In [1]:
```
import sympy as sp
from sympy.plotting import (plot, plot_parametric)
```

CONVERGENCE TESTS AND ERROR ESTIMATES FOR SERIES

So far, you have been introduced to five ways to test for the convergence of a series:

1) Test for Divergence (if a(n) does NOT approach 0: Python command **limit**)

2) Integral Test (if the improper integral of the corresponding function converges or not: Python command **integrate**)

3) Comparison Test (if a(n) < b(n) whose series is convergent OR a(n) > b(n) whose series is divergent: Compare graphically with Python command **plot**)

4) Limit Comparison Test (if a(n)/b(n) is finite and positive: Python command **limit**)

5) Alternating Series Test (if a(n) alternates sign and |a(n)| approaches 0: Python command **limit**)<--should be learning now

2) and 5) have error estimates to approximate a series S with a partial sum S(N):

(#2) Integral Test Remainder: S-S(N) < integral(f(x), (x,N,oo))

(#5) AST Remainder: |S - S(N)| < |a(N+1)|

We will go through a couple of examples to illustrate all of these tests and errors

EXAMPLES:

1) Determine if the sum of a(n) to infinity converges or not. If so, does the sum of |a(n)| to infinity converge?

a(n)= (-1)^(n-1) tan(1/n)/n (starting at n=1. NOTE that since we are summing to oo, it doesn't actually matter WHERE we start as long as n is defined-which means we don't start this one at n=0!)

Since the series is an alternating series, we try the AST immediately. Then we look at |a(n)| (you will soon learn this is called the Absolute Convergence of the series)

In [2]:
```
matplotlib notebook
```

In [3]:
```
n=sp.symbols('n',integer=True) #NOTE that we can assume n to be an integer-and also
a=(-1)**(n-1)*sp.tan(1/n)/n
# Alternating Series Test (NOTE you do the same process for Test for Divergence, ju
L=sp.limit(sp.Abs(a),n,sp.oo)
print('The limit is',L,'so the Alternating series converges.')
```

```
The limit is 0 so the Alternating series converges.
```

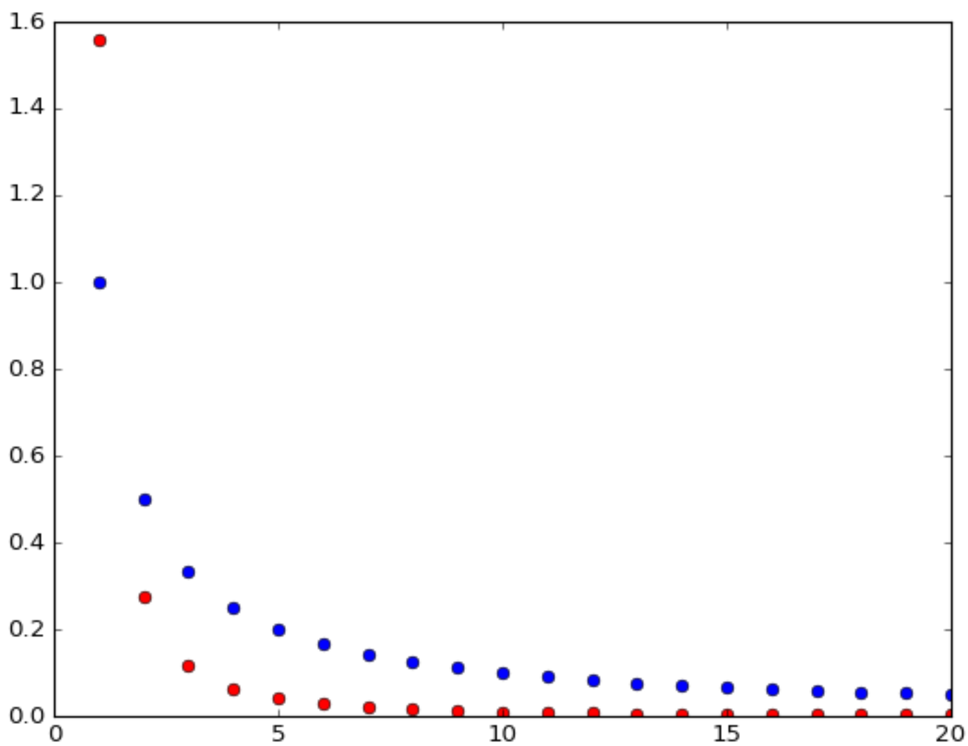
However, we cannot tell anything about the convergence of the sum of |a(n)| yet, since this is also the Test for Divergence for it and is therefore inconclusive. So we try the Integral Test on the series |a(n)|:

In [4]:
```
# Integral Test
x=sp.symbols('x')  #NOTE that since n is defined as an integer, we need to use the
f=sp.tan(1/x)/x
F=sp.integrate(f,(x,1,sp.oo)) #checking this first to see if it can be integrated
print('Improper integral is',F)
```

```
Improper integral is Integral(tan(1/x)/x, (x, 1, oo))
```

No luck (Python cannot integrate it). In general, oo means the series diverges, a finite # means the series converges. So we try the Comparison Test. Since there is an 'n' in the denominator, we try comparing with the series 1/n which diverges:

In [5]:
```
b=1/n
# To compare, we plot the sequences, which means using the matplotlib.pyplot.plot c
import matplotlib.pyplot as plt
nvals=range(1,21)
a1_20=[sp.Abs(a.subs({n:i})) for i in nvals]
b1_20=[b.subs({n:i}) for i in nvals]
plt.plot(nvals,a1_20,'ro',nvals,b1_20,'bo')
```



Out[5]:
```
[<matplotlib.lines.Line2D at 0x7fbd69941518>,
 <matplotlib.lines.Line2D at 0x7fbd5d93f080>]
```

Clearly from the plot, b(n) (in blue) is bigger, but since the b(n) series diverges, this tells us nothing. So we try the Limit Comparison Test with 1/n:

In [6]:
```
L=sp.limit(sp.Abs(a)/b,n,sp.oo)
print('If b(n)=1/n, the limit of |a(n)|/b(n) is',L,'which tells us nothing except t
```

```
If b(n)=1/n, the limit of |a(n)|/b(n) is 0 which tells us nothing except that b(n)
is bigger.
```

Time to get creative here: try a limit comparison with the smaller series, 1/n^2 !!!

(You may recall that tan(1/n)/(1/n) approaches 1 as "1/n" approaches 0, or n approaches oo)

In [7]:
```
b=1/n**2
L=sp.limit(sp.Abs(a)/b,n,sp.oo)
print('If b(n)=1/n^2, the limit of |a(n)|/b(n) is',L,'which tells us both series do
print('Since the series 1/n^2 converges, so does |a(n)|.')
```

```
If b(n)=1/n^2, the limit of |a(n)|/b(n) is 1 which tells us both series do the same
thing.
Since the series 1/n^2 converges, so does |a(n)|.
```

Since the alternating series converges, we can figure out how many terms are needed to sum to within .001.

Recall that for an alternating series, |S - S(N)| < |a(N+1)|. So if we can make the larger side (the right-hand side) < .001, we automatically get the smaller side (the left-hand side) < .001. We will try to **solve** the right-hand side = .001.

In [8]:
```
N=sp.symbols('N',positive=True)
ASTerror=sp.Abs(a).subs(n,N+1)
Nmin=sp.solve(ASTerror-.001,N)
print('minimum number of N is',Nmin)
```

```
---------------------------------------------------------------------------
NotImplementedError                       Traceback (most recent call last)
<ipython-input-8-def6d40ed2b0> in <module>
      1 N=symbols('N',positive=True)
      2 ASTerror=abs(a).subs(n,N+1)
----> 3 Nmin=solve(ASTerror-.001,N)
      4 print('minimum number of N is',Nmin)

/usr/lib/python3/dist-packages/sympy/solvers/solvers.py in solve(f, *symbols, **fla
gs)
    907     ##########################################################################
####
    908     if bare_f:
--> 909         solution = _solve(f[0], *symbols, **flags)
    910     else:
    911         solution = _solve_system(f, symbols, **flags)

/usr/lib/python3/dist-packages/sympy/solvers/solvers.py in _solve(f, *symbols, **fl
ags)
   1167         result = set()
   1168         for n, (expr, cond) in enumerate(f.args):
-> 1169             candidates = _solve(expr, *symbols, **flags)
   1170 
   1171             for candidate in candidates:

/usr/lib/python3/dist-packages/sympy/solvers/solvers.py in _solve(f, *symbols, **fl
ags)
   1412     if result is False:
   1413         raise NotImplementedError(msg +
-> 1414         "\nNo algorithms are implemented to solve equation %s" % f)
   1415 
   1416     if flags.get('simplify', True):

NotImplementedError: multiple generators [N, tan(1/(N + 1))]
No algorithms are implemented to solve equation -1/1000 + tan(1/(N + 1))/(N + 1)
```

Notice we get an error saying Python cannot solve this symbolically. We can use the sympy command "nsolve", which also requires a starting guess. We can likely try any positive value as a starting guess.

In [9]:
```
Nmin=sp.nsolve(ASTerror-.001,10)
print('minimum N is',Nmin)
print('Therefore, we need at least 31 terms to approximate to within .001')
```

```
minimum N is 30.6280469765407
Therefore, we need at least 31 terms to approximate to within .001
```

Let's find out what that approximation is and see if Python can sum the infinite series to compare it.

In [10]:
```
a1_31=[a.subs({n:i}) for i in range(1,32)]
S31=sum(a1_31).evalf()
print('The approximation S(31) is',S31)
```

```
The approximation S(31) is 1.36056951284712
```

Unfortunately, the "summation" command learned in the previou lab does not work.

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 [11]:
```
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: (-1)**(x-1)*tan(1/x)/x,[1,inf])
print('The actual sum of the series is about',S)
print('The difference between S and S(31) is',abs(S-S31))
```

```
The actual sum of the series is about 1.36006581874219
The difference between S and S(31) is 0.000503694104933006
```

EXAMPLE 2:

Clearly, since the series in Example 1 converges, so does the series a(n)=tan(1/n)/n^2. This series CAN be integrated, so determine how many terms are needed to converge to with .001.

We use the same programming technique as above, but instead of solving |S-S(N)| = a(N+1), we **solve** |S-S(N)| = integral(f(x),x=N..oo) (NOTICE the right hand side requires the **integrate** command)

In [12]:
```
n,N=sp.symbols('n N',positive=True)
a=sp.tan(1/n)/n**2
x=sp.symbols('x') # Integral Test, so need real-valued variable x again!
f=sp.tan(1/x)/x**2
Interror=sp.integrate(f,(x,N,oo))
print('The Remainder for the Integral Test is',Interror)
Nmin=sp.solve(Interror-.001,N)
print('minimum number of N is',Nmin,'so 23 terms are needed.')
# Notice we WERE able to solve this symbolically since there is nothing in the Inte

# Again, let's find out what that approximation is and see if Python can sum the in
a1_23=[a.subs({n:i}) for i in range(1,24)]
S23=sum(a1_23).evalf()
print('The approximation S(23) is',S23)

S=nsum(lambda x: sp.tan(1/x)/x**2,[1,inf])
print('The actual sum of the series is about',S)
print('The difference between S and S(23) is',sp.Abs(S-S23))
```

```
The Remainder for the Integral Test is log(tan(1/N)**2 + 1)/2
minimum number of N is [22.3644069897262] so 23 terms are needed.
The approximation S(23) is 1.77210227292052
The actual sum of the series is about 1.77300752350479
The difference between S and S(23) is 0.000905250584271489
```

In [ ]: