In [2]:
```
from sympy import *
from sympy.plotting import (plot, plot_parametric)
```

INTRODUCTION TO SERIES IN PYTHON

Recall that a series converges if the sequence of partial sums converges. In addition, we have several convergence tests which are based on the sequence of the terms of the series (NOTE the distinction between the sequence of terms and the sequence of partial sums!). Therefore, many of the commands used for sequences in Lab 6 will also be used for series, including the numerical plotting command in the **matplotlib.pyplot** package. As done in Lab 6, we simplify our notation below:

In [3]:
```
import matplotlib.pyplot as plt
```

We can now use the numerical plot command as plt.plot.

NOTE: The example here (and in future overviews) is NOT a copy/paste to solve the problems in lab. However, it will USE many of the features you will use to solve your problems, such as (in this case) listing sequences and partial sums, plotting sequences and partial sums, summing series, and using for loops to create recursive sequences.

EXAMPLE 1:

1) Given the series tan(1/n)/n^2, n=1..oo)

a) List the first ten terms of the series and the first ten partial sums

b) Plot the first 50 terms of the series and the first 50 partial sums on the same graph.

c) Show that the series converges by the Integral Test.

d) Use the Remainder Formula for the Integral Test to determine how many terms are needed to sum the series to with 0.001. Check by computing the partial sum and the actual sum of the series.

For part a), we need to use list comprehension for the terms and a command in the numpy library for the partial sums. NOTE that instead of defining the expression separately, we could just define

a1_10=[tan(1/n)/n** 2 for i in range(1,11)]

In [46]:
```
n=symbols('n',integer=True) #NOTE that we can assume n to be an integer-and also positive if neces
sary
a=tan(1/n)/n**2 #the sequence representing the terms of the series
a1_10=[a.subs(n,i) for i in range(1,11)]
# Recall the 'range' command creates a list of integers from 1 INCLUSIVE to 11 EXCLUSIVE
print('The list of terms is',a1_10)
import numpy as np
s1_10=np.cumsum(a1_10)
#cumsum is short for "cumulative summation"-basically the Nth partial sum of the list!
print('The list of partial sums is',s1_10)
# We probably want decimal representation of both sequences.
# Recall you cannot evalf a list, so it requires list comprehension
afloat=[i.evalf() for i in a1_10]
print('As floating point, the list of terms is',afloat)
sfloat=[i.evalf() for i in s1_10]
print('As floating point, the list of partial sums is',sfloat)
```

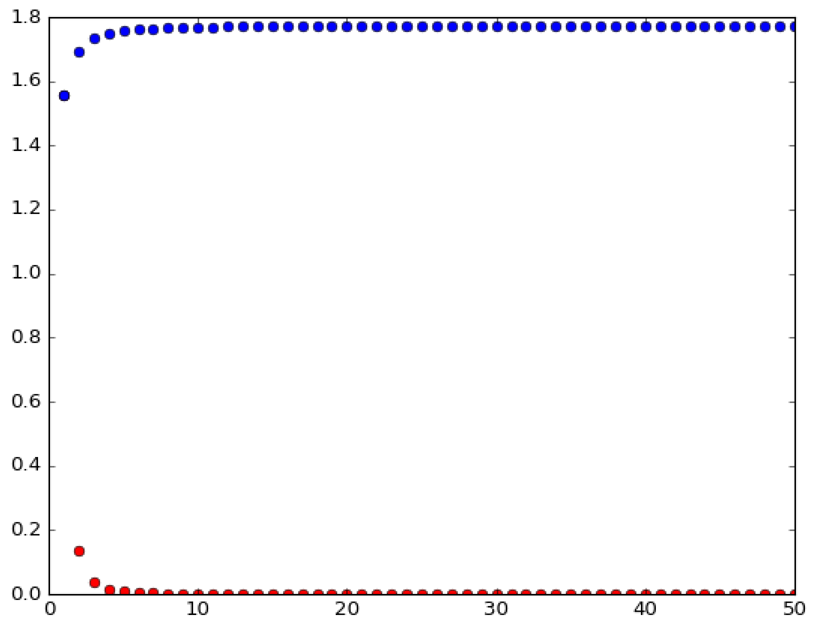
```
The list of terms is [tan(1), tan(1/2)/4, tan(1/3)/9, tan(1/4)/16, tan(1/5)/25, tan(1/6)/36, tan(1/
7)/49, tan(1/8)/64, tan(1/9)/81, tan(1/10)/100]
The list of partial sums is [tan(1) tan(1/2)/4 + tan(1) tan(1/3)/9 + tan(1/2)/4 + tan(1)
 tan(1/4)/16 + tan(1/3)/9 + tan(1/2)/4 + tan(1)
 tan(1/5)/25 + tan(1/4)/16 + tan(1/3)/9 + tan(1/2)/4 + tan(1)
 tan(1/6)/36 + tan(1/5)/25 + tan(1/4)/16 + tan(1/3)/9 + tan(1/2)/4 + tan(1)
 tan(1/7)/49 + tan(1/6)/36 + tan(1/5)/25 + tan(1/4)/16 + tan(1/3)/9 + tan(1/2)/4 + tan(1)
 tan(1/8)/64 + tan(1/7)/49 + tan(1/6)/36 + tan(1/5)/25 + tan(1/4)/16 + tan(1/3)/9 + tan(1/2)/4 + ta
n(1)
 tan(1/9)/81 + tan(1/8)/64 + tan(1/7)/49 + tan(1/6)/36 + tan(1/5)/25 + tan(1/4)/16 + tan(1/3)/9 + t
an(1/2)/4 + tan(1)
 tan(1/10)/100 + tan(1/9)/81 + tan(1/8)/64 + tan(1/7)/49 + tan(1/6)/36 + tan(1/5)/25 + tan(1/4)/16
+ tan(1/3)/9 + tan(1/2)/4 + tan(1)]
As floating point, the list of terms is [1.55740772465490, 0.136575622460948, 0.0384726166122862,
0.0159588700763148, 0.00810840142034690, 0.00467297828617340, 0.00293544815175900, 0.00196336150898
642, 0.00137741515844198, 0.00100334672085451]
As floating point, the list of partial sums is [1.55740772465490, 1.69398334711585, 1.7324559637281
4, 1.74841483380445, 1.75652323522480, 1.76119621351097, 1.76413166166273, 1.76609502317172, 1.7674
7243833016, 1.76847578505101]
```

It looks like the terms will approach 0 (fairly obvious, especially if you **evalf** them), but the partial sums may be approaching 1.77? Let's check the plot in part b)-remember to use plt.plot to plot lists of values. Also note the 'o' option to plot points rather than connecting them. The r and b refer to the color, though that is mainly for your view since you are likely printing black & white.

Also, notice that multiple graphs do NOT have to be put in "tuples" unlike symbolic plots!

In [47]:
```
matplotlib notebook
```

In [48]:
```
nvals=range(1,51) #again note the exclusive right endpoint!
a1_50=[a.subs(n,i) for i in nvals]
s1_50=np.cumsum(a1_50)
plt.plot(nvals,a1_50,'ro',nvals,s1_50,'bo')
```



Out[48]:
```
[<matplotlib.lines.Line2D at 0x7f5e6f4fcf60>,
 <matplotlib.lines.Line2D at 0x7f60e59accc0>]
```

For part c, we just **integrate** the real-valued function f(x)=atan(1/x)/x^2 to see if it is finite or not.

In [49]:
```
x=symbols('x')
f=tan(1/x)/x**2
IntTest=integrate(f,(x,1,oo))
print('The improper integral converges to',IntTest,'so the series converges')
```

```
The improper integral converges to log(1 + tan(1)**2)/2 so the series converges
```

For part d, recall the Remainder Formala is |S-S_N|<the integral of f(x) from N to oo. So if we can make the larger side (the right-hand side) < .001, we automatically get the smaller side (the left-hand side) < .001. So following our algorithmic process, we do the following:

a) **integrate** f from N to infinity

b) **solve** the integral = .001. NOTE we may have to use **nsolve** if Python cannot algebraically solve the equation.

In [50]:
```
x=symbols('x',real=True)
N=symbols('N',real=True)
f=tan(1/x)/x**2
ErrBound=integrate(f,(x,N,oo))
eqn=ErrBound-.001
print(eqn)
Nmin=solve(eqn,N)
print('The number of terms needed is at least',Nmin)
```

```
log(tan(1/N)**2 + 1)/2 - 0.001
The number of terms needed is at least [-22.3644069897262, 22.3644069897262]
```

Therefore, we need at least 23 terms to sum the series to within .001.

For the rest of part d, use the **sum** command to compute the 23rd partial sum. There is another command, **nsum** to try summing the infinite series. To do this, we need to import it and one other command from the **mpmath** library:

In [51]:
```
a1_24=[a.subs(n,i).evalf() for i in range(1,24)]  #Remember the right endpoint is EXCLUSIVE!!!
S24=sum(a1_24)
print('The approximation S(24) is',S24)

from mpmath import nsum, inf
#The nsum command uses inf for infinity, NOT oo!
# The lambda command creates a FUNCTION which is needed for the nsum command
S=nsum(lambda x: tan(1/x)/x**2,[1,inf])
print('The actual sum of the series is about',S)
print('(NOTICE that the series does NOT sum to the same value as the improper integral in part
 c!!!)')
print('The difference between S and S(24) is',abs(S-S24))
```

```
The approximation S(24) is 1.77210227292052
The actual sum of the series is about 1.77300752350479
(NOTICE that the series does NOT sum to the same value as the improper integral in part c!!!)
The difference between S and S(24) is 0.000905250584271267
```

In [ ]: