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

TO INFINITY AND BEYOND!

In class, you are currently looking at "infinite things":

1) Integration over an infinite domain

2) Sequences-an infinitely long list of terms (which lay the foundation for methods of numerical approximation in the rest of the chapter!)

In Python, you can use infinity as a bound: it is represented by oo (just think, "oo(h)...infinity"). You can still model the strategy used to calculate these improper integrals by hand.

Example: Use the method discussed in class to compute the integral of 3/(x^3 + 4x^2+5x) from x=1 to x=infinity. Verify this by integrating directly in Python.

As always, before typing anything in the computer, ask yourself: 1) "What are the steps to do this by hand?", and 2) "What Python command(s) allow me to do this?"

Step 1: Integrate the function from 1 to N (or whatever variable you used in class. Key Python command: **integrate**).

Step 2: Take the limit as N approaches infinity (Key Python command: **limit**)

In [3]:
```
x=symbols('x')
f=3/(x**3+4*x**2+5*x)
# Step 1: if you want to double-check your partial fraction work by hand, t
he Python command is apart
fparfrac=apart(f,x)
print('The partial fraction decomposition is',fparfrac)
# Now integrate this from 1 to N
N=symbols('N')
defint1=integrate(fparfrac,(x,1,N))
print('The integral from 1 to N is',defint1)
# OR just integrate the original function from 1 to N directly in Python
defint2=integrate(f,(x,1,N))
print('The integral from 1 to N is',defint2)
# Step 2: Take the limit as N approaches oo
lim=limit(defint1,N,oo)
print('The limit is',lim,'so the improper integral converges.')

# Of course, all of this was mainly to check our work by hand. It's much fa
ster to just integrate directly in Python:
defint=integrate(f,(x,1,oo))
print('The integral directly in Python is',defint,'so the integral converge
s.')
```

```
The partial fraction decomposition is -3*(x + 4)/(5*(x**2 + 4*x + 5)) + 3/
(5*x)
The integral from 1 to N is 3*log(N)/5 - 3*log(N**2 + 4*N + 5)/10 - 6*atan
(N + 2)/5 + 3*log(10)/10 + 6*atan(3)/5
The integral from 1 to N is 3*log(N)/5 - 3*log(N**2 + 4*N + 5)/10 - 6*atan
(N + 2)/5 + 3*log(10)/10 + 6*atan(3)/5
The limit is -3*pi/5 + 3*log(10)/10 + 6*atan(3)/5 so the improper integral
converges.
The integral directly in Python is -3*pi/5 + 3*log(10)/10 + 6*atan(3)/5 so
the integral converges.
```

NOTE: Can you figure out where the "pi" comes from? You'll find out in the next example!

Sequences can be thought of as functions whose domain is the set of nonnegative integers (or a subset thereof). So while symbolic manipulations can be done on them, plotting (so we can see what is actually going on!) requires a different plot command found in the matplotlib.pyplot library. Since the command is also "plot", we have to distinguish it from the symbolic plot command. One way is to call the full library path, i.e.,

matplotlib.pyplot.plot(...)

As you can see, this can get rather cumbersome. So, in typical mathematician fashion, we create a shorthand notation for it.

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

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

Example: Given the sequence a_n = arctan(n^3/(n+1)):

a) List the first ten terms of the sequence (a_1 to a_10). Give decimal approximations.

b) Plot the first 50 terms of the sequence.

c) Find the limit of the sequence or show it diverges.

For part a), we need to use list comprehension for the terms. NOTE that instead of defining the expression separately, we could just define

a1_10=[atan(n**3/(n+1)) for i in range(1,11)]

In [5]:
```
n=symbols('n',integer=True) #NOTE that we can assume n to be an integer-and
also positive if necessary
a=atan(n**3/(n+1))
a1_10=[a.subs(n,i) for i in range(1,11)]
# Notice the "range" command above. It creates a list of integers from 1 IN
CLUSIVE to 11 EXCLUSIVE
print('The list of terms is',a1_10)
print('As decimal approximations:',a1_10.evalf())
```

```
The list of terms is [atan(1/2), atan(8/3), atan(27/4), atan(64/5), atan(12
5/6), atan(216/7), atan(343/8), atan(512/9), atan(729/10), atan(1000/11)]

---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-5-18884c66eb11> in <module>
      4 # Notice the "range" command above. It creates a list of integers f
rom 1 INCLUSIVE to 11 EXCLUSIVE
      5 print('The list of terms is',a1_10)
----> 6 print('As decimal approximations:',a1_10.evalf())

AttributeError: 'list' object has no attribute 'evalf'
```

As you can see, we cannot just **.evalf()** the list. We have to do it IN our list comprehension (or create a separate list comprehension to do it):

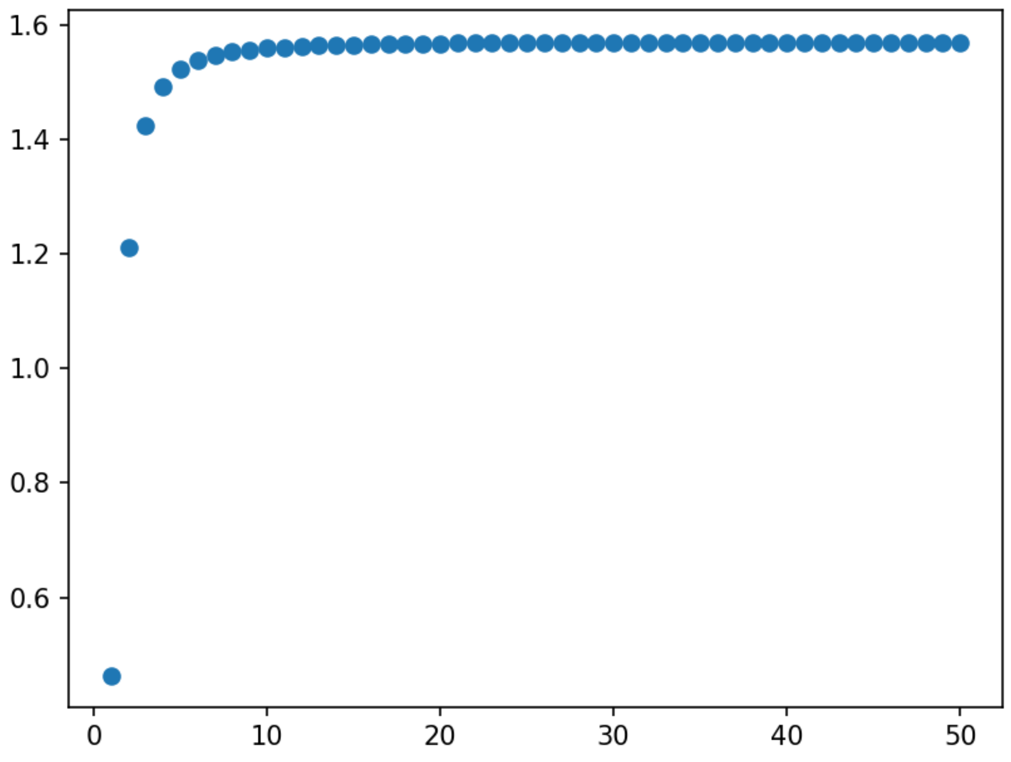
In [7]:
```
# Convert to floating point in a separate list comprehension
a1_10float=[i.evalf() for i in a1_10]
print('As decimal approximations:',a1_10float)

# Or do it all in one list comprehension
a1_10=[a.subs(n,i).evalf() for i in range(1,11)]
print('All at once:',a1_10)
```

```
As decimal approximations: [0.463647609000806, 1.21202565652432, 1.42371797
140649, 1.49282969296335, 1.52283313991782, 1.53840025742952, 1.54747693953
377, 1.55322001195040, 1.55707976596454, 1.55979677042936]
All at once: [0.463647609000806, 1.21202565652432, 1.42371797140649, 1.4928
2969296335, 1.52283313991782, 1.53840025742952, 1.54747693953377, 1.5532200
1195040, 1.55707976596454, 1.55979677042936]
```

Even though the terms inside the arctangent function are clearly getting larger (looking at the exact list), the numbers appear to be getting closer to something...perhaps 1.56? So we look at the graph of the first 50 terms to check. Remember to use **plt.plot** to plot lists of values. Also note the 'o' option to plot points rather than connecting them.

In [8]:
```
matplotlib notebook
```

```
In [9]: # Part b
        nvals=range(1,51) #again note the exclusive right endpoint!
        a1_50=[a.subs(n,i) for i in nvals]
        plt.plot(nvals,a1_50,'o')
```



```
Out[9]: [<matplotlib.lines.Line2D at 0x5281530>]
```

Clearly, the terms are approaching something below 1.6! Now we find the exact limit (again key Python command: **limit**)

```
In [10]: L=limit(a,n,oo)
         print('The limit of the sequence is',L,'or approximately',L.evalf())

         The limit of the sequence is pi/2 or approximately 1.57079632679490
```

And if you don't know, now you know-the "pi" in the first example came from the limit of atan(N+2) as N approaches infinity. Informally, we can say "arctan(oo) = pi/2"!

One more example that will be useful on this lab:

EXAMPLE 3 (RECURSIVE SEQUENCES):

Given a(0)=11, define the sequence a recursively by

a(n) = |a(n-1)-1| if n is odd

a(n) = a(n-1)/2 if n is even

a) List the first 30 terms of the sequence. What appears to be happening?
b) Find (by hand) a recursive formula for the odd-numbered terms and a recursive formula for the even-numbered terms. Assuming the limit exists (a(n+1) --> L and a(n) --> L), find it.

We use a for loop to define the sequence (this also uses the **for** command, but there are multiple steps to repeat instead of just one in list comprehension). The easiest strategy is to build this sequence term by term, as described in the steps below:

1) define a as a list of one number

2) determine the next number in the sequence, which depends on whether n is even or odd (if/else statements). The easiest way to determine if a number is even or odd is to divide it by 2 and check the remainder. In Python, the % refers to the remainder when dividing.

3) append each new value to the list

In [2]:
```
# Step 1
a=[11]
for n in range(1,31):
    if n % 2:  # true=1 which means n is odd
        a.append(abs(a[n-1]-1))
    else:  # false=0 which means n is even
        a.append(a[n-1]/2)
print('The sequence of terms is',a) # Ran the code here to see what happene
d
print('It appears that they eventually bounce between 2/3 (odd) and 1/3 (ev
en).')
L=symbols('L',positive=True)
print('For even terms, a(2n+2)=|a(2n)-1|/2.')
# Replace both with L to calculate the limit
print('The limit is',solve(L-abs((L-1))/2,L))
print('For odd terms, a(2n+1)=|a(2n-1)/2 - 1|.')
# Replace both with L
print('The limit is',solve(L-abs(L/2-1),L))
```

```
The sequence of terms is [11, 10, 5.0, 4.0, 2.0, 1.0, 0.5, 0.5, 0.25, 0.75,
0.375, 0.625, 0.3125, 0.6875, 0.34375, 0.65625, 0.328125, 0.671875, 0.33593
75, 0.6640625, 0.33203125, 0.66796875, 0.333984375, 0.666015625, 0.33300781
25, 0.6669921875, 0.33349609375, 0.66650390625, 0.333251953125, 0.666748046
875, 0.3333740234375]
It appears that they eventually bounce between 2/3 (odd) and 1/3 (even).
For even terms, a(2n+2)=|a(2n)-1|/2.
The limit is [1/3]
For odd terms, a(2n+1)=|a(2n-1)/2 - 1|.
The limit is [2/3]
```

In [ ]: