Import NumPy and SciPy

```
In [3]: import numpy as np
```

```
In [4]: import scipy as sp
```

Differences in temperament between NumPy and SciPy; see, e.g., http://www.scipy.org/NegativeSquareRoot

```
In [5]: exp(pi*np.sqrt(-1)) + 1
```

```
In [6]: exp(pi*sp.sqrt(-1)) + 1
```

Getting data in and out of SciPy

Import Matlab data into python

```
In [7]: import scipy.io as sio
```

```
In [8]: struct = sio.loadmat('testbox.mat') # Imports Matlab data structure
```

```
In [9]: struct
```

```
In [10]: box = struct['box'] # Extracts data object from structure
```

Construct sequence of n (linearly-spaced) numbers, from a to b

```
In [6]: a = 50; b = 100-0.1; n = 57;
```

```
In [7]: sp.linspace(a,b,n)
```

Construct sequence of n (base-10 logarithmically-spaced) numbers between 10^a and 10^b

```
In [11]: a = -1; b = 1; n = 20;
```

In [13]:
```
sp.logspace(a,b,n)
```

Construct coordinate array

In [19]:
```
x,y = np.mgrid[0:5,0:5]
```

In [15]:
```
x
```

In [16]:
```
y
```

In [17]:
```
np.sqrt(x**2 + y**2)
```

Construct tiled array

In [20]:
```
x = np.linspace(0,10,11);
```

In [21]:
```
x
```

In [22]:
```
np.c_[x,x**2]
```

SymPy Introduction

In [23]:
```
from sympy import *
```

In [24]:
```
1/2 + 1/3
```

In [25]:
```
Rational(1,2) + Rational(1,3)
```

In [26]:
```
Rational(5,6).evalf(6)
```

In [27]:
```
1./2. + 1./3.
```

Calculus with symbolic variables

In [28]:
```
x = Symbol("x");
```

In [29]:
```
limit( sin(x) / x, x, 0)
```

In [30]:
```
diff(erf(x),x)
```

In [31]:
```
integrate(1./sqrt(pi) * exp(-x**2),x)
```

In [32]:
```
integrate(1./sqrt(pi) * exp(-x**2), (x,-oo,0) ).evalf(3)
```

Matrix computations

In [33]:
```
M = Matrix( ([1, x], [x, 1]) )
pprint(M)
```

In [39]:
```
%load_ext sympyprinting
```

In [40]:
```
M.inv()
```

In [74]:
```
M.cholesky()
```

Interfacing with other languages

See files in f2py_example. Things to note:

i) differences between subroutines and functions in Fortran are not really important here

ii) passing arrays is fine, but you need to pass their dimensions as well

iii) In general, SciPy contains many useful functions that work as quickly as Fortran/C. But if you need to crunch something inside a loop, or nested loops, a

considerable speed-up is possible

In [80]:
```
!f2py --fcompiler=gfortran -c example.f90 -m example
```

In [ ]:
```
import example
```

In [ ]:
```
a = 14; b = 78;
```

In [ ]:
```
c,d = example.arithmetic(a,b)
```

A weave example (courtesy of Nat Butler) is below

In [91]:
```
from scipy import weave
```

In [92]:
```
x = np.arange(1000)/1000.
y = np.zeros(1000,dtype=np.double)
n=len(x)
```

In [100]:
```
code = """
    int i;
    for (i=0;i<n;i++) {
      if (*x<0.5) *y = exp(-(*x)*2);
      else *y = exp(-(*x));
      x++; y++;
    }
"""
```

In [101]:
```
weave.inline(code,['n','x','y'])
```

In [103]:
```
print y
```