, holdoff, plotlines, text, etc.
- Accepts NaN and Inf

- An interactive console **`matpy`** with MatLab-like startups, and commands **`help, functions, demo, lookfor`**, etc. **`interact.py`** as a simpler and more robust substitute of matpy console.
- Special math functions (like **`gamma, erf`** from **`cephes`** library.
- Classes of statistical distributions, with members **`.pdf, .cdf, .cdfc, .rand, .cdfi, .cdfci`**.
- User changeable format for matrix elements.
- Split matrix into list of rows and columns.
- Elementwise comarison. any. all.
- (Scaled) histogram
- Augmented assignment.
- ordinary names for math operators.
- Member functions .T .H .I .C without ().
- DynSys/kalman.py for Kalman filters (tests/test_kalman).
- Patch for op for elementwise operators by Greg Lielens.

1.3 Latest features (version 0.4.0)

- List of names and docs generated by interact.py.
- plainsolve and LMS solve.
- Many efficiency improvements.
- The object nothing=Nothing(), useful for suppressing graphics.
- Check for attempts to plot complex numbers
- Adding tilde operator.

2 Installation

2.1 Prerequisite

All the prerequisite packages and utilities are listed on the MatPy home page: http://MatPy.sourceforge.net.

- Obviously you need **`Python`**. Version 1.5.2, 1.6 and 2.0 have been tested.
- You need the **`Numeric`** package. Version 15 and 16 have been tested.
- You need the **`Gnuplot`** package if you want plotting.
- You need the **`Cephes`** library if you want special functions. This is also used for statistical stuff.
- If you want to generate the docs yourself, you need **`LaTeX`** and **`Latex2HTML`**. But you can also get the generated docs in the MatPy-docs tarball.

2.2 Installation with `make`

The easiest installation method is using `make` (if you can):

- Edit the `Makefile` to prescribe where to put things (Change `MODDIR`, `DOCDIR` and `BINDIR`).
- Do a `make install`. This will install python module files, run tests, make docs and install docs.

That's all you have to do if you've got all tools (See below). I installed it on Linux, Solaris, and Windows NT with CygWin.

If you want to do it step by step, but still using `make` (See also the file `INSTALL`):

- `make installmod` will install python module files.
- `make examples` will run the test examples and put results in the docs dir.
- `make documentation` will generate the docs (after making examples). Note you can download `MatPy-docs` instead.
- `make installdoc` will install the documentation files.

2.3 Installation by hand copying

If `make` does not work for you, you can do it by hand, using the `Makefile` as a guide. Here's an outline:

- Install the modules: Choose a directory `DIR` in your `$PYTHONPATH`. Copy the whole directory `MatPy-$VERSION` to be `DIR/MatPy`. You can leave out all the files not ending in `.py` or you can keep them.
- Run the tests: Run programs in the `tests` directory. You can also do this as described in 3
- Generate the documentations: (Note you can download `MatPy-docs` instead.) For each `test_TOPIC` generate `result_TOPIC` in `docs` directory, then generate the docs in `docs`.
- Install documentation: Copy everything in `docs` into a desired place.

Please let me know about installation on different systems. In the future it is intended to use the `Distutils` package.

3 Getting started and examples

3.1 Getting started

You can start by either trying the interactive mode or running and examining the examples.

To start with the interactive mode, within any python interactive environment, type `from MatPy.interact import *`. Then follow the instructions to type `help, names, lookfor, demo`, and so on. The `demo` command would run all the examples as described below. [1]

The examples are in the `tests` directory. These examples serve to

- Test the correctness of the implementation `MatPy`.

[1]The file `interact.py` is a simpler and more robust alternative to the file `matpy`. It is to be run within python command line interpreter instead of carrying its own light-weight version.

- Demo the relevant behavior of **`MatPy`**.
- As examples of usage for **`MatPy`**.

There are several ways to use the examples

- During installation. **`make installmod; make examples`** would run all the examples. Alternatively, **`make install`** will call **`make`** which will call **`make examples`**.
- After install. run **`matpy`** (directly on Unix or run **`python matpy`** on Windows). Within it you can explore **`help`**, **`functions`**, **`demo`**, **`lookfor(name)`**, etc.
- Run individually. For example, **`python tests/test_basic.py`** will test some basic functionalities. To run them all, do **`python tests/test_all.py`**.
- Within normal python interpreter. Type **`from MatPy.tests import *`** will run all the tests.
- Just read the example code to see how it works.

The programs and results are listed below.

3.2 Tests of basic operations

The file **`test_basic.py`** tests most of the basic operations. Results are at 3.2

```
#!/usr/bin/env python
# (C) 2000 Huaiyu Zhu <hzhu@users.sourceforge.net>.  Licence: GPL
# $Id: test_basic.py,v 1.6 2000/09/13 01:52:36 hzhu Exp $
"""
Test basic operations for Matrix module
"""

from MatPy.Matrix import Matrix, r_range, zeros, ones, eye, rand, norm, mnorm
from MatPy.Matrix import max, min, cumsum, sum, mfunc, solve
from MatPy.mfuncs import expm, logm, sqrtm

import  time
def tic(): global _time; _time = time.time()
def toc(): print "time lapse =",  time.time() - _time

print "-"*40, "basic vector"
a = r_range(-1, 3)
print a, a.__class__
a[2] = 9
assert a.shape == (1,4)
assert a[0,2] == a[2] == 9

print "-"*40, "basic matrix"
b = zeros((2,2))
b [1,0] = 1; b[0,1] = 2
print b
print b.shape,  b[0,1], b[1,0]
print 1 + b + 1
print 2 - b - 2
print 3 * b * 3
print "-"*40, "inverse matrix"
print b.I
print 4 / b
print 4 / b * b

assert norm(b.I * b - eye(2)) < 1e-15
assert norm(b.I - 1/b) < 1e-15

print "-"*40, "arithmatic, norm"
print 1.1 * Matrix(a)*2 + 2 + a

print "-"*40, "special matrices"
c = ones(a.shape) * 2 + zeros(a.shape) + rand(a.shape)
print c

print "-"*40, "transpose, multiplication"
d = -a.T
assert d.shape == (4,1)
assert norm(d.T + a) < 1e-15

print d*c, c*d

print "-"*40, "linear transform"
X = rand((9,4))
y = X * d
print y.T

print "-"*40, "eigenvalues"
C = rand((5,5))
print C
print C.eig

print "-"*40, "matrix functions:"
print "-"*40, "identity function"
print C
D = mfunc(lambda x:x, C)
print D
assert  norm(D - C) < 1e-14

print "-"*40, "sqrtm, powm"
A = X.T * X
B = sqrtm(A)
print B
assert norm(B**2 - A) < 4e-14
assert norm(B*B - A) < 2e-14

print "-"*40, "expm, logm"
print expm(C)
assert logm(expm(C)), norm(logm(expm(C)) - C) < 1e-15

print "-"*40, "inverse, eye"
print A.I
assert norm(A.I * A - eye(4)) < 1e-14
print norm(A.I * A - eye(4))

print "-"*40, "solve by inverse"
tic()
```

```
A = X.T * X
b = X.T * y
d1 = (A.I * b)
toc()
e = d1 - d
assert norm(e) < 1e-13
assert mnorm(e) < 1e-13


print "-"*40, "linear equation"
tic()
d2 = solve(X , y)
toc()
assert norm(d2 - d) < 8e-14
assert norm(d2 - d1) < 8e-14
assert norm(X*d2 - y) < 2e-14
```

Result obtained with

`>>> from MatPy.tests import test_basic`

```
---------------------------------------- basic vector
[-1   0   1   2  ] MatPy.Matrix.Matrix
---------------------------------------- basic matrix
[ 0      2
  1      0     ]
(2, 2) 2.0 1.0
[ 2      4
  3      2     ]
[ 0     -2
 -1      0     ]
[ 0      18
  9      0     ]
---------------------------------------- inverse matrix
[ 0      1
  0.5    0     ]
[ 0      4
  2      0     ]
[ 4      0
  0      4     ]
---------------------------------------- arithmatic, norm
[-1.2    2      30.8   8.4   ]
---------------------------------------- special matrices
[ 2.83   2.44   2.72   2.73  ]
---------------------------------------- transpose, multiplication
[ 2.83   2.44   2.72   2.73
  0      0      0      0
 -25.5  -22    -24.5  -24.5
 -5.66  -4.88  -5.44  -5.45  ] [-27.1  ]
---------------------------------------- linear transform
[-4.71  -4.01  -5.28  -5.73  -2     -10.1  -2.5   -1.81  -7.76  ]
---------------------------------------- eigenvalues
[ 0.664  0.0771  0.0881  0.755  0.658
  0.301  0.0204  0.408  0.908  0.138
  0.0539  0.641  0.911  0.895  0.908
  0.7    0.223  0.7    0.384  0.597
  0.907  0.807  0.859  0.43   0.869 ]
[ 2.89+0j -0.591+0j  0.715+0j -0.0847+0.38j -0.0847-0.38j ]
```

```
---------------------------------------- matrix functions:
---------------------------------------- identity function
[ 0.664  0.0771  0.0881  0.755  0.658
  0.301  0.0204  0.408  0.908  0.138
  0.0539  0.641  0.911  0.895  0.908
  0.7    0.223  0.7    0.384  0.597
  0.907  0.807  0.859  0.43   0.869 ]
[ 0.664  0.0771  0.0881  0.755  0.658
  0.301  0.0204  0.408  0.908  0.138
  0.0539  0.641  0.911  0.895  0.908
  0.7    0.223  0.7    0.384  0.597
  0.907  0.807  0.859  0.43   0.869 ]
---------------------------------------- sqrtm, powm
[ 1.91   0.623  0.616  0.641
  0.623  1.02   0.297  0.263
  0.616  0.297  1.37   0.677
  0.641  0.263  0.677  1.51  ]
---------------------------------------- expm, logm
[ 3.93   1.51   2.33   3.08   3.22
  1.93   2.15   2.29   2.76   2.17
  3.17   3.08   5.83   4.73   4.89
  3.13   2.01   3.47   4.3    3.67
  4.36   3.28   4.79   4.67   6.14  ]
---------------------------------------- inverse, eye
[ 0.774 -0.718 -0.202 -0.233
 -0.718  1.69  -0.144  0.0645
 -0.202 -0.144  1.17  -0.653
 -0.233  0.0645 -0.653  0.977 ]
1.01899608191e-15
---------------------------------------- solve by inverse
time lapse = 0.0102770328522
---------------------------------------- linear equation
time lapse = 0.00766706466675
```

3.3 Tests of linear algebra operations

The file **`test_linear.py`** tests linear algebra operations, including inverse and solving equations. Results are at 3.3

```
#!/usr/bin/env python
# (C) 2000 Huaiyu Zhu <hzhu@users.sourceforge.net>.  Licence: GPL
# $Id: test_linear.py,v 1.4 2000/07/26 20:58:27 hzhu Exp $
"""
Test linear algebra for Matrix module
"""

from MatPy.Matrix import *
from MatPy.mfuncs import sqrtm, expm, logm

print "-"*40, "matrix functions"
A = rand((3,3))
print A.typecode, norm(A + A - 2* A)
print norm(A*A*A / A /(A*A) - eye(3))
print norm(A**2 - A*A)
B = sqrtm(A)
print B
print B.typecode, norm(A - B*B), mnorm(A - B*B)
```

```
x = rand((3,1))
y = A*x
print solve(A,y)
print y.T/A.T

C = logm(A)
D = expm(C)
print C.typecode, D.typecode, norm(D-A)
```

Result obtained with
`>>> from MatPy.tests import test_linear`

```
---------------------------------------- matrix functions
d 0.0
2.17261484018e-13
1.42177919159e-15
[ 0.829  0.26   0.446
  0.587  0.631  0.101
 -0.0228  0.648  0.772 ]
d 7.79136136032e-16 1.13356978563e-15
[ 0.497
  0.223
  0.216 ]
[ 0.497  0.223  0.216 ]
d d 1.53536257996e-15
```

3.4 Tests of slicing into submatrices

The file **`test_slice.py`** tests slicing a matrix into submatrices and assigning values to submatrices. Results are at 3.4

```
#!/usr/bin/env python
# (C) 2000 Huaiyu Zhu <hzhu@users.sourceforge.net>.  Licence: GPL
# $Id: test_slice.py,v 1.2 2000/07/26 20:58:27 hzhu Exp $
"""
Test slicing for Matrix module
Note that getslice and setslice is only used for 1d arrays.
For higher dim, the getitem and setitem are used with slice objects.
"""

class A:
def __getitem__(self, *i): print "getitem", i
def __setitem__(self, *i): print "setitem", i
def __getslice__(self, *i): print "getslice", i
def __setslice__(self, *i): print "setslice", i

a = A()
a[1]
a[1] = 2
a[1:2]
a[1:2] = 3
a[1,2]
a[1,2] = 3
a[1,2:3]
a[1:2, 3]
```

```
a[1:2, 3] = 4
a[1, 2:3] = 4
a[1:2, 3:4]
a[1:2, 3:4] = 5

#-------------------------------------------------------------------
from MatPy.Matrix import *

print "-"*40, "row vectors"
a = rand(6)
print a
print a[1]
a[1] = 2
print a
print a[1:4]
a[1:4] = [2,3,4]
print a

print "-"*40, "column vectors"
a = rand(6).T
print a
print a[1]
a[1] = 2
print a
print a[1:4]
a[1:4] = [2,3,4]
print a

print "-"*40, "matrices items"
A = rand((4,4))
print A, A.__class__
print A[1,2]
A[1,3] = 3
print A

print "-"*40, "matrices slices"
print A[1, 2:4]
A[1, 2:4] = [1,2]
print A

print "-"*40, "vectors as submatrices"
print A[0,:]
print A[:,0]

print "-"*40, "matrices slices"
print A[1:3, 2:4]
A[1:3, 2:4] = zeros((2,2))
print A
```

Result obtained with

`>>> from MatPy.tests import test_slice`

```
getitem (1,)
setitem (1, 2)
getslice (1, 2)
setslice (1, 2, 3)
getitem ((1, 2),)
```

```
setitem ((1, 2), 3)
getitem ((1, slice(2, 3, None)),)
getitem ((slice(1, 2, None), 3),)
setitem ((slice(1, 2, None), 3), 4)
setitem ((1, slice(2, 3, None)), 4)
getitem ((slice(1, 2, None), slice(3, 4, None)),)
setitem ((slice(1, 2, None), slice(3, 4, None)), 5)
---------------------------------------- row vectors
[ 0.829  0.897  0.761  0.983  0.312  0.575 ]
0.896994531155
[ 0.829  2      0.761  0.983  0.312  0.575 ]
[ 2      0.761  0.983 ]
[ 0.829  2      3      4      0.312  0.575 ]
---------------------------------------- column vectors
[ 0.0946
  0.658
  0.989
  0.875
  0.991
  0.32  ]
0.658199727535
[ 0.0946
  2
  0.989
  0.875
  0.991
  0.32  ]
[ 2
  0.989
  0.875 ]
[ 0.0946
  2
  3
  4
  0.991
  0.32  ]
---------------------------------------- matrices items
[ 0.699  0.171  0.584  0.973
  0.909  0.767  0.211  0.302
  0.31   0.537  0.978  0.599
  0.569  0.984  0.213  0.403 ] MatPy.Matrix.Matrix
0.210701078176
[ 0.699  0.171  0.584  0.973
  0.909  0.767  0.211  3
  0.31   0.537  0.978  0.599
  0.569  0.984  0.213  0.403 ]
---------------------------------------- matrices slices
[ 0.211  3     ]
[ 0.699  0.171  0.584  0.973
  0.909  0.767  1      2
  0.31   0.537  0.978  0.599
  0.569  0.984  0.213  0.403 ]
---------------------------------------- vectors as submatrices
[ 0.699  0.171  0.584  0.973 ]
[ 0.699
  0.909
  0.31
  0.569 ]
---------------------------------------- matrices slices
[ 1      2
  0.978  0.599 ]
[ 0.699  0.171  0.584  0.973
  0.909  0.767  0      0
  0.31   0.537  0      0
  0.569  0.984  0.213  0.403 ]
```

3.5 Tests of assembling from block matrices

The file **`test_block.py`** tests making a large matrix from blocks of submatrices. Results are at 3.5

```
#!/usr/bin/env python
# (C) 2000 Huaiyu Zhu <hzhu@users.sourceforge.net>.  Licence: GPL
# $Id: test_block.py,v 1.6 2000/09/13 01:22:57 hzhu Exp $
"""
Test constructing block matrices for Matrix module
"""
from MatPy.Matrix import *
#------------------------------------------------------------------
print "-"*40, "Assemble block matrices"

print "For proper matrices"
a = ones((2,3))
b = [a*1, a*2, a*3]
print Matrix_c(b)
print Matrix_r(b)

print "For row vectors"
a = ones(3)
b = [a*1, a*2, a*3]
print Matrix_c(b)
print Matrix_r(b)

print "For col vectors"
a = ones(2).T
b = [a*1, a*2, a*3]
print Matrix_c(b)
print Matrix_r(b)

#------------------------------------------------------------------

print "For different shapes"
a = zeros((0,0))
b = ones(3)
c = ones((2,3))*2
print a, b, c
print Matrix_c((a, b, c))
print Matrix_r((a.T, b.T, c.T))

#------------------------------------------------------------------
print "-"*40, "Split into block matrices"

print "For proper matrices"
a = rand((2,3))
```

```
print len(rows(a)), rows(a)
print len(cols(a)), cols(a)

print "For row vectors"
a = rand(3)
print len(rows(a)), rows(a)
print len(cols(a)), cols(a)

print "For col vectors"
a = rand(2).T
print len(rows(a)), rows(a)
print len(cols(a)), cols(a)


#------------------------------------------------------------------
print "-"*40, "Split and assemble"
a = rand((2,3))
print a

b = rows(a)
print b
c = Matrix_c(b)
print c
assert norm(a - c) == 0

d = cols(a)
print d
e = Matrix_r(d)
print e
assert norm(a - e) == 0
```

Result obtained with

`>>> from MatPy.tests import test_block`

```
---------------------------------------- Assemble block matrices
For proper matrices
[ 1   1   1
  1   1   1
  2   2   2
  2   2   2
  3   3   3
  3   3   3  ]
[ 1   1   1   2   2   2   3   3   3
  1   1   1   2   2   2   3   3   3  ]
For row vectors
[ 1   1   1
  2   2   2
  3   3   3  ]
[ 1   1   1   2   2   2   3   3   3  ]
For col vectors
[ 1
  1
  2
  2
  3
  3  ]
```

```
[ 1   2   3
  1   2   3  ]
For different shapes
[ ] [ 1   1   1  ] [ 2   2   2
  2   2   2  ]
[ 1   1   1
  2   2   2
  2   2   2  ]
[ 1   2   2
  1   2   2
  1   2   2  ]
----------------------------------------- Split into block matrices
For proper matrices
2 [[[  0.829,  0.355,  0.802 ]], [[  0.24 ,  0.704,  0.917 ]]]
3 [[[  0.829 ],
 [  0.24  ]], [[  0.355 ],
 [  0.704 ]], [[  0.802 ],
 [  0.917 ]]]
For row vectors
1 [[[  0.596,  0.169,  0.664 ]]]
3 [[[  0.596 ]], [[  0.169 ]], [[  0.664 ]]]
For col vectors
2 [[[  0.63  ]], [[  0.527 ]]]
1 [[[  0.63  ],
 [  0.527 ]]]
----------------------------------------- Split and assemble
[ 0.528  0.829  0.289
  0.611  0.525  0.862 ]
[[[  0.528,  0.829,  0.289 ]], [[  0.611,  0.525,  0.862 ]]]
[ 0.528  0.829  0.289
  0.611  0.525  0.862 ]
[[[  0.528 ],
 [  0.611 ]], [[  0.829 ],
 [  0.525 ]], [[  0.289 ],
 [  0.862 ]]]
[ 0.528  0.829  0.289
  0.611  0.525  0.862 ]
```

3.6 Tests of elementwise operations with extensions

The file **`test_cross.py`** tests issues concerning elementwise operations and operations with different types: scalar, vector, matrix. Results are at 3.6

```
#!/usr/bin/env python
# (C) 2000 Huaiyu Zhu <hzhu@users.sourceforge.net>.  Licence: GPL
# $Id: test_cross.py,v 1.3 2000/10/05 01:33:21 hzhu Exp $
"""
Test cross operations for Matrix module
"""

from MatPy.Matrix import ones, norm

a = ones((1,3))*2.
b = ones((3,1))*3.

def checkequal(x, y):
print x
print y
assert norm(x-y) == 0

print "-"*40
checkequal(a.e_add(b),  b.e_add(a))
checkequal(a.e_sub(b), -(b.e_sub(a)))

print "-"*40
checkequal (a.e_mul(b), b.e_mul(a))
checkequal (a.e_div(b), (b.e_div(a)).re_div(1))

print "-"*40
checkequal(a.e_add(1),  a.re_add(1))
checkequal(a.e_sub(1), -(a.re_sub(1)))

print "-"*40
checkequal (a.e_mul(3), a.re_mul(3))
checkequal (a.e_div(3), (a.re_div(3)).re_div(1))

print "-"*40

try:
print a+b
print b+a
print a+b
print b+a
except ValueError:
print "These do not work, as expected"

print "-"*40

print a*b
print b*a

try:
print a/b
print b/a
except:
print "These should work with generalized inverse.  Need new code."
```

Result obtained with

>>> from MatPy.tests import test_cross

```
----------------------------------------
[ 5      5      5
  5      5      5
  5      5      5     ]
[ 5      5      5
  5      5      5
  5      5      5     ]
[-1     -1     -1
 -1     -1     -1
 -1     -1     -1     ]
[-1     -1     -1
 -1     -1     -1
 -1     -1     -1     ]
----------------------------------------
```

```
[ 6      6      6
  6      6      6
  6      6      6     ]
[ 6      6      6
  6      6      6
  6      6      6     ]
[ 0.667  0.667  0.667
  0.667  0.667  0.667
  0.667  0.667  0.667 ]
[ 0.667  0.667  0.667
  0.667  0.667  0.667
  0.667  0.667  0.667 ]
----------------------------------------
[ 3      3      3     ]
[ 3      3      3     ]
[ 1      1      1     ]
[ 1      1      1     ]
----------------------------------------
[ 6      6      6     ]
[ 6      6      6     ]
[ 0.667  0.667  0.667 ]
[ 0.667  0.667  0.667 ]
----------------------------------------
These do not work, as expected
----------------------------------------
[ 18    ]
[ 6      6      6
  6      6      6
  6      6      6     ]
These should work with generalized inverse.  Need new code.
```

3.7 Tests of the shape attribute

The file **`test_shape.py`** reveals shapes of various objects. Results are at 3.7

```
#!/usr/bin/env python
# (C) 2000 Huaiyu Zhu <hzhu@users.sourceforge.net>.  Licence: GPL
# $Id: test_shape.py,v 1.1.1.1 2000/06/03 23:56:50 hzhu Exp $
"""
Test various shapes
"""

from MatPy.Matrix import *

a = Matrix(1)
print a, a.shape

b = Matrix([1])
print b, b.shape

c = Matrix([[1]])
print c, c.shape

try:
d = Matrix([[[1]]])
print d, d.shape
except ValueError:
```

```
print "This does not work, as expected"

print "-"*40

a = Matrix(range(3))
print a, a.shape

b = Matrix([range(3)])
print b, b.shape

try:
c = Matrix([[range(3)]])
print c, c.shape
except ValueError:
print "This does not work, as expected"

print b[0]
print b[1]
print b[2]
print b[0,0]
print b[0,1]
print b[0,2]

print "-"*40

a = Matrix([])
print a, a.shape

b = Matrix([[]])
print b, b.shape

try:
c = Matrix([[[]]])
print c, c.shape
except ValueError:
print "This does not work, as expected"
```

Result obtained with

>>> from MatPy.tests import test_shape

```
[ 1  ] (1, 1)
[ 1  ] (1, 1)
[ 1  ] (1, 1)
This does not work, as expected
----------------------------------------
[ 0   1   2  ] (1, 3)
[ 0   1   2  ] (1, 3)
This does not work, as expected
0
1
2
0
1
2
----------------------------------------
```

```
[ ] (1, 0)
[ ] (1, 0)
This does not work, as expected
```

3.8 Tests of interaction with scalers

The file **`test_scalar.py`** tests several scalar representations, including the Scalar class. Results are at 3.8

```
#!/usr/bin/env python
# (C) 2000 Huaiyu Zhu <hzhu@users.sourceforge.net>.  Licence: GPL
# $Id: test_scalar.py,v 1.3 2000/09/13 01:52:36 hzhu Exp $
"""
Test basic operations for Scalar module
"""

from MatPy.Matrix import *

a = Matrix(2.)
A = rand((2,3))

print a
print A

print A.__tadd__(a)
print a.__tadd__(A)


print "-"*40
from MatPy.Scalar import *

nums = [1, 2., -3, -3.0, Scalar(1), Scalar(2.0), Scalar(1j), Scalar(1.j)]

for i in nums:
for j in nums: print "%7s" % ‘i*j‘,
print

print Scalar(3)*Scalar(0.99) < Scalar(1) + Scalar(2) \
  == Scalar(3.) < Scalar(3) + 1e-12
```

Result obtained with
`>>> from MatPy.tests import test_scalar`

```
[ 2      ]
[ 0.829  0.126  0.782
  0.112  0.00821  0.746 ]
[ 2.83   2.13   2.78
  2.11   2.01   2.75  ]
[ 2.83   2.13   2.78
  2.11   2.01   2.75  ]
----------------------------------------
      1     2.0      -3    -3.0       1     2.0      1j      1j
    2.0     4.0    -6.0    -6.0     2.0     4.0      2j      2j
     -3    -6.0       9     9.0      -3    -6.0     -3j     -3j
   -3.0    -6.0     9.0     9.0    -3.0    -6.0     -3j     -3j
      1     2.0      -3    -3.0       1     2.0      1j      1j
    2.0     4.0    -6.0    -6.0     2.0     4.0      2j      2j
     1j      2j     -3j     -3j      1j      2j (-1+0j) (-1+0j)
     1j      2j     -3j     -3j      1j      2j (-1+0j) (-1+0j)
1
```

3.9 Tests of various multiplications

The file **`test_multiply.py`** tests various multiplication modes. Results are at 3.9

```
#!/usr/bin/env python
# (C) 2000 Huaiyu Zhu <hzhu@users.sourceforge.net>.  Licence: GPL
# $Id: test_multiply.py,v 1.1.1.1 2000/06/03 23:56:50 hzhu Exp $
"""
Test various types of multiplication
"""

from MatPy.Matrix import *

a = ones((1,3))
b = ones((3,1))
print a
print b

print "-"*40

print a * 3.
print 3. * a

print b * 2.
print 2. * b

print "-"*40

print a * b
print b * a


print "-"*40

a = ones((2,3))
b = ones((3,2))
print a
print b

print "-"*40

print a * 3.
print 3. * a

print b * 2.
print 2. * b

print "-"*40

print a * b
print b * a
print "-"*40
```

Result obtained with
`>>> from MatPy.tests import test_multiply`

```
[ 1   1   1  ]
[ 1
  1
  1  ]
----------------------------------------
[ 3      3      3     ]
[ 3      3      3     ]
[ 2
  2
  2     ]
[ 2
  2
  2     ]
----------------------------------------
[ 3  ]
[ 1   1   1
  1   1   1
  1   1   1  ]
----------------------------------------
[ 1   1   1
  1   1   1  ]
[ 1   1
  1   1
  1   1  ]
----------------------------------------
[ 3      3      3
  3      3      3     ]
[ 3      3      3
  3      3      3     ]
[ 2      2
  2      2
  2      2     ]
[ 2      2
  2      2
  2      2     ]
----------------------------------------
[ 3   3
  3   3  ]
[ 2   2   2
  2   2   2
  2   2   2  ]
----------------------------------------
```

3.10 Tests of gplot module

The file **`test_gplot.py`** tests the interface to gnuplot. Results are at 3.10

```
#!/usr/bin/env python
# (C) 2000 Huaiyu Zhu <hzhu@users.sourceforge.net>.  Licence: GPL
# $Id: test_gplot.py,v 1.6 2000/10/05 01:28:27 hzhu Exp $
```

```
"""
Test gplot module
"""

from MatPy.Matrix import r_range, rand, cumsum
from MatPy.efuncs import sin, cos, exp, sqrt
from MatPy.gplot import Gplot

if __name__ == "__main__": waittime = None
else: waittime = 3
g = Gplot()
#from MatPy import nothing
#g = nothing

#------------------------------------------------------------------
title = "Gplot.plot : Plot four curves"
print title

x1 = r_range(100)/4.0
y1 = sin(x1)/2 - 1
x = r_range(300)/5.0
y2 = cos(sqrt(x)*10)/3
y3 = x.__tpow__(4); y3 = 2*y3/max(y3)
y4 = 1-exp(-x/3);
print y1.shape, y2.shape, y3.shape, y4.shape

g.title(title)
g.xlabel('x')
g.ylabel('y')
g.text((31,-0.8), "first label")
g.text((21,1.2), "second label")
g.plot((y1, y2, y3, y4),
   names=("sin x - 1", "cos sqrt x", "xˆ4", "1-eˆ-x"),
   xs=[x1] + [x]*3)

g.hardcopy("plot.ps")
g.wait(waittime)
g = Gplot()  # There seems no way to remove texts
#g = nothing

#------------------------------------------------------------------

title = """Gplot.mesh : mesh z against x and y """
print title
g.title(title)
g.xlabel('x')
g.ylabel('y')

x = r_range(40)/2.0
y = r_range(30)/10.0 - 1.5

for i in range(10):
 a = (sin(x+i) + 0.1*x).T
 b = y.__tpow__(2)
 m = a.__tsub__(b)
 #print m.shape, a.shape, b.shape
 g.mesh(m, x, y).wait(0.4)  # On my linux box this is neces-
```

```
sary for disk sync?
g.hardcopy("mesh.ps")
g.wait(waittime)

#------------------------------------------------------------------
title = "Gplot.plot : Using matrix columns as y"
print title
ys = rand((30,4))

g.title(title)
g.plot(ys).wait(waittime)

#------------------------------------------------------------------
title = "Gplot.plot : Using matrix as part of x, the rest de-
fault to index"
print title
xs = rand((30,2))
ys = cumsum(ys)
xs = cumsum(xs)

g.title(title)
g.plot(ys, xs).wait(waittime)


#------------------------------------------------------------------
title = "Gplot.holdon Gplot.holdoff : Plotting 10 lines in sequence"
print title

from MatPy.Matrix import cumsum, Matrix_c, zeros
from MatPy.Stats.distribs import randn
g.title(title)

g.holdon
for i in range(5):
 ys = Matrix_c([zeros((1,2)), randn((30,2))])
 g.plot(cumsum(ys)).wait(0.2)
g.wait(waittime)
g.holdoff

#------------------------------------------------------------------
title =  "Gplot.plotlines : Plot 8 lines, each composed 2d points"
print title
from MatPy.Matrix import max, max2
from MatPy.efuncs import abs

g.title(title)
lines = []
M = 0
for i in range(8):
line = cumsum(randn((90,2)))
lines.append(line)
g.text(line[-1]*1.3, ’%s’%i)
M = max(max2(abs(line)), M)

M = M*1.3
g.axis((-M,M), (0,0), equal=1)
g.plotlines(lines).wait(waittime)
```

Result obtained with

```
>>> from MatPy.tests import test_gplot
```

```
Gplot.plot : Plot four curves
(1, 100) (1, 300) (1, 300) (1, 300)
Gplot.mesh : mesh z against x and y
Gplot.plot : Using matrix columns as y
Gplot.plot : Using matrix as part of x, the rest default to index
Gplot.holdon Gplot.holdoff : Plotting 10 lines in sequence
Gplot.plotlines : Plot 8 lines, each composed 2d points
```

3.11 Tests of elementwise functions

The file **`test_efuncs.py`** tests the elementwise functions. Results are at 3.11

```
#!/usr/bin/env python
# (C) 2000 Huaiyu Zhu <hzhu@users.sourceforge.net>.  Licence: GPL
# $Id: test_efuncs.py,v 1.9 2001/08/29 09:17:28 hzhu Exp $
"""
Test efuncs module
"""

from MatPy.Matrix import r_range, rand, norm
from MatPy.efuncs import *
import sys

if __name__ == "__main__": waittime = None
else: waittime = 1

# Check consistency
def checkequal(x, y, r):
try:
assert norm(x-y) <= (norm(x)+norm(y)) *   r
except:
print sys.exc_info()
print "x =", x
print "y =", y
raise

A = rand((3,3))
checkequal(sqrt(A).__tpow__(2),  A, 1e-14)

checkequal(exp(A)+exp(-A),  2 * cosh(A), 1e-14)
checkequal(exp(A)-exp(-A),  2 * sinh(A), 1e-14)
checkequal(sinh(A), cosh(A).__tmul__(tanh(A)), 1e-14)
checkequal(cosh(A).__tpow__(2), 1 + sinh(A).__tpow__(2), 2e-14)

checkequal(1j*sin(A),  sinh(A*1j), 1e-14)
checkequal(cos(A),  cosh(A*1j), 1e-14)
checkequal(sin(A),  cos(A).__tmul__(tan(A)), 1e-14)
checkequal(cos(A).__tpow__(2), 1 -  sin(A).__tpow__(2), 2e-14)

checkequal(cos(acos(A)), A, 1e-14)
checkequal(sin(asin(A)), A, 1e-14)
checkequal(tan(atan(A)), A, 1e-14)


# Plot graphics
i = 0
def test(g, __name__, funcs, x):
"For each f in funcs, draw a line with f(args)"
global i
ys = map(lambda f,x=x:f(x), funcs)
names = map(lambda f:f.__name__, funcs)

if not len(funcs) == len(names): raise ValueError

g.plot(ys, names=names, xs=[x]*len(funcs))
g.hardcopy("efunc_%s.ps" %i)
i = i + 1
wait(waittime)

#------------------------------------------------------------------
print "All of these tests are shown in graphics"

from MatPy.gplot import Gplot, wait
g = Gplot()
x = r_range(100)/10.

#--------------------------------------------
g.title("Positive range functions")
test(g, __name__,
 (log, log10),
 (x+.1)/3)

#--------------------------------------------
g.title("Miscellenious functions")
test(g, __name__,
 (abs, ceil, floor),
 x-5)
"sign does not work?"

#--------------------------------------------
g.title("Trigonometric functions")
g.axis(yrange=(-3,3))
test(g, __name__,
 (sin, cos, tan, atan),
 (x-5)*2)
g.axis()

#--------------------------------------------
g.title("Exponential and related functions")
test(g, __name__,
 (exp, sinh, cosh, tanh, asinh),
 (x-5)/3)

#--------------------------------------------
g.title("Gamma and related special functions")
test(g, __name__,
 (lgam, psi, rgam),
 x+.1)
```

```
"""
abs,
ceil,
floor,
sign,
sqrt,
sinc,
exp,
log,
log10,
sinh,
cosh,
tanh,
asinh,
sin,
cos,
tan,
asin,
acos,
atan,
gam,
lgam,
igam,
igamc(x, a):
igami(y, a):
beta(a, b):
lbeta(a, b):
ibeta(x, a, b):
ibetai(y, a, b):
psi,
rgam,
"""
```

Result obtained with
`>>> from MatPy.tests import test_efuncs`

```
All of these tests are shown in graphics
```

3.12 Tests of matrix functions

The file **`test_mfuncs.py`** tests the matrix functions. Results are at 3.12

```
#!/usr/bin/env python
# (C) 2000 Huaiyu Zhu <hzhu@users.sourceforge.net>.  Licence: GPL
# $Id: test_mfuncs.py,v 1.4 2001/08/29 09:17:28 hzhu Exp $
"""
Test mfuncs module
"""

from MatPy.Matrix import rand, norm, eye
from MatPy.mfuncs import *
import sys

# Print out values
A = rand((3,3))
```

```
print A

for f in [sqrtm, expm, logm, sinhm, coshm, tanhm, sinm,
  cosm, tanm, asinm, acosm, atanm]:
print f(A)

# Check consistency
def checkequal(x, y, r):
try:
assert norm(x-y) <= (norm(x)+norm(y)) *   r
except:
print sys.exc_info()
print "x =", x
print "y =", y
raise

checkequal(sqrtm(A) ** 2, A, 1e-14)

checkequal(expm(A)+expm(-A), 2 * coshm(A), 1e-14)
checkequal(expm(A)-expm(-A), 2 * sinhm(A), 1e-14)
checkequal(sinhm(A), coshm(A)*tanhm(A), 1e-14)
#print "-"*40
#print coshm(A)**2
#print sinhm(A)**2
#print norm(coshm(A)**2 - sinhm(A)**2 - eye(3))
#print "-"*40
checkequal(coshm(A)**2 - sinhm(A)**2, eye(3), 2e-14)

checkequal(1j*sinm(A), sinhm(A*1j), 1e-14)
checkequal(cosm(A), coshm(A*1j), 1e-14)
checkequal(sinm(A), cosm(A)*tanm(A), 4e-14)
checkequal(cosm(A)**2 + sinm(A)**2, eye(3), 2e-14)

checkequal(cosm(acosm(A)), A, 1e-14)
checkequal(sinm(asinm(A)), A, 1e-14)
checkequal(tanm(atanm(A)), A, 1e-14)
```

Result obtained with

`>>> from MatPy.tests import test_mfuncs`

```
[ 0.828  0.139  0.5
  0.6    0.367  0.73
  0.12   0.85   0.14  ]
[ 0.875-0.0572j  0.0442+0.149j  0.308-0.135j
  0.36-0.118j  0.538+0.307j  0.467-0.278j
  0.0271+0.18j  0.576-0.469j  0.386+0.424j ]
[ 2.52   0.644  1.02
  1.31   2.04   1.34
  0.654  1.28   1.68  ]
[-0.518-0.267j  0.459+0.696j  0.387-0.63j
  0.548-0.549j -0.424+1.43j  0.413-1.3j
  0.492+0.836j  0.207-2.18j -0.494+1.98j ]
[ 1.03   0.293  0.671
  0.831  0.55   0.934
  0.282  1      0.277 ]
[ 1.48   0.351  0.351
  0.481  1.49   0.402
  0.372  0.271  1.4   ]
[ 0.617 -0.00855  0.327
  0.369  0.188  0.52
 -0.0342  0.682  0.0104 ]
[ 0.66   0.0146  0.36
  0.412  0.218  0.563
 -0.0104  0.721  0.0299 ]
[ 0.645 -0.247 -0.243
 -0.334  0.638 -0.277
 -0.262 -0.183  0.695 ]
[ 2.82   1.74   2.19
  2.9    2.24   2.72
  1.81   2.28   1.58  ]
[ 0.891-0.33j  0.175-0.271j  0.549-0.283j
  0.665-0.385j  0.413-0.316j  0.792-0.33j
  0.157-0.286j  0.904-0.234j  0.171-0.245j ]
[ 0.68+0.33j -0.175+0.271j -0.549+0.283j
 -0.665+0.385j  1.16+0.316j -0.792+0.33j
 -0.157+0.286j -0.904+0.234j  1.4+0.245j ]
[ 0.643  0.0117  0.349
  0.398  0.212  0.546
 -0.0128  0.701  0.0286 ]
```

3.13 Tests of tensor operations

The file **`test_tensor.py`** tests some tensor operations. Results are at 3.13

```
#!/usr/bin/env python
# (C) 2000 Huaiyu Zhu <hzhu@users.sourceforge.net>.  Licence: GPL
# $Id: test_tensor.py,v 1.2 2000/06/22 23:56:15 hzhu Exp $
"""
Test Tensor module
"""

from MatPy.Tensor import *
from MatPy import formats
formats.format = lambda x:"% -5.2g"%x

A = B = Tensor([[[1.,2],[3,4]],[[5,6],[7,8]]])
print A, A.shape
print A[0], A[0].shape
print A[0,1], A[0,1].shape
print A[0,1,1], A[0,1,1].shape

print "-"*40
A[0,0,0] = 9
print A, A.shape

print "-"*40
A = Tensor([[1,2],[3,4]])
print A, A.shape

print "-"*40
A = Tensor([[1,2,3,4]])
print A, A.shape

print "-"*40
```

```
A = Tensor([1,2,3,4])
print A, A.shape

print "-"*40
A = Tensor([1])
print A, A.shape

print "-"*40
A = Tensor(1)
print A, A.shape

print "-"*40
A = Tensor([])
print A, A.shape

print "-"*40
from MLab import rand
B[0,0,:] = rand(2)
print B, B.shape

print "========================================================="
print "Testing changing format"
a = MLab.arange(4)/3.

print formats.strF(a)
print formats.reprF(a),  "\n", "-"*40

a = MLab.array([a+1, a-1])
formats.formatF = lambda x:"% -7.4g"%x
print formats.strF(a)
print formats.reprF(a), "\n", "-"*40

a = MLab.array([a*4,a+10,a-1])
formats.formatF = lambda x:"% -5.2g"%x
print formats.strF(a)
print formats.reprF(a), "\n", "-"*40
```

Result obtained with

`>>> from MatPy.tests import test_tensor`

```
[[ 1      2
   3      4     ]
 [ 5      6
   7      8     ] ] (2, 2, 2)
[ 1      2
  3      4     ] (2, 2)
 3      4     (2,)
 4     (1,)
----------------------------------------
[[ 9      2
   3      4     ]
 [ 5      6
   7      8     ] ] (2, 2, 2)
----------------------------------------
[ 1   2
  3   4  ] (2, 2)
```

```
----------------------------------------
[ 1   2   3   4  ] (1, 4)
----------------------------------------
 1   2   3   4  (4,)
----------------------------------------
 1  (1,)
----------------------------------------
 1  (1,)
----------------------------------------
 (0,)
----------------------------------------
[[ 0.828  0.597
   3      4     ]
 [ 5      6
   7      8     ] ] (2, 2, 2)
========================================================
Testing changing format
 0      0.333  0.667  1
[  0    ,  0.333,  0.667,  1     ]
----------------------------------------
[ 1       1.333   1.667   2
 -1      -0.6667 -0.3333  0      ]
[[  1     ,  1.333 ,  1.667 ,  2      ],
 [ -1     , -0.6667, -0.3333,  0      ]]
----------------------------------------
[[ 4     5.3   6.7   8
  -4    -2.7  -1.3   0    ]
 [ 11    11    12    12
   9     9.3   9.7   10   ]
 [ 0     0.33  0.67  1
  -2    -1.7  -1.3  -1    ] ]
[[[  4   ,  5.3 ,  6.7 ,  8    ],
  [ -4   , -2.7 , -1.3 ,  0    ]],
 [[  11  ,  11  ,  12  ,  12   ],
  [  9   ,  9.3 ,  9.7 ,  10   ]],
 [[  0   ,  0.33,  0.67,  1    ],
  [ -2   , -1.7 , -1.3 , -1    ]]]
----------------------------------------
```

3.14 Tests of various transpose and flips

The file **`test_transpose.py`** tests transpose and flips. Results are at 3.14

```
#!/usr/bin/env python
# (C) 2000 Huaiyu Zhu <hzhu@users.sourceforge.net>.  Licence: GPL
# $Id: test_transpose.py,v 1.2 2000/07/26 20:58:27 hzhu Exp $
"""
Test transpose, diag, flipud, fliplr
"""

from MatPy.Matrix import rand, Diag, diag, flipud, fliplr

a = rand((3,3))
print "-"*44,"\n", a
b = diag(a)
print "-"*44,"\n", b
print "-"*44,"\n", Diag(b)
print "-"*44,"\n", Diag(b.T)

print "-"*44,"\n", flipud(a)
print "-"*44,"\n", fliplr(a)
```

Result obtained with

`>>> from MatPy.tests import test_transpose`

```
--------------------------------------------
[ 0.828  0.489  0.89
  0.0371  0.59   0.979
  0.383  0.201  0.554 ]
--------------------------------------------
[ 0.828  0.59   0.554 ]
--------------------------------------------
[ 0.828  0      0
  0      0.59   0
  0      0      0.554 ]
--------------------------------------------
[ 0.828  0      0
  0      0.59   0
  0      0      0.554 ]
--------------------------------------------
[ 0.383  0.201  0.554
  0.0371  0.59   0.979
  0.828  0.489  0.89  ]
--------------------------------------------
[ 0.89   0.489  0.828
  0.979  0.59   0.0371
  0.554  0.201  0.383 ]
```

3.15 Tests of elementwise comparison

The file **`test_compare.py`** tests elementwise comparisons and summarizations. Results are at 3.15

```
#!/usr/bin/env python
# (C) 2000 Huaiyu Zhu <hzhu@users.sourceforge.net>.  Licence: GPL
# $Id: test_compare.py,v 1.2 2000/09/13 01:52:36 hzhu Exp $
"""
Test comparisons for Matrix module
"""

from MatPy.Matrix import r_range, c_range, rand, any, all, find, sum2
from MatPy.Matrix import max, min, sum, cumsum

a = rand((2,5))
a[1,2] = .5
print a

print "-"*40, "\nComparing two matrices"
b = 1 -a
print a.e_cmp(b)
print a.gt(b)
print a.ge(b)
print a.le(b)
print a.lt(b)


print "-"*40, "\nComparing matrix with number"
b = 0.5
print a.e_cmp(b)
print a.gt(b)
print a.ge(b)
print a.le(b)
print a.lt(b)

print any(a.lt(.1)), any(a.lt(.9))
print all(a.lt(.1)), all(a.lt(.9))
print sum2(a.lt(b))

print "-"*40, "\nComparing row vectors"
a = r_range(5)
b = 4-a
print a, b
print a.e_cmp(b)
print a.gt(b)
print a.ge(b)
print a.le(b)
print a.lt(b)

print any(a.lt(1)), any(a.lt(5))
print all(a.lt(1)), all(a.lt(5))
print sum2(a.lt(b))

print "-"*40, "\nComparing col vectors"
a = c_range(5)
b = 4-a
print a, b
print a.e_cmp(b)
print a.gt(b)
print a.ge(b)
print a.le(b)
print a.lt(b)

print any(a.lt(1)), any(a.lt(5))
print all(a.lt(1)), all(a.lt(5))
print sum2(a.lt(b))

print "-"*40, "max, min, sum, cumsum"
X = rand((9,4))
print X
print max(X)
print min(X)
print sum(X)
print cumsum(X)

print "-"*40, "max, min for multiple matrices"
Y = rand((4,4))
a = max(Y, Y.T, Y)
b = min(Y, Y.T, Y)
```

```
print a
print b
assert a == a.T
assert b == b.T
assert all(a.ge(Y))
assert all(Y.ge(b))

print "-"*40, "max, min for matrices and numbers"
print max(Y, 0.4, 0.6)
print min(Y, 0.4, 0.6)

print "-"*40, "max, min for multiple numbers"
print max(1, 0.5, -1)
print min(1, 0.5, -1)

print "-"*40, "find indices of true elements"
T = Y.lt(0.5)
print find(T)
```

Result obtained with

>>> from MatPy.tests import test_compare

```
[ 0.828  0.718  0.911  0.165  0.286
  0.15   0.134  0.5    0.891  0.0222 ]
----------------------------------------
Comparing two matrices
[ 1   1   1  -1  -1
 -1  -1   0   1  -1  ]
[ 1   1   1   0   0
  0   0   0   1   0  ]
[ 1   1   1   0   0
  0   0   1   1   0  ]
[ 0   0   0   1   1
  1   1   1   0   1  ]
[ 0   0   0   1   1
  1   1   0   0   1  ]
----------------------------------------
Comparing matrix with number
[ 1   1   1  -1  -1
 -1  -1   0   1  -1  ]
[ 1   1   1   0   0
  0   0   0   1   0  ]
[ 1   1   1   0   0
  0   0   1   1   0  ]
[ 0   0   0   1   1
  1   1   1   0   1  ]
[ 0   0   0   1   1
  1   1   0   0   1  ]
[ 0   0   0   0   1  ] [ 1   1   1   1   1  ]
[ 0   0   0   0   0  ] [ 1   1   0   1   1  ]
5
----------------------------------------
Comparing row vectors
[ 0   1   2   3   4  ] [ 4   3   2   1   0  ]
[-1  -1   0   1   1  ]
[ 0   0   0   1   1  ]
[ 0   0   1   1   1  ]
[ 1   1   1   0   0  ]
[ 1   1   0   0   0  ]
[ 1   0   0   0   0  ] [ 1   1   1   1   1  ]
[ 1   0   0   0   0  ] [ 1   1   1   1   1  ]
2
----------------------------------------
Comparing col vectors
[ 0
  1
  2
  3
  4  ] [ 4
  3
  2
  1
  0  ]
[-1
 -1
  0
  1
  1  ]
[ 0
  0
  0
  1
  1  ]
[ 0
  0
  1
  1
  1  ]
[ 1
  1
  1
  0
  0  ]
[ 1
  1
  0
  0
  0  ]
[ 1  ] [ 1  ]
[ 0  ] [ 1  ]
2
---------------------------------------- max, min, sum, cumsum
[ 0.996  0.14   0.342  0.482
  0.347  0.426  0.602  0.345
  0.58   0.0796  0.435  0.373
  0.487  0.0976  0.244  0.637
  0.186  0.579  0.485  0.659
  0.243  0.785  0.412  0.747
  0.439  0.472  0.574  0.367
  0.837  0.659  0.704  0.481
  0.625  0.427  0.184  0.382 ]
[ 0.996  0.785  0.704  0.747 ]
[ 0.186  0.0796  0.184  0.345 ]
[ 4.74   3.66   3.98   4.47  ]
[ 0.996  0.14   0.342  0.482
  1.34   0.566  0.944  0.827
  1.92   0.645  1.38   1.2
  2.41   0.743  1.62   1.84
  2.6    1.32   2.11   2.5
  2.84   2.11   2.52   3.24
  3.28   2.58   3.09   3.61
  4.12   3.24   3.8    4.09
  4.74   3.66   3.98   4.47  ]
---------------------------------------- max, min for multiple matrices
[ 0.321  0.712  0.346  0.658
  0.712  0.226  0.654  0.931
  0.346  0.654  0.275  0.316
  0.658  0.931  0.316  0.305 ]
[ 0.321  0.619  0.0865  0.302
  0.619  0.226  0.636  0.0598
  0.0865  0.636  0.275  0.259
  0.302  0.0598  0.259  0.305 ]
---------------------------------------- max, min for matri-
ces and numbers
[ 0.6    0.712  0.6    0.6
  0.619  0.6    0.654  0.931
  0.6    0.636  0.6    0.6
  0.658  0.6    0.6    0.6   ]
[ 0.321  0.4    0.0865  0.302
  0.4    0.226  0.4    0.4
  0.346  0.4    0.275  0.259
  0.4    0.0598  0.316  0.305 ]
---------------------------------------- max, min for multiple numbers
1.0
-1.0
---------------------------------------- find indices of true elements
[(0, 0), (0, 2), (0, 3), (1, 1), (2, 0), (2, 2), (2, 3), (3, 1), (3, 2), (3, 3)]
```

3.16 Tests of probability distributions

The file **`test_probs.py`** tests classes and functions of probability distributions. Results are at 3.16

```
#!/usr/bin/env python
# (C) 2000 Huaiyu Zhu <hzhu@users.sourceforge.net>.  Licence: GPL
# $Id: test_probs.py,v 1.7 2000/08/06 07:21:17 hzhu Exp $
"""
Statistical distributions
"""

from MatPy.Stats.utils import histplot
if __name__ == "__main__": waittime = None
else: waittime = 1

i = 0
def test(g, dist, x, __name__):
"For dist: generate rand, draw pdf, cdf and cdfc on points x"
global i

print dist.__doc__
print dist.rand((2,5))
```

```
g.title("The %s distribution" % dist.__class__.__name__)

n = 400
y = dist.rand((n,1))

histplot(y, x=30, waittime=1, g=g, scaled=1)
g.holdon
curves = [dist.pdf(x), dist.cdf(x), dist.cdfc(x)]
g.plot(curves, xs=[x]*3, names=["pdf", "cdf", "cdfc"])

g.hardcopy("probs_%s.ps" %i)
i = i + 1
g.holdoff
wait(waittime)

#-------------------------------------------------------------------

from MatPy.gplot import Gplot, wait
g = Gplot()

from MatPy.Stats.distribs import binomial, negbinomial, poisson
from MatPy.Stats.distribs import beta, gamma, normal, chi2, t, F

from MatPy.Matrix import r_range
x = r_range(100)/10.

#-------------------------------------------------------------------

print "-"*30, "Testing distributions in [-Inf, Inf]"
for dist in [normal(), t(2)]:
test(g, dist, x-5, __name__)

print "-"*30, "Testing distributions in [0, Inf]"
for dist in [gamma(2,1), chi2(4), F(5,9)]:
test(g, dist, x+1e-100, __name__)


"""
binomial,
negbinomial,
poisson,
beta,
gamma,
normal,
chi2,
t,
F,
"""
```

Result obtained with

```
>>> from MatPy.tests import test_probs

------------------------------ Testing distributions in [-Inf, Inf]
Normal distribution:
pdf =  1/(b*sqrt(pi)) * exp(-((x-a)/b)^2 / 2)

[-0.946  1.57   1.06   0.253  0.351
```

```
 -0.219 -1.43   1.07   0.582  1.73  ]
<MatPy.gplot.Gplot instance at 0x8199e04>
Student t distribution (n>0):
pdf =  gam((n+1)/2) / (sqrt(n*pi)* gam(n/2)) * (1+xˆ2/n)ˆ(-(n+1)/2)

[-8.9   -0.61   0.397 -0.263 -1
 -0.851 -0.223  0.0603  1.71  -0.419 ]
<MatPy.gplot.Gplot instance at 0x8199e04>
------------------------------ Testing distributions in [0, Inf]
Gamma distribution (a>0, b>0):
pdf = b/gamma(a) * (b*x)ˆ(a-1) * exp(-b*x)

[ 0.948  1.42   0.311  3.95   2.33
  1.04   0.762  1.99   2.3    2.09  ]
<MatPy.gplot.Gplot instance at 0x8199e04>
Chi square distribution with n degrees of freedom (n>0):
pdf_t = 1/(2* gam(n/2)) * (t/2)ˆ(n/2-1) * exp(-t/2)
= pdfg(x, n/2, 1/2)

[ 5.01   5.46   2.91   2.99   4.75
  1.29   1.22   1.74   13     5.41  ]
<MatPy.gplot.Gplot instance at 0x8199e04>
F distribution (n1>0, n2>0):
(also known as Snedcor’s density or the variance ratio density)
pdf = density of (u1/n1)/(u2/n2),
where u1 and u2 are Chi variates with n1 and n2 degrees of freedom.

[ 2.28   0.158  0.474  0.718  1.42
  0.421  3.45   1.76   0.627  1.52  ]
<MatPy.gplot.Gplot instance at 0x8199e04>
```

3.17 Tests of statistical functions

The file **`test_sfuncs.py`** tests statistical functions. Results are at 3.17

```
#!/usr/bin/env python
# (C) 2000 Huaiyu Zhu <hzhu@users.sourceforge.net>.  Licence: GPL
# $Id: test_sfuncs.py,v 1.7 2000/08/06 07:21:17 hzhu Exp $
"""
Test stand-alone functions of some distributions.
"""

from MatPy.Matrix import r_range
from MatPy.Stats.distribs import *
if __name__ == "__main__": waittime = None
else: waittime = 1

i = 0
def test(g, __name__, funcs, x, *args):
"For each f in funcs, draw a line with f(args)"
global i

ys = map(lambda f,args=args:apply(f, args), funcs)
g.plot(ys, names=["pdf", "cdf", "cdfc"], xs=[x]*len(funcs))

g.hardcopy("sfunc_%s.ps" %i)
i = i + 1
```

```
wait(waittime)

#-----------------------------------------------------------------
print "All of these tests are shown in graphics"

from MatPy.gplot import Gplot, wait
g = Gplot()
x = r_range(100)/10.+1e-300

#-------------------------------------------
g.title("Gamma distribution")
a = 2; b = 0.5
test(g, __name__,
 (pdfg, cdfg, cdfcg),
 x, x, a, b)

x = x - 5
#-------------------------------------------
g.title("Normal distribution")
test(g, __name__,
 (pdfn, cdfn, cdfcn),
 x, x)

"""
binomial,
negbinomial,
poisson,
beta,
gamma,
normal,
chi2,
t,
F,
"""
```

Result obtained with

`>>> from MatPy.tests import test_sfuncs`

```
All of these tests are shown in graphics

igam underflow error

igam underflow error
```

3.18 Tests of statistical utilities

The file **`test_sutils.py`** tests statistical utilities: hist, histplot. Results are at 3.18

```
#!/usr/bin/env python
# (C) 2000 Huaiyu Zhu <hzhu@users.sourceforge.net>.  Licence: GPL
# $Id: test_sutils.py,v 1.4 2000/10/05 01:32:27 hzhu Exp $
"""
Testing statistics utilities: hist, histplot
"""
```

```
from MatPy.Stats.distribs import randn
from MatPy.Stats.utils import hist, histplot
from MatPy.gplot import wait

if __name__ == "__main__": waittime = None
else: waittime = 1

data = randn((300,1))

print "calculating data for histogram"
(nn, xx) = hist(data)
print nn.T
print xx.T

print "Default histogram"
g = histplot(data, waittime=waittime)
g.title("Default Histogram")
# Using replot here seems wrong - it tries to find original.  Why?
# g.replot()
# But the hardcopy already contains title, even without replot.
g.hardcopy("hist_0.ps")
wait(waittime)

print "Scaled histogram"
histplot(data, waittime=waittime, g=g, scaled=1)

print "Histogram with spefied bins"
g.title("Histogram with Spefied Bins")
g = histplot(data, randn(20).T, waittime=waittime, g=g)
g.hardcopy("hist_1.ps")
```

Result obtained with
`>>> from MatPy.tests import test_sutils`

```
calculating data for histogram
[ 2      6      23     39     58     61     66     31     12     1     ]
[-2.7   -2.13  -1.56  -0.99  -0.42   0.15   0.72   1.29   1.86   2.43  ]
Default histogram
None
Scaled histogram
<MatPy.gplot.Gplot instance at 0x8186dfc>
Histogram with spefied bins
<MatPy.gplot.Gplot instance at 0x8186dfc>
```

3.19 Tests of tests kalman filter on a random walk

The file **`test_kalman.py`**
. Results are at 3.19

```
#!/usr/bin/env python
# (C) 2000 Huaiyu Zhu <hzhu@users.sourceforge.net>.  Licence: GPL
# $Id: test_kalman.py,v 1.4 2001/08/26 12:40:44 hzhu Exp $
"""
test_kalman.py - testing class LinearSystem with kalman filtering
```

```
System description:
    state:              x+  =   A*x + u + w
    measurement:        z   =   H*x + v

Much of the code is for recording history and displaying.

"""
from MatPy.Matrix import eye, zeros, norm, Matrix_cr, Matrix_r
from MatPy.Stats.distribs import randn
from MatPy.DynSys.kalman import LinearSystem
from MatPy.efuncs import triu
from MatPy.Graphics.trajplot import TrajPlot

class RandomWalk(LinearSystem):

""" RandomWalk: Wiener processes with given kernels """

def init_params(self, n):
""" sets system parameters:
A = [I,I*dt ; I 0]  state (p, v) transition
dt    :  time interval
n     :  dimension of positions
"""
zz = zeros((n,n))
ee = eye(n)
dt = 0.04
self.A = Matrix_cr([[ee, ee*dt], [zz, ee]])
self.Q2 = q = Matrix_cr([[randn((n,n))/n, zz], [zz, randn((n,n))/n]])
self.Q = q * q.T

self.H = Matrix_r([ee, zz])
self.R2 = r = randn((n,n)) * .4
self.R = r * r.T
self.RI = self.R.I
self.n = n*2
self.m = n
return self

def init_history(self):

""" initialize history of system """

self.g = g = TrajPlot(2, names=["System", "Estimate"],
  wait_time=0.1)

g.title("Kalman filter: Trajectories of state and estimate")
g.axis_equal = 1

self.e = [];
print " "*10, " state ", " "*20, "estimate", " "*10, " error "

def add_history(self, x_sys, x_est):

""" add on step to history, print, then plot """

# get past history
e = self.e
```

```
e.append(norm(x_sys - x_est))

# record new data, and plot the first two axes
xx = Matrix((x_sys[0], x_est[0]))
yy = Matrix((x_sys[1], x_est[1]))
self.g.plotadd(xx, yy)

# print state, estimate or their distance
print x_sys.T, x_est.T, "%.4g" % e[i]

def show_history(self, waittime):

""" show entire history """

# plot errors
g = Gplot()
g.title("Kalman filter: Distances between states and estimates")
g.plot([Matrix(self.e)])
wait(waittime)


#-------------------------------------------------------------------
""" Simulate linear system with Kalman filtering """
from MatPy.gplot import Gplot, wait
from MatPy.Matrix import Matrix
from MatPy.efuncs import abs

linsys = RandomWalk().init_params(n=2)
linsys.init_state()
linsys.init_history()

x = randn(linsys.x.shape)
for i in range(20):
x, u = linsys.evolve(x)
z = linsys.measure(x)
tx = linsys.filter(z, u)
linsys.add_history(x, tx)

if __name__ == "__main__": waittime = None
else: waittime = 2
linsys.show_history(waittime)
```

Result obtained with

`>>> from MatPy.tests import test_kalman`

```
            state                           estimate             error
[ 0.511  2.11  -3.58  -1.05  ] [ 0.468  2.23  -1.22   0.33  ] 2.737
[-0.257  1.97  -3.36  -2.91  ] [ 0.252  2.49  -0.259  0.106 ] 4.382
[ 0.376  4.11  -2.4   -1.6   ] [ 0.237  5.37  -0.0149  1.06  ] 3.791
[ 0.0281  4.09  -3.8   -0.969 ] [ 0.0747  4.52  -1.53   0.855 ] 2.947
[-0.203  4.6   -4.1   -2.59  ] [ 0.000496  4.59  -0.282 -0.148 ] 4.535
[-2.25   4.64  -4.12  -4.01  ] [-2.14   4.3    0.683 -0.268 ] 6.097
[-0.499  3.74  -4.69  -2.47  ] [-0.33   3.72  -0.483  0.408 ] 5.106
[ 1.98   2.38  -6.63  -4.76  ] [ 1.5    2.99  -1.84  -1.53  ] 5.827
[ 1.16   1.11  -8.27  -3.61  ] [ 0.914  1.64  -1.64  -0.862 ] 7.203
[ 1.5   -1.53  -8.55  -1.27  ] [ 1.24  -1.38  -2.56  -0.747 ] 6.018
[ 1.92  -2.4   -8.77  -2.41  ] [ 2.03  -2.28  -3.45  -1.63  ] 5.375
[ 2.75  -1.74  -9.9   -2.19  ] [ 2.23  -1.2   -4.94  -0.683 ] 5.244
[ 3.32  -1.48  -10    -2.94  ] [ 3.54  -1.79  -5.61  -0.753 ] 4.965
[ 2.29  -1.09  -10.7  -3.22  ] [ 2.24  -1.4   -6.61  -1.35  ] 4.538
[ 1.95   1.13  -11.4  -3.15  ] [ 1.37   0.714 -7.24  -0.65  ] 4.929
[ 1.13   3.68  -9.84  -2.15  ] [ 1.79   3.57  -3.84   1.01  ] 6.811
[ 0.843  2.93  -12    -1.97  ] [ 1.19   3.66  -6.33   2.28  ] 7.17
[ 0.559  4.84  -9.96  -1.44  ] [ 0.978  5.35  -5.74   1.76  ] 5.344
[-0.445  6.32  -10.5  -1.15  ] [ 0.147  6.88  -5.65   2.75  ] 6.257
[ 0.702  6.76  -8.87  -1.82  ] [ 0.74   7.05  -4.54   2.5   ] 6.127
```

4 Graphics with gnuplot

4.1 Graphics generated with gnuplot

These postscript files are generate from **`test_gplot.py`**.

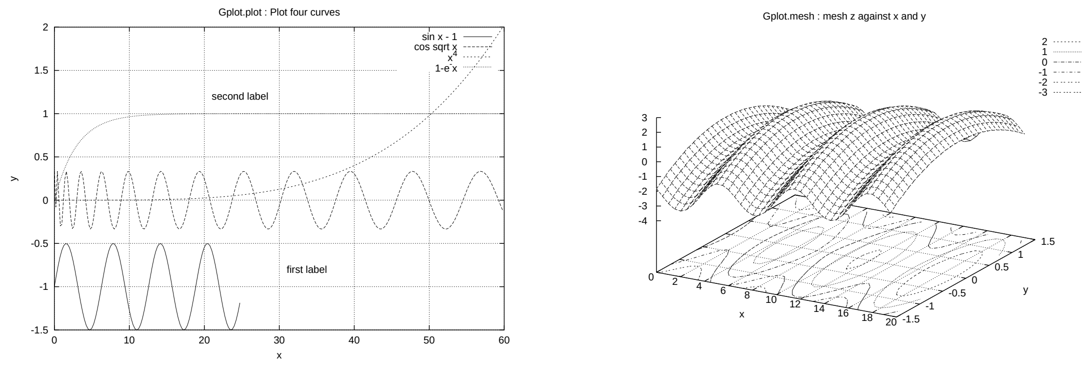

Figure 1: Result using plot and mesh

4.2 Graphics of elementwise functions

These postscript files are generate from **`test_efuncs.py`**.

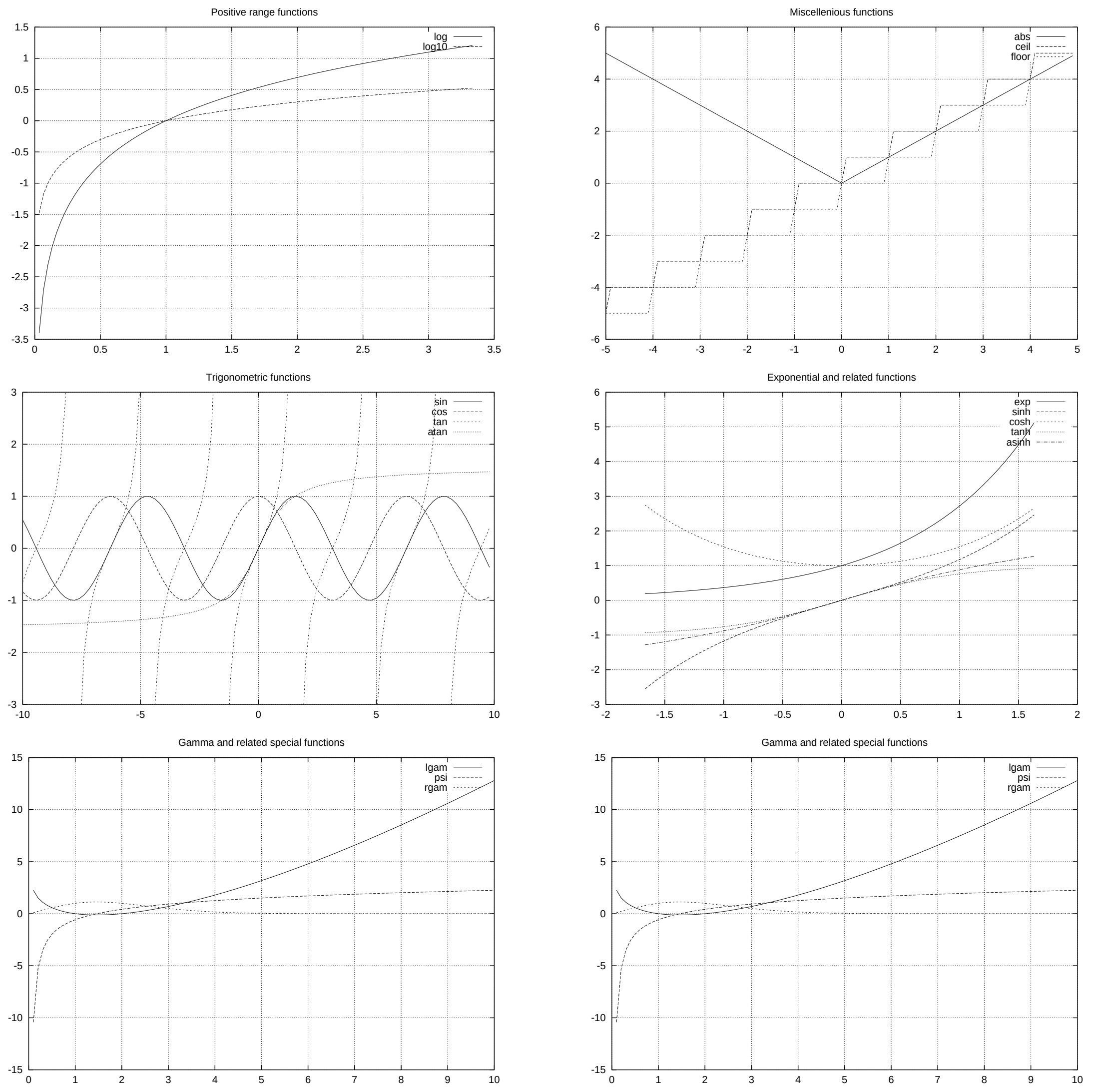

Figure 2: Elementwise functions

4.3 Graphics of statistical functions

These postscript files are generate from **`test_sfuncs.py`**.

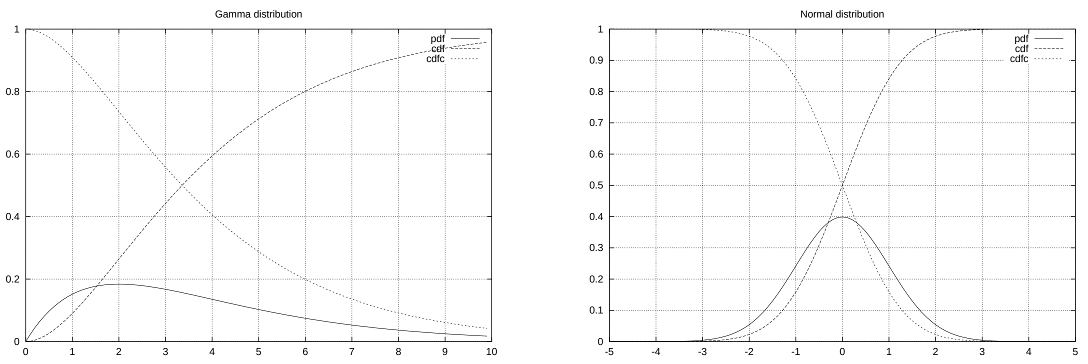

Figure 3: Statistical functions

4.4 Graphics of probability distributions

These postscript files are generate from **`test_probs.py`**.

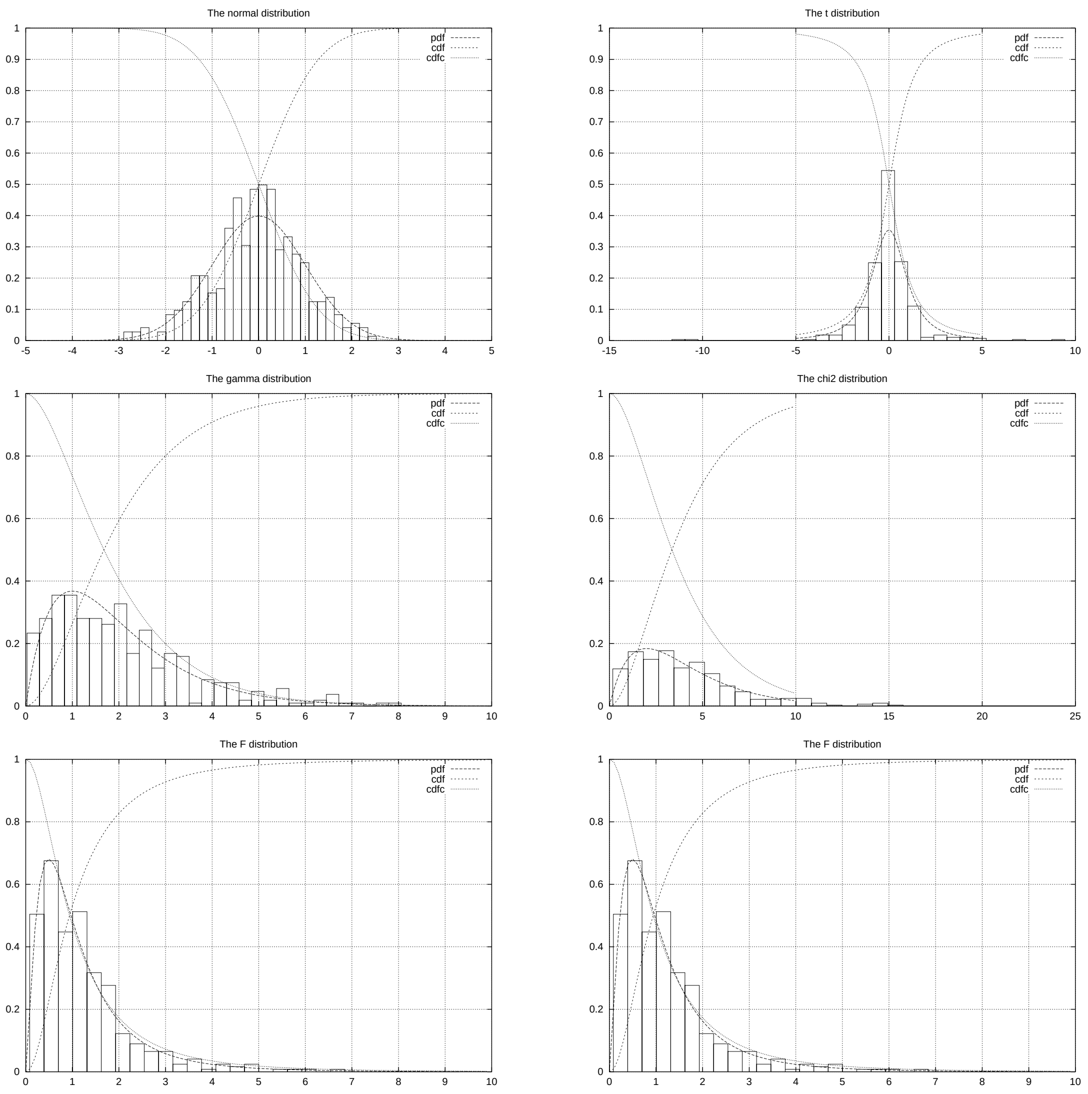

Figure 4: Probability distributions

4.5 Graphics of histograms

These postscript files are generate from **`test_sutils.py`**.

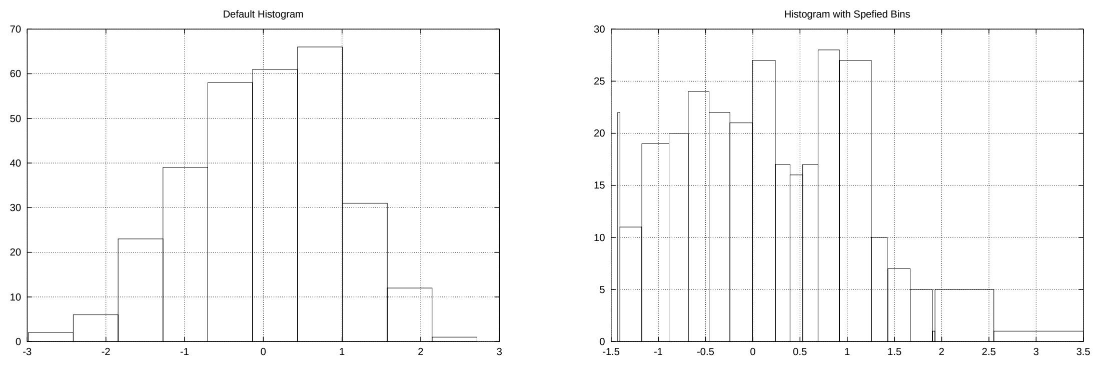

Figure 5: Histograms functions

4.6 Gnuplot preferences for X11

If you are using X (the windowing system on Unix-like OSes) you might want to tweak you `~/.Xdefaults` to change preference for gnuplot. Here's my setting:

```
gnuplot*background: black
gnuplot*textColor: white
gnuplot*borderColor: white
gnuplot*axisColor: white
gnuplot*line1Color: red
gnuplot*line2Color: green
gnuplot*line3Color: blue
gnuplot*line4Color: magenta
gnuplot*line5Color: cyan
gnuplot*line6Color: sienna
gnuplot*line7Color: orange
gnuplot*line8Color: coral
gnuplot*Font: 7x14

#gnuplot*backgroundGray: black
gnuplot*textGray: white
gnuplot*borderGray: gray50
gnuplot*axisGray: gray50
gnuplot*line1Gray: gray100
gnuplot*line2Gray: gray60
gnuplot*line3Gray: gray80
gnuplot*line4Gray: gray40
gnuplot*line5Gray: gray90
gnuplot*line6Gray: gray50
gnuplot*line7Gray: gray70
gnuplot*line8Gray: gray30

gnuplot*borderWidth: 2
gnuplot*axisWidth: 2
gnuplot*line1Width: 2
gnuplot*line2Width: 2
gnuplot*line3Width: 2
gnuplot*line4Width: 2
gnuplot*line5Width: 2
gnuplot*line6Width: 2
gnuplot*line7Width: 2
gnuplot*line8Width: 2
```

5 List of names

These are just what's implemented now. Many more can be added here easily.

THIS IS INCOMPLETE LIST!! For a complete list type **`functions`** in **`matpy.py`**.

5.1 Modules and classes

This is a very incomplete list

- Modules: **Matrix, gplot, efuncs, mfuncs, Tensor, Scalar, Stats.distribs, DynSys.kalman**,
- Classes: **Matrix.Matrix, gplot.Gplot, Tensor.Tensor, Scalar.Scalar, Stats.dsitribs.[distribution name]**

5.2 Functions

- Special values: **NaN, nan, Inf, inf**,
- Functions that generates matrices: **Mrange, zeros, ones, eye, rand**,
- Elementwise functions: **abs, ceil, floor, sign**, **sqrt, sinc**, **exp, log, log10, sinh, cosh, tanh**, **sin, cos, tan, asin, acos, atan**,
- Matrix functions: **sqrtm, expm, logm, sinhm, coshm, tanhm, sinm, cosm, tanm, asinm, acosm, atanm**,
- Columnwise functions (result is a single row): **sum, prod, mean, median, std, min, max, ptp**,
- Columnwise functions (result is a matrix of same size): **sort, diff, trapz, cumsum, cumprod**,
- Functions giving different sized objects: **cov, diag, sums, norm, mnorm**,

5.3 Imported names in matpy interpreter

```
   This is MatPy mode. Type help to get started

nan:
exp(x) : elementwise exponential of x
solve(a,b) : return LMS solution of a*x==b
map1(f, *args): a costly reimplementation of map
acosm(x) : matrix arc cosine of x
cumprod(x) : return columnwise cumulative prod
map2(f, *args): elementwise f on matrices args
trapz(x) : return columnwise integration using trapzoidal rule
svd(x) : return Matrix u, array s, Matrix v as svd decomposition
max2(x) : return max of all elements
std(x) : return covariance matrix for columns of x
Stats.distribs : Defines statistical distributions
atan(x) : elementwise arc tangent of x
mmap2(f, *args): return Matrix as elementwise f on matrices args
wait(wtime, str, prompt) : wait wtime or
Int:
sinc(x) : elementwise sin(x)/x
Interp:
min2(x) : return min of all elements
__name__:
Matrix_cr(x) : return a matrix from a double list as a column of rows
sinh(x) : elementwise hyperbolic sine of x
acos(x) : elementwise arc cosine of x
ibeta(x,a,b) : elementwise incomplete beta integral of x,
sinm(x) : matrix sine of x
beta(a,b) : elementwise beta function = gamma(a) * gamma(b) / gamma(a+b)
This module provides access to some objects used or maintained by the
```

```
Diag(x) : return diag matrix if x is row or col vector
efunc(f, *args) : elementwise function f of args
max(x) : return columnwise maximum if one argument,
flipud(x) : return up-down flipped
Matlab(tm) compatibility functions.
class MatrixType : base class for Matrix - only defines repr and str
names          to see list of current objects (functions, classes,...)
cephes:
sin(x) : elementwise sine of x
prod(x) : return columnwise product
to_arrayC(data) : convert to complex array
RandomArray:
exit([status])
mreduce2(f, *args): return Matrix as elementwise f on matrices args
lgam(x) : elementwise log gam |x|
mfuncC(f, x) : matrix function with possibly complex eigenvalues.
sqrt(x) : elementwise square root of x
to_number(data) : conbert to number if possible
sum(x) : return columnwise sum
diag(x) : return diagonal of x as row vector
class Gplot(Gnuplot) : gnuplot window as an object with methods for
min(x) : return columnwise minimum if one argument,
diff(x) : return columnwise diff, keeping first row.  Inverse of cumsum
asin(x) : elementwise arc sine of x
class Tensor : rudimentary wrapper around NumPy arrays
sinhm(x) : matrix hyperbolic sine of x
any(x) : return row 0-1 vector, indicates each column being partly true
exit([status])
logm(x) : matrix logarithm of x base e
inf:
to_array(data) : convert to array
triu(x, k=0) : upper triangular part (on >=k)-th diagonal) of x.
class ScalarType : base class for Scalar - only defines repr
tanm(x) : matrix tangent of x
NaN:
LinearAlgebra:
cosh(x) : elementwise hyperbolic cosine of x
interp:
cosem(x) : matrix cosine of x
demo : Run demos one by one
DynSys.kalman : Defines LinearSystem with kalman filtering
MatPy.Stats : statistics oriented stuff
atanm(x) : matrix arc tangent of x
tril(x, k=0) : lower triangular part (on <=k-th diagonal) of x.
mfuncs.py : Matrix-wise functions
mean(x) : return columnwise mean
Inf:
efuncRC(*args) : elementwise function of matrices args,
r_range(*i) : row vector of range(i)
eig(x) : return eigenvalues as an array, eigenvectors as a Matrix
cols(x) : split x into a list of columns
class Scalar : class of numbers with customizable formats
flipud(x) : return left-right flipped
igamc(x>0,a>0) : elementwise complemented incomplete gamma integral
rgam(x) : elementwise 1 / gam(x)
Numeric module defining a multi-dimensional array and useful proce-
dures for
```

```
to_list(data) : convert to double list
median(x) : return columnwise median
efuncC(f, *args) : elementwise function f of args with possible com-
plex elements
tanh(x) : elementwise hyperbolic tangent of x
igami(y, a>=0) : elementwise inverse incomplete gamma integral
cose(x) : elementwise cosine of x
mfunc(f, x) : matrix function f of matrix x.
std(x) : return columnwise standard deviation
MatPy.DynSys : Modules for dynamical systems
efuncs.py - element-wise functions
floor(x) : elementwise largest integer not greater than x
ones(size) : matrix of ones of given size
gam(x) : elementwise gamma function of x
ceil(x) : elementwise smallest integer not less than x
sign(x) : elementwise sign of x in (-1, 0, +1)
to_Matrix(data) : covert to Matrix if possible
tan(x) : elementwise tangent of x
__file__:
asinm(x) : matrix arc sine of x
expm(x) : matrix exponential of x
A collection of string operations (most are no longer used in Python 1.6).
psi(x) : elementwise derivative of log gam(x)
asinh(x) : elementwise inverse hyperbolic sine of x
sort(x) : return columnwise sorted
norm(x) : return Frobenius norm of x
__doc__:
log10(x) : elementwise logarithm of x base 10
names:
Matrix_r(x) : return a matrix from a list as a row
__builtins__:
igam(x>0, a>0) : elementwise incomplete gamma integral
zeros(a, typecode) : return a zero tensor
c_range(*i) : col vector of range(i)
tanhm(x) : matrix hyperbolic tangent of x
reduce2(f, *args): elementwise f on matrices args
Matrix_c(x) : return a matrix from a list as a column
reduce1(f, list): a costly reimplementation of reduce
lookfor(name) : Search docs for keyword <name>
log(x) : elementwise logarithm of x base e
eye(size) : identity matrix of given size
rows(x) : split x into a list of rows
solve(a,b) : return LMS solution of a*x==b
approx_real(x) : return x.real if |x.imag| < |x.real| * _eps_approx
size(data) : return data.shape
unit(n, m) : unit vector of dimension n, pointing at mth direction
find(x) : return list of indices (i,j) for which x[i,j] is true
cumsum(x) : return columnwise cumulative sum
norm1(x) : return columnwise norms
diff(x) : return columnwise diff
sqrtm(x) : matrix square root of x
coshm(x) : matrix hyperbolic cosine of x
Statistics utilities
lbeta(a,b) : elementwise log |beta(x)|
Mrange(*i) : row vector of range(i) - deprecated
MatPy : Matrix package for python, with visualization.
solve(a,b) : return solution of a*x==b
```

```
abs(x) : elementwise absolute value of x
sum2(x) : return sum of all elements
ibetai(y, a>=0, b>=0) : elementwise inverse incomplete beta integral
mfunc_p(f, x) : matrix function with positive eigenvalues
class Matrix : main matrix class
ptp(x) : return columnwise ptp?
norm(x) : return Hilbert-Schmidt (?) norm of x
rand(size) : standard uniformly distributed random matrix of given size
all(x) : return row 0-1 vector, indicates each column being all true
```

6 A dedicated interactive console

You can start the whole thing by simply typing **`matpy.py`** in a shell which starts python and does the necessary imports and implements some other goodies. Excerpt from the author (Henning Schroeder):

"... In order to play with your package I hacked together a small interactive console so that you can quickly enter expressions without caring about importing all the stuff etc. Additionally I added a small help system and a history (with load and save). ..."

Currently the **`lookfor`** function is not implemented.

An example session follows

```
$ matpy.py

Python 1.5.2 (#1, Sep 17 1999, 20:15:36)  [GCC egcs-
2.91.66 19990314/Linux (egcs- (<MatPy-console>)
Copyright 1991-1995 Stichting Mathematisch Centrum, Amsterdam

    Command completion bound to tab.  You can change it to C-
g by typing
    readline.parse_and_bind('C-g: complete')

You are working in MatPy mode! Type help to get started


>>> help

Type
    help(function)   to get more information on a special function
    lookfor(name)    to search the help for name
    functions        to see a list of available functions

    log_on()         to start logging
    log_off()        to stop logging
    dump()           to show the history
    load('filename') to load filename
    save('filename') to save filename

    CTRL-D to exit

It is recommed to read the included documenta-
tion so that you understand
the way MatPy works.
```

>>> help(Matrix)

```
Matrix class similar to matlab
```

>>> functions

```
Matrix_c, Matrix_r, Mrange, abs, acos, acosm, ap-
prox_real, arange, asin,
asinm, atan, atanm, cdf_g, cdfc_g, ceil, com-
pactAxes, cos, cosh, coshm,
cosm, cov, cumprod, cumsum, diag, diff, efunc, eig, erf, exp, expm, eye,
floor, format, formatC, lgam, log, log10, logm, long, longC, max, mean,
median, mfunc, mfuncC, mfunc_p, min, mnorm, norm, ones, pdf_g, prod, ptp,
rand, repr, reprC, short, shortC, sign, sin, sinc, sinh, sinhm, sinm, size,
solve, sort, sqrt, sqrtm, std, str, strC, sum, sums, svd, tan, tanh, tanhm,
tanm, to_Matrix, to_array, to_number, trapz, wait, zeros
```

7 Design and implementation issues

In these discussions **`Matlab`** and **`Octave`** will be used interchangeably when expressions are concerned, because they are essentially the same.

7.1 Vectors as matrices

There are very good reasons that vectors should be treated as $1 \times n$ and $n \times 1$ matrices:

- This is so when $n \times m$ matrices are considered as linear transforms from m-dimensional to n-dimensional spaces.
- Vectors can be sliced off matrices. Matrices can be built from vectors. They obey the same rules. Most operations should be valid for both.
- All the rules for vectors and covectors map naturally into the rules for column and row vectors.

The array type in **`NumPy`** behaves more like a diagonal matrix than ordinary vectors. Therefore we implement vectors as matrices instead.

However, it might be worthwhile to implement a **`Vector`** subclass for efficiency reasons. I do not know how to let **`Matrix(a)`** sometimes return a **`Vector`** depending on **`a.shape`**.

7.2 Do we need a scalar class?

Superficially scalars can be treated as 1×1 matrices. However, this is not the case as scalar multiplication is a different beast from Matrix multiplication. In this regard a should be treated as aI.

On the other hand, $A + a$ in **`Matlab`** behaves like $A + a\infty\infty^T$. We adopt this convention here, provisionally, as this is also the default of **`NumPy`**. This need more discussions.

At the moment scalar are represented by native **`Python`** numbers. However, we also implemented a **`Scalar`** wrapper class so that it can have its own class methods such as `__repr__` and be subclassed. Hopefully in the future python can have a native **`Number`** class.

7.3 Efficiency issues

In Numeric the row and columns are implemented asymmetrically. In **`MatPy`** we try to provide both types of operations by transposing both the operands and the results. I do not know the cost of these overheads. In the future it might be worthwhile to implement both versions in C.

Because the types of **`NumPy`** and Mat do not match exactly, there are many conditional statements in the initializers. Such costs will be eliminated when these classes are implemented natively.

The matrix functions like **`sqrtm, expm, logm`**, etc, are implemented by eigenvalue decomposition through an intermediate **`mfunc`**. It looks worthwhile to re-implement this in C.

7.4 Robustness, especially complex numbers

`NumPy` gives off **`NaN`** far too often compared with **`Octave`**. This is understandable from the CompSci origin. **`Python`** is heavily integer-oriented (**`1/2==0`**) while **`NumPy`** still favors real numbers much more than complex ones (**`sqrt(-2)==nan`**). These are very awkward for matrix computations because many moderately complicated matrix computations quickly leads to full of **`nan`**s.

The short term solution is to add **`0j`** to the operands to force complex computation.

Long term solution is have more robust implementations. The experience of **`Octave`** and its large code base (which dates back to some excellent Fortran packages) should be well utilized in this regard. It is not clear to me how, but it could be extremely useful, to directly implement many operations in terms of the **`Octave`** source code base.

7.5 Elementwise versus matrix functions

Both elementwise and matrix functions are very useful in practice. In **`Matlab`** there are two versions of operators: the plain operators (matrix operation like **`*`**) and dot operators (elementwise operations like **`.*`**), and two versions of functions: the plain functions (elementwise like **`exp`**) and m-suffixed functions (matrix functions like **`expm`**).

Two difficulties exist,

- **`Python`** do not recognize these "dot operators". This may be overcome by patching the parser like what has been done for the "augmentation operators" **`+=`** and **`*=`**, etc. Discussions and experiments are underway. Right now **`A .* B`** can be invoked by `A.__dotmul__(B)`.
- The default behavior in **`Matlab`** is for the symbolic operators like **`*, /`** is different from that for the named functions like **`exp, sin`**. This leaves the function **`pow`** potentially confusing. Is **`pow(x,2)==x.**2`** or **`pow(x,2)==x**2`**? The safest approach appears to be leaving out this function.