Program to study the Chebyshev Approximation of an exponential

September 8, 2011

The function we will approximate is

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

in the region $-1 < x < 1$. We sample the function at N points and then study truncation to fewer points. The basic functions are defined below:

1a ⟨*chebcoefs* 1a⟩≡

```
def chebcoef(a,b,func,n):
  bma = (b-a)/2
  bpa = (a+b)/2
  c = zeros((n,1))
  pikbyn = pi*linspace(0.5,n-0.5,n)/n
  y = cos(pikbyn)
  fn = func(y*bma + bpa)
  fac = 2.0/n
  for j in range(n):
    c[j] = sum(fn * cos(j*pikbyn))*fac
  return c
```

`Chebev` is the NR version of the function, that uses the Clenshaw algorithm to compute the coefficients.

1b ⟨*chebev* 1b⟩≡

```
def chebev(x,c,a,b,n0):
  xx=x[where((x-a)*(x-b) <= 0)]
  z1 = (2*xx-a-b)/(b-a)
  z2 = 2*z1
  dd=zeros(z2.shape)
  d=zeros(z2.shape)
  for j in range(n0-1,0,-1):
    sv = d
    d = z2*d - dd + c[j]
    dd = sv
  y = z1*d - dd + 0.5*c[0]
  return y
```

`Direct` is the algorithm that computes the function directly.

2a ⟨*direct* 2a⟩≡
```
def direct(x,c,a,b,n0):
  xx=x[where((x-a)*(x-b) <= 0)]
  z1 = (2*xx-a-b)/(b-a)
  z2 = 2*z1
  T = xx;Told=ones(xx.shape);y=0.5*c[0]*Told;
  for j in range(1,n0):
    y=y+c[j]*T
    Tnew=z2*T-Told
    Told=T
    T=Tnew
  return y
```

We define our function

2b ⟨*function* 2b⟩≡
```
def f(x):
  y=exp(x)
  return y
```

Now we do the function approximation on 200 points and plot the coef magnitudes.

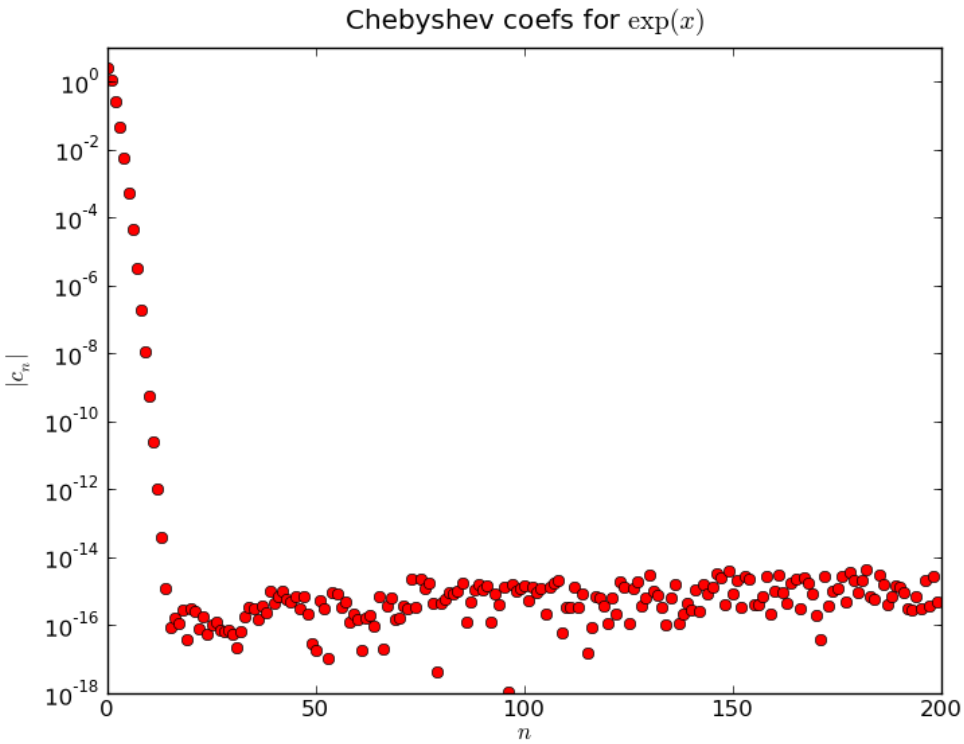
2c ⟨* 2c⟩≡
```
from scipy import *
from matplotlib.pyplot import *
⟨chebcoefs 1a⟩
⟨chebev 1b⟩
⟨direct 2a⟩
⟨function 2b⟩
N=200
c=chebcoef(-1,1,f,N)
figure(0)
semilogy(range(N),abs(c),"ro");
title(r"Chebyshev coefs for $\exp(x)$")
xlabel("$n$")
ylabel("$|c_n|$")
```

We now evaluate the function with 100 coefficients and plot the function as well as its error:

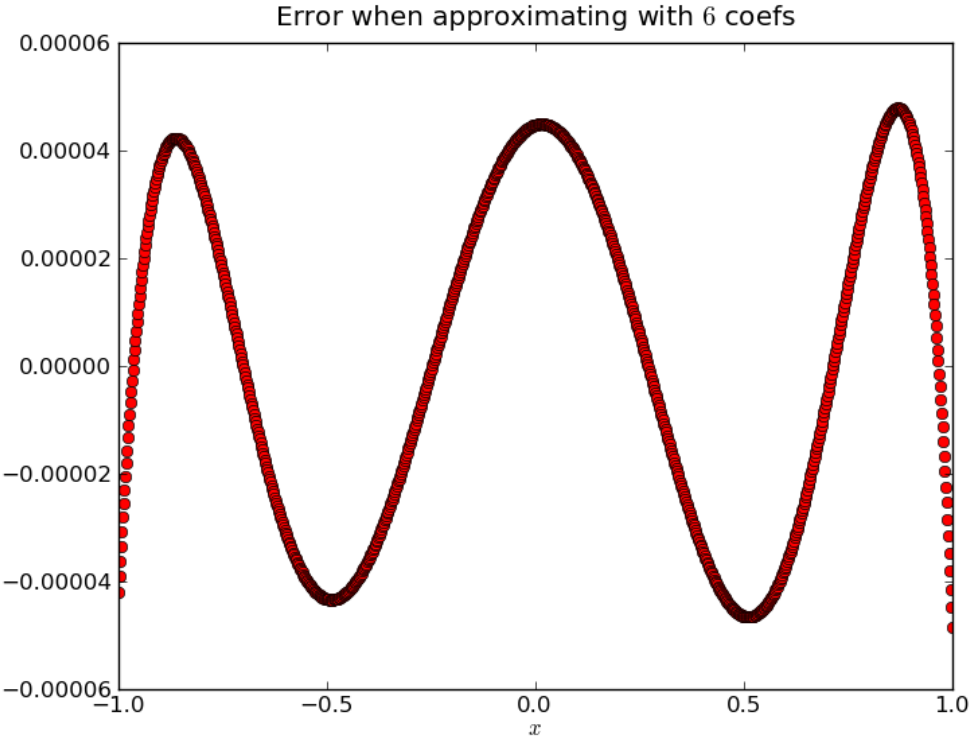
3 ⟨* 2c⟩+≡

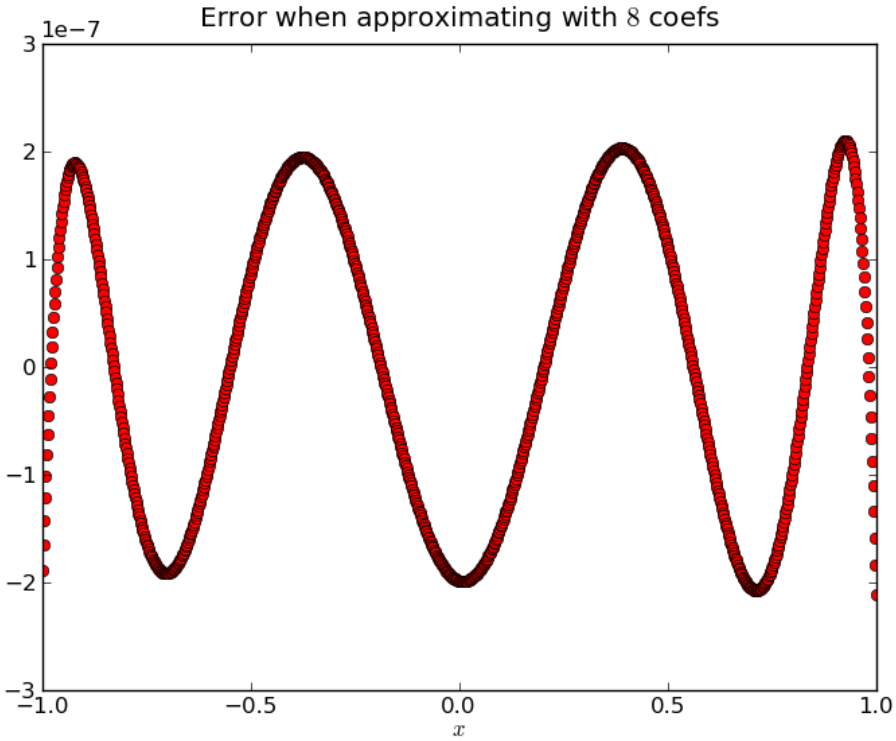
```
x=linspace(-1,1,1000)
y=chebev(x,c,-1,1,30)
yy=direct(x,c,-1,1,30)
figure(5)
plot(x,f(x),"k",x,y,"r",x,yy,"g");
legend([r"$\exp(x)$","clenshaw","direct"])
figure(6)
plot(x,f(x)-y,"k",x,f(x)-yy,"go")
legend(["clenshaw","direct"])
title("Error in fit vs x")
xlabel("$x$")
ylabel("Error")
print "Max difference between Clenshaw and Direct=%g\n" % abs(y-yy).max()
s=[]
for i in range(6,13,2):
  y=chebev(x,c,-1,1,i)
  figure((i-4)/2)
  plot(x,y-f(x),'ro')
  title(r"Error when approximating with $%d$ coefs" % i)
  xlabel("$x$")
```

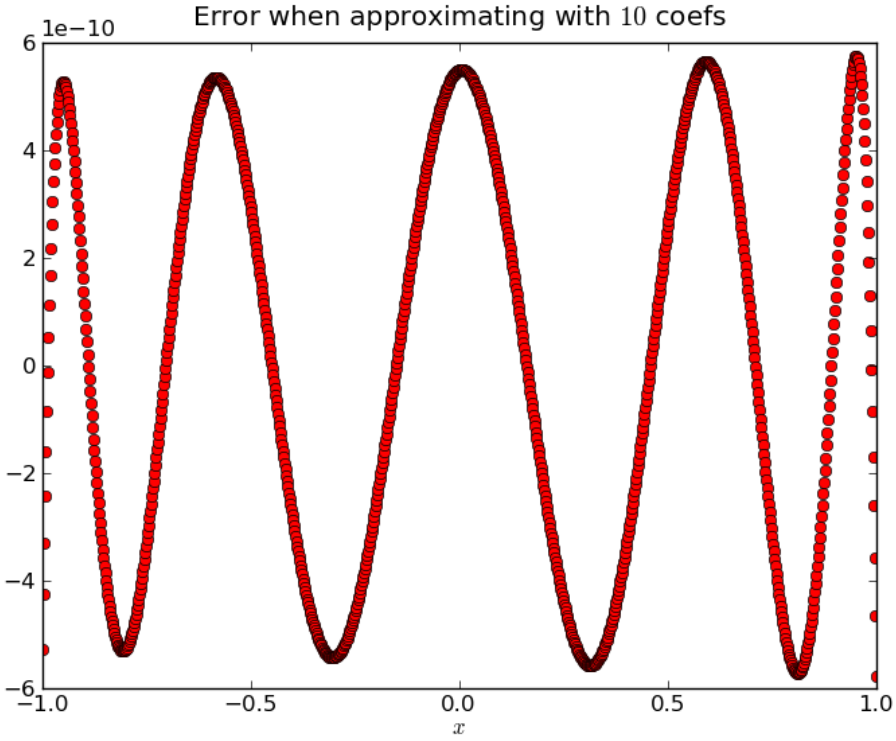
Here is what we get for the Chebyshev coefficients:

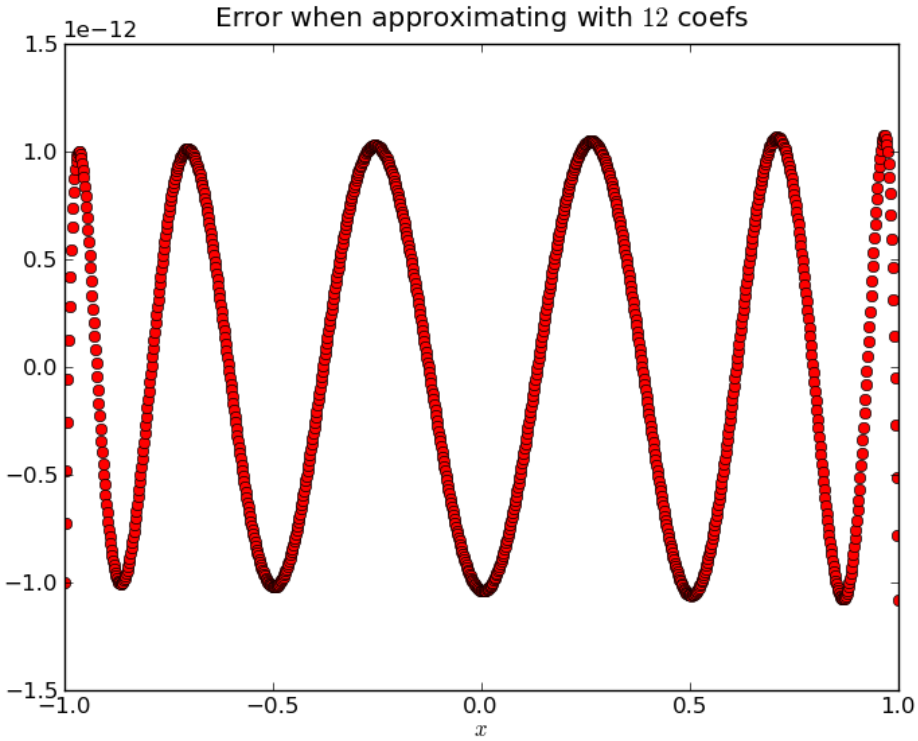


The rapid exponential decay of the coefficients is expected and seen. Just a few terms are enough to give us a 10 digit approximation of e^x over $(-1, 1)$. Here are the error profiles when we reconstruct with 6, 8, 10 and 12 coefficients.




Error when approximating with 8 coefs
1e−7
3
2
1
0
−1
−2
−3
−1.0
−0.5
0.0
0.5
1.0
x


Error when approximating with 10 coefs
1e−10
6
4
2
0
−2
−4
−6
−1.0
−0.5
0.0
0.5
1.0
x



As discussed in class the error is nearly uniformly distributed, which is why Chebyshev is so optimal. It gives us, with very little trouble, the best polynomial fit to the given order of any function.

Note that both the direct method and the Clenshaw method yield the same answer. The error in the 15^{th} digit is easily explained by round off and truncation errors. Thus both methods are almost equally accurate, though the plots do show slightly more error for the direct method.