Notes on the derivation of Finite Differences kernels, on regularly spaced grids, using aritrary sample points

Nicholas Maxwell
Departments of Physics and Mathematics
University of Houston
Houston, TX 77204-5006
namaxwell@uh.edu
nicholas.maxwell@gmail.edu

We present a method to solve for error-free kernels which may differentiate a locally analytic function on a regularly spaced grid, using arbitrary sample points for the differentiation kernel.

I. INTRODUCTION

The approach presented here is a generalization of that in [3], wherein a method for solving centered Finite Differencing kernels is presented, for the case of $k = 2$, the Cartesian Laplacian. The approach here is to generate kernels which sample different numbers of points to the left and to the right of a grid point, therefore allowing one to approximate a derivative at the edge of a grid, where there are no points on one side. The only assumption made about the function being differentiated is that it be locally analytic over the grid, so that analysis involving Taylor series is appropriate.

No claim is made as to the originality of this material; the Finite Differences approach is well established, [3] was the only reference used. The purpose of this material is to present a clear and simple method to achieve the above goals.

Given a function $f : \mathbb{R} \to \mathbb{R}$, we sample it at the points (x_i), $x_i = h\,i + x_0$, $h > 0$, so that we have the sequence of numbers (f_i), $f_i = f(x_i)$. The problem is to approximate $f^{(k)}$, the k^{th} derivative of f. To do this we compute the sequence $(f_i^{(k)})$, $f_i^{(k)} = f^{(k)}(x_i)$, where $f_i^{(k)}$ is given by

$$f_i^{(k)} = \frac{1}{h^k} \sum_{j\,=\,-L}^{R} f_{i+j}\,\delta_j. \tag{I.1}$$

We call the sequence of numbers (δ_j) a differentiation kernel, as the above structure is discrete convolution.

II. THEORY

We start with a function f which is analytic on the interval I. For $x, z \in I$, we have the Taylor series of f, evaluated around a fixed z,

$$T(x;\, f, z) = \sum_{n=0}^{\infty} \frac{(x-z)^n}{n!}\, f^{(n)}(z). \tag{II.1}$$

We have the sequences $(f_i^{(n)})$ of n^{th} derivatives of f, and (x_i) of numbers, where $f_i^{(n)} = f^{(n)}(x_i)$, and $x_{i+1} = x_i + h$, $x_i \in I, h > 0$. Then we write,

$$\tilde{f}_{i;j} = \sum_{n=0}^{N} \frac{(i-j)^n}{n!}\, h^n f_j^{(n)}, \tag{II.2}$$

so that $\tilde{f}_{i;j}$ is the truncated Taylor series of f evaluated at the point x_i, around the point x_j. Then, we form the row vector $B_j = (B_{j,n})$, $B_{j,n} = h^n f_j^{(n)}$. And the column vector $a_{i,j} = (a_{i,j,n})$, $a_{i,j,n} = \frac{(i-j)^n}{n!}$. Then we can write $\tilde{f}_{i;j}$ as the product of B_j with $a_{i,j}$,

$$\tilde{f}_{i;j} = B_j\; a_{i,j}. \tag{II.3}$$

Now we wish to take a linear combination of several $\tilde{f}_{i;j}$,

$$\sum_{i=j-L}^{j+R} \delta_i \, \tilde{f}_{i;j} = \sum_{i=j-L}^{j+R} \delta_i \, B_j \, a_{i,j} = B_j \sum_{i=j-L}^{j+R} \delta_i \, a_{i,j}, \tag{II.4}$$

where δ_i is the i^{th} component of a column vector $\delta = (\delta_i)$. This can be written in matrix form; let A_j be the matrix whose columns are $a_{i,j}$, $A_j = (a_{j-L,j}; a_{j-L+1,j}; ...; a_{j+R-1,j}, \; a_{j+R,j})$.

$$\sum_{i=j-L}^{j+R} \delta_i \, \tilde{f}_{i;j} = B_j \; A_j \; \delta \tag{II.5}$$

So δ takes a linear combination of the function points f_i, and the above expression gives the truncated Taylor series of that linear combination with respect to the point x_j, in a convenient matrix form. Of course, we wish this linear combination to be a good approximation of $f^{(k)}(x_j)$, so that δ is a differentiation kernel. We can write $h^k \, f^{(k)}(x_j)$ as $B_j \, b_k$, where b_k is a column vector whose entries are zero, except for the entry corresponding to $f^{(k)}$, which takes the value 1, so $b_k = (b_{k,n})$, $b_{k,n} = \delta_{n,k}$, where $\delta_{n,k}$ is the Kronecker-delta, not to be confused with the above differentiation kernel. We then have,

$$B_j \; A \; \delta = B_j \, b_k \; \rightarrow \; A_j \; \delta = \; b_k \; \rightarrow \; \delta = A_j^{-1} \, b_k, \tag{II.6}$$

so we can see that δ is the k^{th} column of the inverse of A_j. This linear system is solved using standard LUP factorization and back substitution.

We examine the form of A_j. We pick $N = 1 + L + R$, so that A_j is square. The rows of A_j are indexed by n, and the columns by i, n runs form 1 to $N = 1 + L + R$, and i runs from $j - L$ to $j + R$. So R is the number of points f_j sampled to the right of x_j, and L the number of points to the left, we have

$$A_{j,n,i} = \frac{(i-j)^n}{n!}. \tag{II.7}$$

III. SOLUTION AND APPLICATION

If we wish to evaluate the k^{th} derivative only, then we look at the Taylor expansions around x_0, as from where we choose to enumerate (x_i) is arbitrary, then $j = 0$, and we can drop the j index; we solve $A \, \delta = b_k$, where

$$A_{n,i} = \frac{i^n}{n!}. \tag{III.1}$$

Clearly, this matrix A is nonsingular, however, it is ill-conditioned, due primarily to the $\frac{1}{n!}$ term scaling each row. In order to derive accurate kernels, we need to solve moderately sized matrices, where N may be on the order of 20. Then the ratio of row 1 to row N will roughly be $\frac{1}{N!}$, the base-10 log of which is roughly, using Sterling's approximation, $-0.4 - 0.43((N + \frac{1}{2} \ln N - N))$.

So we need only try to solve an $N = 18$ sized matrix before the ratio of the first row to the last is less than machine precision on modern personal computers using double precision, and thus the last row will effectively be zero, and the matrix will be singular; we see severe numerical error long before that. Moreover, the goal being accuracy, we require the kernel to be free of any numerical error. However, we need only compute each kernel once, and store them in an efficiently accessed manner.

We can solve the system by taking advantage of template programming, featured by many modern programming languages. We can write a simple LUP solving algorithm, which is, until compile time, independent of the data type used for the underlying field of the matrix. Then we can use either an arbitrary precision decimal number class, or, seeing as the underlying field need only be rational, we can use a rational number class. The rational approach would yield exact solutions, the arbitrary precision approach would yield arbitrarily accurate results.

If the LUP algorithm indexes matrix elements starting at 1, then

$$A'_{i',j'} = A_{i'-1,j'-L-1} = \frac{(j'-L-1)^{i'-1}}{(i'-1)!}, \tag{III.2}$$

should be the correct form of the matrix to pass the LUP algorithm.

Kernels, for $k = 2$ were solved with different permutations of R, L up to 40, using the C++ programming language and the ARPREC number class [1], with a precision of 700 digits. Run time was on the order of an hour on a basic personal computer, which is good for a one time computation. The resulting kernels were stored in header files, for later inclusion at compile time. Only enough digits need be stored as the precision of the target data type requires, here this was 17 digits. For simplicity, the LUP algorithm used was that presented in [2], and adapted to make use of template programming. A good application of this approach is to apply the Laplacian operator to a discrete grid of function points; by letting $R = 0, L = 24$, one can approximate the solution quite well at the edges, avoiding many complications. Near the edge one may take $(R, L) = (1, 24)$, then $(2, 24)$, etc., thus giving a gradual change from one-sided differentiation to centered differentiation.

[1] ARPREC, http://crd.lbl.gov/ dhbailey/mpdist/

[2] Flannery, B.P., S.A. Teukolsky, W.T. Vetterling, "Numerical Recipes : the Art of Scientific Computing", Cambridge University Press, (2007).

[3] L. Ann, Q. H. Tran and W. W. Symes, Dispersion and cost analysis of some finite difference schemes in one-parameter acoustic wave modeling, Comput. Geosci. 1 (1997), 133