EE5471: Assignment on Spline Interpolation and Rational Interpolation

Harishankar Ramachandran

August 19, 2019

Linking C functions into python

I have taken and modified the spline function in numerical recipes and made it available as spline.c. But there are two problems now.

1. The main code takes a table of data and then computes y'' values by calling another function, and then evaluates at new points by calling a third function. But weave does not allow us to add a function definition in the code snippet.
2. Not in the current assignment but later, we will want to call python functions from the C code. I do not know a way to do this via weave.

The first problem

The problem with numerical recipes code is that it uses nrutil.h. I have put a version of spline.c that does not require nrutil.h which you can use in weave.

Even so, use of spline is non-trivial. So here is an example code that uses spline.c and does interpolation. This can be downloaded as testspline.py

```
# script to test the spline routine
from scipy import *
from matplotlib.pyplot import *
from scipy import weave
# define support code
with open("spline.c","r") as f:
  scode=f.read()
h=logspace(-3,0,16)
# h=input("Enter h as a numpy vector: ")
N=(2.0*pi)/h
err=zeros(h.shape)
for i in range(len(h)):
  x=linspace(0,2*pi,N[i])
  y=sin(x)
  n=int(N[i])
  xx=linspace(0,2*pi,10*n+1)
  y2=zeros(x.size)
  u=zeros(x.size)
  yy=zeros(xx.size)
```

```
  code="""
  #include<math.h>
  int i;
  double xp;
  spline(x,y,n,cos(x[0]),cos(x[n-1]),y2,u);
  for(i=0;i<=10*n;i++){
    xp=xx[i];
    splint(x,y,y2,n,xp,yy+i);
  }
"""
  weave.inline(code,["x","y","n","y2","u","xx","yy"],\
    support_code=scode,extra_compile_args=["-g"],\
    compiler="gcc")

  if i==0:
    figure(0)
    plot(x,y,'ro')
    plot(xx,yy,'g')
    title("Interpolated values and data points for n=%d" % N[i])
  z=abs(yy-sin(xx))
  err[i]=z.max()
figure(1)
loglog(h,err)
title("Error vs. spacing")
show()
```

The nice thing about interpolation is that all the functions are evaluated in Python. It is the table that is passed to C and the interpolated values returned.

F2Py and cython do not have the second problem. They are capable of "call back", meaning that the C or Fortran program can call a Python function in turn. As mentioned earlier, F2Py is the fastest, followed by weave, followed by cython. In this case, my attempt with F2Py was about three times faster than Cython, though I spent far more time with Cython.

The Assignment

1. Study cython to understand how to interface Python to C. Look at

   http://cython.readthedocs.io/en/latest/src/tutorial/external.html

   for a short introduction to this. Try to create an interface to spline.c

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

$$f(x) = \frac{x^{1+J_0(x)}}{\sqrt{(1+100x^2)(1-x)}} = \frac{x^{1+J_0(x)}}{\sqrt{1-x+100x^2-100x^3}} \tag{1}$$

   between 0 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 at $x = 0$ and at $x = 0.9$?

(c) Use spline interpolation on the function in Eq. (1). Use $y'' = 0$ at the boundaries, and obtain the spline coefficients and hence interpolate as above. How does the error vary with the number of uniformly spaced spline points? How many points are required to achieve six digit accuracy? Use the C code above with weave.

Note: What you vary are the number of points in the table. You test by interpolating on many more points and generating an error profile. Plot the same for different cases on same plot. Find the maximum error in each case, and study the scaling (how does maximum error vary with spacing of points in your table?)

(d) Revise the C code to implement the *not-a-knot* boundary condition. Repeat Q3 and see how the error profiles and the maximum errors vary.

(e) Use Eq. 1 to *analytically* compute the derivitives at the end points. Note that $J_0'(x)$ can be written in terms of $J_1(x)$ and hence computed. Use that information to obtain the spline coefficients. How does the error in the interpolated value compare with Q3 and Q4? This is an important issue for an experimentalist since he usually does not know y' at x_0 and x_n, but can use either of Q3 or Q4 to generate his spline fit.

(f) Use hundred times the actual derivitive as your boundary condition. How does the spline fit change? Plot the points near the edge and show the way that the errors decay. Obtain a functional dependence of error vs distance from edge. This is what you need to explain in the theory assignment.

(g) Can a non-uniform spacing of spline points help reduce the number of spline points required? Try to determine the optimal spacing at which error appears to be uniform across the domain. **Develop a theory for the optimal spacing to use assuming error is similar to cubic interpolation.**