Contact (https://www.linkedin.com/profile/view?id=181655952)

Search

- Home (/)
- PyBlog (/pyblog.html)
- General Python (/general-python.html)
- Topic list (/topic-list.html)
- Code library (/code-library.html)
- Lecture notes (/lecture-notes.html)
- Contact (/contact.html)

Non-uniform first order finite differences (http://www.scientificpython.net/pyblog/non-uniform-first-order-finite-differences)

4/23/2012

3 Comments (http://www.scientificpython.net/pyblog/non-uniform-first-order-finite-differences#comments)

Suppose we take a set of $n+1$ ordered, but non-uniformly distributed grid points $\{x_0, x_1, \ldots, x_n\}$. If we have a function $f(x)$ that we can evaluate on these grid points. The function can be approximated as being piecewise parabolic. At a point x_k, approximate $f(x)$ with three quadratic Lagrange polynomials.

$$f(x) \approx f(x_{k-1})\ell_{k-1}(x) + f(x_k)\ell_k(x) f(x_{k+1})\ell_{k+1}(x)$$

Let $h_k = x_k - x_{k-1}$. We can write the three polynomials as

$$\ell_{k-1}(x) = \frac{x^2 - (x_{k+1} + x_k)x + x_k x_{k+1}}{h_k(h_k + h_{k+1})}$$

$$\ell_k(x) = -\frac{x^2 - (x_{k+1} + x_{k-1})x + x_{k-1}x_{k+1}}{h_k h_{k+1}}$$

$$\ell_{k+1}(x) = \frac{x^2 - (x_{k+1} + x_{k-1})x + x_k x_{k-1}}{(h_k + h_{k+1})h_{k+1}}$$

To numerically approximate the first derivative of f at the grid point x_k, we can differentiate the Lagrange interpolants

$$\ell'_{k-1}(x) = \frac{2x - (x_{k+1} + x_k)}{h_k(h_k + h_{k+1})}$$

$$\ell'_k(x) = -\frac{2x - (x_{k+1} + x_{k-1})}{h_k h_{k+1}}$$

$$\ell'_{k+1}(x) = \frac{2x - (x_{k+1} + x_{k-1})}{(h_k + h_{k+1})h_{k+1}}$$

For all interior grid points, we approximate $f'(x_k)$ as

$$f'(x_k) \approx a_k f(x_{k-1}) + b_k f(x_{k-1}) + c_k f(x_{k+1})$$

where the weights are obtained by evaluating the derivatives of the Lagrange polynomials at x_k

$$a_k = \ell'_{k-1}(x_k) = -\frac{h_{k+1}}{h_k(h_k + h_{k+1})}$$

$$b_k = \ell'_k(x_k) = \frac{h_{k+1} - h_k}{h_k h_{k+1}}$$

$$c_k = \ell'_{k+1}(x_k) = \frac{h_k}{h_{k+1}(h_k + h_{k+1})}$$

These formulas hold for $k = 1, \ldots, n-1$
We have to do something different for $k=0$ and $k=n$, so we use one-sided differences

$$f'(x_0) \approx a_0 f(x_0) + b_0 f(x_1) + c_0 f(x_2)$$

with

$$a_0 = -\frac{2h_1 + h_2}{h_1(h_1 + h_2)}$$

$$b_0 = \frac{h_1 + h_2}{h_1 h_2}$$

$$c_0 = -\frac{h_1}{h_2(h_1 + h_2)}$$

Similarly at the right end, we have

$$a_n = \frac{h_n}{h_{n-1}(h_{n-1} + h_n)} b_n = -\frac{h_{n-1} + h_n}{h_{n-1}h_n} c_n = \frac{2h_n + h_{n-1}}{h_n(h_{n-1} + h_n)}$$

To numerically differentiate f on the grid, we can apply an almost tridiagonal matrix $\mathbf{D}$ to the sampled function.

$$\mathbf{f}' = \mathbf{D}\mathbf{f}$$

Where the differentiation matrix is

$$\mathbf{D} = \begin{pmatrix} a_0 & b_0 & c_0 & & & \\ a_1 & b_1 & c_1 & & & \\ & a_2 & b_2 & c_2 & & \\ & & \ddots & \ddots & \ddots & \\ & & & a_{n-1} & b_{n-1} & c_{n-1} \\ & & & a_n & b_n & c_n \end{pmatrix}$$

Below is a Python code which, when given the vector $\mathbf{x}$, produces the matrix $\mathbf{D}$ as a sparse matrix.

```
import numpy as np
from scipy.sparse import csr_matrix

def nufd(x):
    n = len(x)
    h = x[1:]-x[:n-1]
    a0 = -(2*h[0]+h[1])/(h[0]*(h[0]+h[1]))
    ak = -h[1:]/(h[:n-2]*(h[:n-2]+h[1:]))
    an = h[-1]/(h[-2]*(h[-1]+h[-2]))
    b0 = (h[0]+h[1])/(h[0]*h[1])
    bk = (h[1:] - h[:n-2])/(h[:n-2]*h[1:])
    bn = -(h[-1]+h[-2])/(h[-1]*h[-2])
    c0 = -h[0]/(h[1]*(h[0]+h[1]))
    ck = h[:n-2]/(h[1:]*(h[:n-2]+h[1:]))
    cn = (2*h[-1]+h[-2])/(h[-1]*(h[-2]+h[-1]))
    val  = np.hstack((a0,ak,an,b0,bk,bn,c0,ck,cn))
    row = np.tile(np.arange(n),3)
    dex = np.hstack((0,np.arange(n-2),n-3))
    col = np.hstack((dex,dex+1,dex+2))
    D = coo_matrix((val,(row,col)),shape=(n,n))
    return D
```

/uploads/1/1/5/9/11598566/nufd.py)

nufd.py
Download File (/uploads/1/1/5/9/11598566/nufd.py)

Like 0 Tweet

3 Comments (http://www.scientificpython.net/pyblog/non-uniform-first-order-finite-differences#comments)

caduqued 6/7/2012 02:10:53 am

Really nice site! and the info is extremely clear! Thanks for sharing!

Reply

jess 12/2/2012 09:19:51 am

Should coo_matrix be csr_matrix? Thanks for the implementation though! Have bookmarked your site - it looks extremely interesting!

Reply

Ben 3/28/2013 12:51:03 am

Very helpful thanks: a small correction to the expression for the first derivative:
f′(xk)≈akf(xk−1)+ bkf(xk−1)+ ckf(xk+1)
should be
f′(xk)≈akf(xk−1)+ bkf(xk)+ ckf(xk+1)
And the expression for f(x) itself is missing a +.

Reply

Your comment will be posted after it is approved.

Leave a Reply.

Name (required)

Email (not published)

Website

Comments

Notify me of new comments to this post by email

Submit

RSS Feed (/1/feed)

View my profile on LinkedIn (http://www.linkedin.com/in/veltri)

Archives

Categories