In [77]:
```
import numpy as np
import matplotlib as mp
import matplotlib.pyplot as plt
```

Problem 5: Limits in R(p,q)

In [31]:
```
n = 1.
previous_result = (1.+1./n)**n #Initialize the result and previous res
ult

table = np.array([[np.log10(n),previous_result]])

n*=10.
result = (1.+1./n)**n

table = np.concatenate((table,[[np.log10(n),result]]))

while np.rint(result*10.**11.)!=np.rint(previous_result*10.**11.): # s
o long as they don't agree to 12 decimals...
    n*=10.
    previous_result = result
    result = (1.+1./n)**n
    table = np.concatenate((table,[[np.log10(n),result]])) #update the
table

print ('Log10(n_stop) = ', np.log10(n)-1.)
print ('e = ', np.rint(result*10.**11.)/(10.**11.))
print ('    Log10(n)        e')
print (table)
```

```
Log10(n_stop) =  13.0
e =  2.71611003409
    Log10(n)        e
[[ 0.          2.        ]
 [ 1.          2.59374246]
 [ 2.          2.70481383]
 [ 3.          2.71692393]
 [ 4.          2.71814593]
 [ 5.          2.71826824]
 [ 6.          2.71828047]
 [ 7.          2.71828169]
 [ 8.          2.7182818 ]
 [ 9.          2.71828205]
 [10.          2.71828205]
 [11.          2.71828205]
 [12.          2.7185235 ]
 [13.          2.71611003]
 [14.          2.71611003]]
```

In [32]:
```
np.e # The true value of e
```

Out[32]: 2.718281828459045

Something went wrong here, clearly. What, exactly? We were doing well, up until about n = 10^8. The problem is that the error in x^n scales linearly with n. When n becomes greater than or comparable to 1/eps, the error begins to grow. This explains the breakdown in our series. Since numpy uses double precision (I think...), the breakdown occurs at around 10^9 ~ 2^32. This breakdown of the algorithm does not, however, mean that the loop stops. Why does it stop at all? The error has plateaued. It will eventually continue to increase again.

Problem 6: Fun with Square Roots