EE5471: Interpolation

Harishankar Ramachandran

August 10, 2024

1 Introduction

In this assignment we will study interpolation. The codes are in C but we will call them from Python and graph the results.

2 Polynomial Interpolation

We will use the Fortran code in Numerical Recipes for interpolation. The code can be compiled and and linked to a main program in Fortran, but that is not our interest. We want to link this code to Python. To do this we use *f2py*. This is a package that is part of numpy, a package installed as a part of python.

f2py is also available as a seperate commandline command when you install numpy, and that is the way we will use it here. Given the fortran code polint.f (or polint.f90), we run the following command:

```
f2py -c -m polint polint.f
```

This compiles polint.f (which is what the -c triggers), and then creates python module polint (triggered by -m). The output of this command is a .so file:

```
polint                                            polint.f    testpolint.py polint.cp
```

polint.f and polint.f90 are source files. testpolint.py is a python file to test polint.f. polint is a sub directory that holds the compile files, usually removed after compilation. The library is the .so file. The name says polint has been converted by cpython for x86_64 archetecture on the gnu linux operating system.

If we look at testpolint.py we see how to use the library:

```
# script to test the polint module
from scipy import *
from matplotlib.pyplot import *
import polint as p
from numpy import sin,linspace,array # sin, linspace and
# array used to be part of scipy but have been shifted to
# numpy
# interpolate on a table of sin(x) as a test xx=linspace(0,2*pi,10) # creat
yy=sin(xx)
x=linspace(-pi,3*pi,501) # points at which to interpolate
y=[p.polint(xx,yy,w)[0] for w in x]
dy=[p.polint(xx,yy,w)[1] for w in x]
# plot the outputs
figure(1) plot(xx,yy,'ro',x,y,'r',x,sin(x),'g')
title("Interpolating sin(x)")
legend(["Table values","Interpolated values","True function"])
figure(2)
```

```
semilogy(x,abs(array(dy)),'r',x,abs(array(sin(x)-y)),'g') title("Estimated
legend(["Estimated Error","True Error"])
show()
#end if
```

In all the following problems, plot both the interpolated values and the actual values in a plot. Also show both the actual error vs the estimated error in another plot. Use semilog plots when showing error.

1. In python sample $\sin\left(x+x^2\right)$ from 0 to 1 at 5 points. Use these points as your table and do fourth order interpolation on these 200 points

   ```
   xx=linspace(-0.5,1.5,200)
   ```

   Since all the points in the table are used for 4^{th} order interpolation, this allows you to see what the effect of choosing a window that is not centred about the desired `xx` value.

2. Sample the same function at 30 points from 0 to 1. Now you have to choose the nearest 5 points and do fourth order interpolation. How does the accuracy change? What is the change due to?

3. With the same table of values, vary the order of interpolation. How does the error vary. Plot the error vs x for different orders in a semi log plot. Explain the curves you get.

4. Vary the interpolation order n from 3 to 20 and determine the way the maximum error varies with order.

5. We require a 6 digit accurate method to compute the function

   $$f(x) = \frac{\sin(\pi x)}{\sqrt{1-x^2}} \tag{1}$$

   between 0.1 and 0.9. The function is known *exactly* (to 15 digits) at certain locations, $x_k = x_0 + kdx$, $k = 0, \ldots, n$ where n is the order of interpolation required.

   (a) Convert the function to a table, spaced 0.05 apart, sampling it from 0.1 to 0.9.

   (b) In Python, plot this function and determine its general behaviour. Is it analytic in that region? What is the radius of convergence? What is the nature of the function's behaviour near ± 1?

   (c) Use the `polint` routine to interpolate the function at a thousand points between 0.1 and 0.9 for different orders. What order gives 6 digit accuracy? Explain the convergence in terms of the table spacing and the ROC.

2 $\langle * \; 2 \rangle \equiv$
$\langle$*code-old* (never defined)$\rangle$
$\langle$*code1* (never defined)$\rangle$