Project 1

Stefan Popov

Part A

For the first part of this project, we are to write a C++ function that would implement the Nested Multiplication algorithm. The function would be written in a separate source file and it would take a Matrix object, containing the coefficients in front of the n-degree polynomial that we wish to evaluate, and a double that would contain the variable we wish to evaluate the n-degree polynomial for. The value would be calculated using the pseudocode algorithm provided in the description and later be returned in the function.

In a file called proj1_a.cpp we were to write a main method that would use the function and the Matrix class provided to us to create a Linspace from [-1,1] and evaluate the Taylor Polynomials of degree 4, 8 and 12 for

$$f(x) = e^x = \sum(\frac{x^n}{n!})$$

for every point in the Linspace. We would then compute

$$f(x) = e^x$$

for every point in the Linspace and evaluate the error term for each of the above mentioned polynomials. The resulting vectors are then to be saved and plotted.

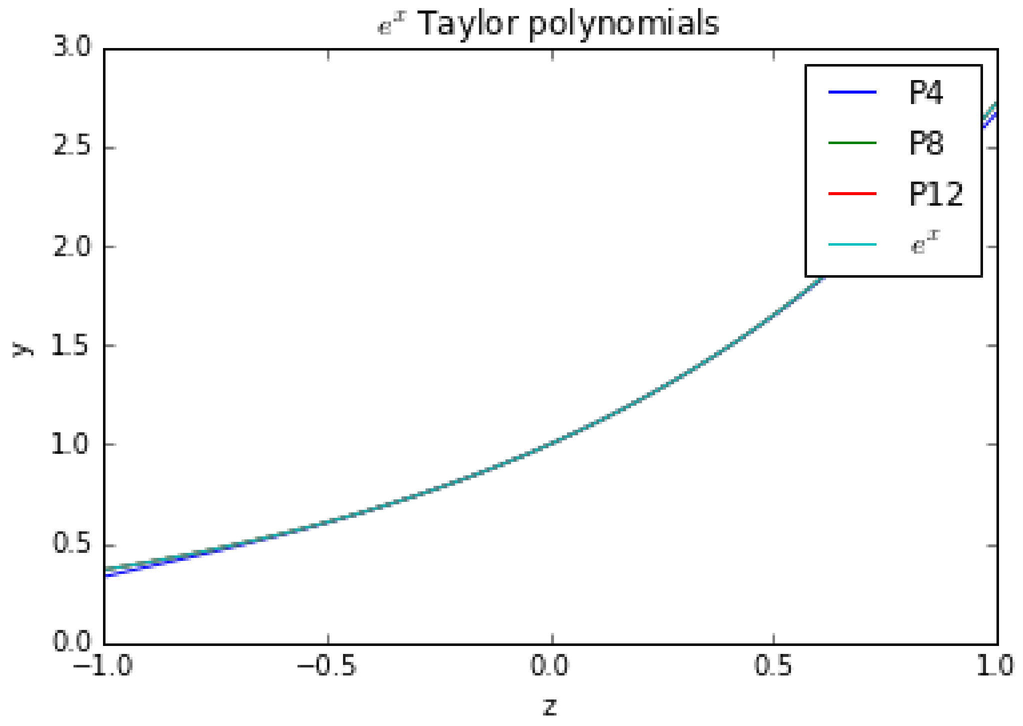
In [8]:
```
%pylab inline
z= loadtxt("z.txt")
f= loadtxt("f.txt")
p4 = loadtxt("p4.txt")
p8 = loadtxt("p8.txt")
p12 = loadtxt("p12.txt")
err4 = loadtxt("err4.txt")
err8 = loadtxt("err8.txt")
err12 = loadtxt("err12.txt")
```

```
Populating the interactive namespace from numpy and matplotlib
```

```
WARNING: pylab import has clobbered these variables: ['f']
`%matplotlib` prevents importing * from pylab and numpy
```

In [9]:
```
plot(z, p4, label= 'P4')
plot(z, p8, label= 'P8')
plot(z, p12, label='P12')
plot(z, f, label = '$e^x$')
xlabel('z')
ylabel('y')
title('$e^x$ Taylor polynomials')
legend()
```

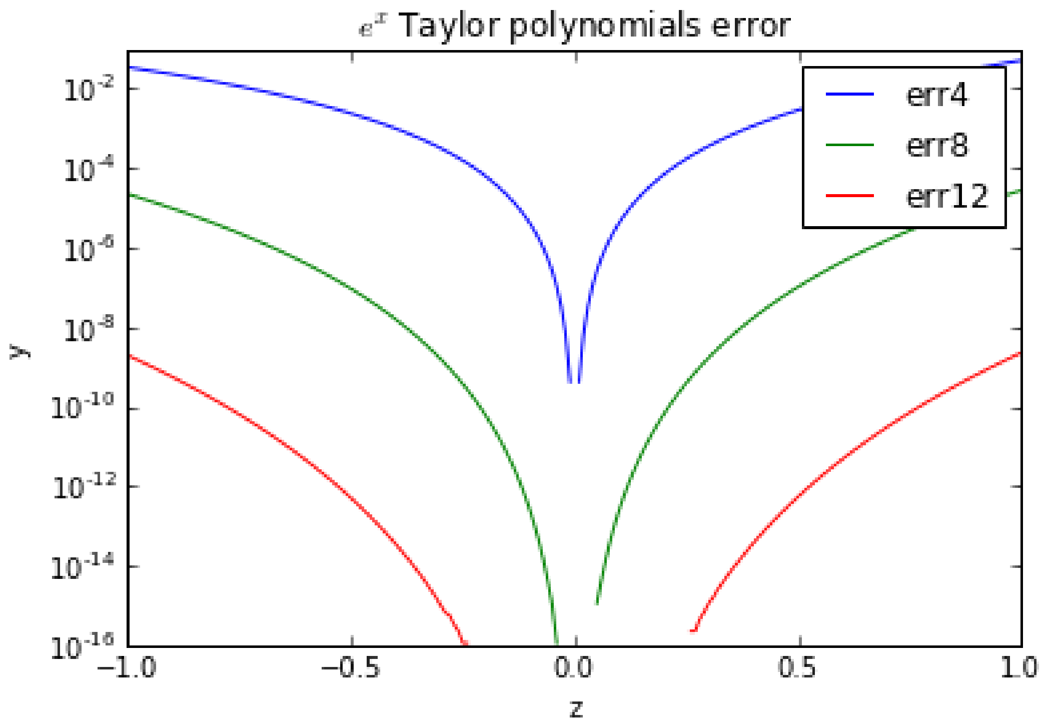
Out[9]:
```
<matplotlib.legend.Legend at 0x7f8ecbd14390>
```



Based on the above results, we can see that P12 is the best approximation to $f(x) = e^x$, which is logical, since P12 uses the most Taylor Polynomial terms, therefore should be the closest approximation out of the above used.

In [10]:
```
semilogy(z, err4, label= 'err4')
semilogy(z, err8, label= 'err8')
semilogy(z, err12, label='err12')
xlabel('z')
ylabel('y')
title('$e^x$ Taylor polynomials error')
legend()
```

Out[10]:
```
<matplotlib.legend.Legend at 0x7f8ecbec3a20>
```



These approximations were consistent with the upper bounds gotten from the derivations - $E_4 = 10^{-1}$, $E_8 = 10^{-4}$ and $E_{12} = 10^{-8}$, meaning the approximations are correct and confirmed.