NUMERICAL DIFFERENTIATION IN MULTIPLE DIMENSIONS

via Polynomial Interpolation

Derek Li
Scientific Computing, Summer 2021

Abstract

Numerical differentiation by polynomial interpolation and the related methods of derivative approximation with finite differences are a foundational aspect of scientific computing. In this study, we present a method for deriving finite difference formulas of arbitrary orders of accuracy using polynomial interpolation. This is implemented as a symbolic routine in MATLAB. These formulas are classified by the number of interpolation points n, the order of differentiation k, the location where the derivative is evaluated, denoted the pivot, and the order of accuracy e, related as $n = k + e$. We assemble these finite difference expressions into a *centric* differentiation matrix that applies central formulas when possible and pseudo-central formulas near the boundaries, all of the same accuracy order.

This provides a commodious numerical differentiation module that, to the author's knowledge, has no built-in counterpart in MATLAB or Python.[1] This may be referred to as *directional differentiation*. As the order of accuracy describes asymptotic behavior, we show by an example that as different partial functions may converge at different rates, and that computer memory is finite, practically for multidimensional functions, the asympotitic convergence may not be observed or observable.

1. POLYNOMIAL INTERPOLATION

Numerical differentiation is instrumental when the function of interest is not known by definition, such as in physical experiments where discrete measurements are made, or when deriving the derivative analytically is difficult. A general approach is to interpolate the available function values with an *interpolant* functions, whose derivatives are then evaluated at desired points as estimates of the original. Polynomial interpolants are easily constructed and differentiated, with the *Interpolation Error Theorem* warranting error orders for differentiable functions. The expressions derived for the derivatives are known as *finite difference formulas*.

In this section, we pose the 1D problem, the solution to which translates directly to multiple dimensions, as treated later. To pose the problem, we assume function values are taken at a set of grid points uniformly spaced by h. To approximate the derivative at some grid point x_i, we utilize some n consecutive neighboring points, including itself, to interpolate a polynomial of $n - 1^{\text{th}}$ degree. We then evaluate the derivative of the polynomial at x_i.

[1] The Julia language has a package *FiniteDiff* which computes first and second derivatives with finite differences at various orders of accuracy.

The expression derived for the derivative at this point is independent of the specific abscissa value and applies, as a formula to all situations where we have the same relative selection of neighboring points. This is simply because for a shifted abscissa $x' = x + h$,

$$\frac{df}{dx'} = \frac{df}{dx}$$

at the corresponding abscissa values.

Thus our problem is formulated as interpolating a polynomial through n uniformly spaced abscissa, which we conveniently choose to be

$$\vec{x} = \begin{pmatrix} 0 & h & 2h & \dots & (n-1)h \end{pmatrix} \tag{1}$$

With corresponding ordinate values of function $f(x)$ as

$$\vec{y} = \begin{pmatrix} y_0 & y_1 & y_2 & \dots & y_{n-1} \end{pmatrix}. \tag{2}$$

Evaluating the derivative of the polynomial at different abscissa locations, the location of evaluation called the *pivot*, yields different finite difference formulas that use different neighboring ordinate values relative to the pivot, which are nonetheless of the same order of accuracy, as they originate from the same underlying polynomial.

$$P(x) = a_0 + a_1 x + a_2 x^2 ... a_{n-1} x^{n-1} \equiv \vec{a} \cdot \vec{x}^p \tag{3}$$

The interpolant polynomial is of degree $n-1$, whose coefficients are determined from the fitted points. This constitutes a linear system of equations, where a row of the matrix corresponds to a distinct abscissa and the columns contain consecutive powers of different abscissas. This matrix is known as the *Vandermonde* matrix.

$$V\vec{x} \equiv \begin{bmatrix} 1 & x_0 & x_0^2 & \dots & x_0^{n-1} \\ 1 & x_1 & x_1^2 & \dots & x_1^{n-1} \\ \vdots & \vdots & \dots & \vdots & \\ 1 & x_{n-1} & x_{n-1}^2 & \dots & x_{n-1}^{n-1} \end{bmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix} \tag{4}$$

$$= \begin{bmatrix} 1 & h & h^2 & \dots & h^{n-1} \\ 1 & 2h & (2h)^2 & \dots & (2h)^{n-1} \\ \vdots & \vdots & \dots & \vdots & \\ 1 & (n-1)h & [(n-1)h]^2 & \dots & [(n-1)h]^{n-1} \end{bmatrix} \begin{pmatrix} a_0 \\ a_1 \\ \vdots \\ a_{n-1} \end{pmatrix} \tag{5}$$

$$= \vec{y} \tag{6}$$

For distinct abscissa values, the Vandermonde matrix is non-singular, so $\vec{a}$ is uniquely defined by the abscissa and ordinate values. This is proved by assuming the fundamental theorem of algebra[2], which states that a single variate polynomial of degree $n \in \mathbb{N}$ has exactly n roots, which may be repeated, real or complex. Suppose to the contrary that given pairwise distinct

[2]Proof idea from Martin Sleziak [1].

values $x_0, x_1, \ldots, x_{n-1}$, the Vandermonde matrix is singular, that is $\exists\, \vec{b} = (b_0, b_1, \ldots, b_{n-1})$ such that

$$b_0 \begin{pmatrix} x_0^0 \\ x_1^0 \\ \vdots \\ x_{n-1}^0 \end{pmatrix} + b_1 \begin{pmatrix} x_0 \\ x_1 \\ \vdots \\ x_{n-1} \end{pmatrix} + \cdots + b_{n-1} \begin{pmatrix} x_0^{n-1} \\ x_1^{n-1} \\ \vdots \\ x_{n-1}^{n-1} \end{pmatrix} = \mathbf{0}.$$

The abscissa x_i from each of the n rows in this linear system is then a root of the $n-1^{\text{th}}$ degree polynomial

$$P(x) = b_0 + b_1 x + b_2 x^2 + \cdots + b_{n-1} x^{n-1}$$

Which is a contradiction with the fundamental theorem unless P is the trivial polynomial, that is 0. This assures us that $\vec{a}$ can be determined by elementary operations.

The generalized linear system (4) allows for deriving polynomial interpolants, thereby finite difference formulas, for customized non-uniformly spaced points, useful in various situations. In this study, we proceed with uniformly sampled points which possess a form amenable to unified treatment.

The derivation of the coefficients can be executed by a program capable of performing arithmetical operations symbolically. The MATLAB *Symbolic Toolbox* is employed to solve the above system of linear equations symbolically in h and $\vec{y}$.

2. POLYNOMIAL DIFFERENTIATION

The solution to (5), which we've shown is well-defined, gives the formula of the polynomial. We now need only to differentiate the polynomial to an arbitrary order and evaluate the derivative at different pivot positions. This task can be handled by symbolic computing with the ability of differentiating polynomials and substitution. For example, using three points, we obtain the following polynomial.

$$P(x) = y_0 - \frac{3y_0 - 4y_1 + y_2}{2h} x + \frac{y_0 - 2y_1 + y_2}{2h^2} x^2$$

Symbolically differentiating this expression and evaluating the derivative at $x = 0, h, 2h$ yields the following finite difference formulas, the second of which the well-used simple central difference.

$$\begin{aligned} \hat{f}'(0) &= \frac{-3y_0 + 4y_1 - y_2}{2h} \\ \hat{f}'(h) &= \frac{-y_0 + y_2}{2h} \\ \hat{f}'(2h) &= \frac{y_0 - 4y_1 + 3y_2}{2h} \end{aligned}$$

Here $\hat{f}'$ is the estimate of the derivative. This is conveniently organized into a matrix

$$\hat{f}' = h^{-1} \begin{bmatrix} -\frac{3}{2} & 2 & -\frac{1}{2} \\ -\frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{2} & -2 & \frac{3}{2} \end{bmatrix} \begin{pmatrix} y_0 \\ y_1 \\ y_2 \end{pmatrix} \equiv F\vec{y}.$$

The matrix F collects the coefficients of the finite difference formulas, excluding the factor h^{-1}. Computationally, these can be acquired by first evaluating the derivative of the polynomial at the desire pivot, and then differentiating with respect to the ordinate y_i, which can only exist as a monomial.[3] Mathematically, if we have a three-point grid and wish to perform numerical differentiation, the multiplication $h^{-1}F\vec{y}$ produces the estimated derivatives at the grid points. Thus F here is an example of a *differentiation matrix*, whose product with the ordinate vector, scaled by the proper power of step size, yields the derivative.

This is readily generalized to using an arbitrary number of points for interpolation, the result of which would be a collection of finite difference formulas F. Similarly, we can obtain higher order derivatives, say k^{th} order, by differentiating the polynomial k times and then evaluating. It can be easily seen with Cramer's rule that for the k^{th} derivative, the differentiation matrix, when applied, should be scaled by the power h^{-k}.

The procedure hitherto described of interpolating a polynomial and evaluating its derivative at different pivot locations to obtain finite difference formulas is implemented in script *polydiff.m*. The complexity of this algorithm cannot be customarily estimated for the operations performed are symbolic rather than on floating point numbers. Indeed, the runtime for the collection of formulas of order greater than 4 is highly non-trivial. Thus, these formulas were pre-computed and stored as a native MATLAB array that is loaded with the differentiation module, up to using 41 points of interpolation. If a higher order formula is requested, it will be computed and memoized.

Now, estimates of a first derivative can be obtained with using different numbers of points to interpolate, using at least 2. The difference between using different degrees of polynomial interpolants is the asymptotic order of accuracy, denoted $O(h^p)$, where $p \in \mathbb{N}$. This states that $\exists\, \alpha \in \mathbb{R}$ so that

$$\lim_{h\to 0} \hat{f}'(x_i) - f'(x_i) = \alpha h^p. \tag{7}$$

Here, p is denoted the *order of accuracy* of a finite difference formula. We note that this is an asymptotic definition which may only materialize at small enough step sizes h. In general, the subsequent relation exists.

PROPOSITION. *For a function f that can be differentiated n times in some closed neighborhood, a finite difference formula obtained through a polynomial interpolant of n distinct points therein, of an order of differentiation k, and order of accuracy e obeys the following relation*

$$n = k + e.$$

It would be delightful to prove this relation. Albeit, it will not be given here. The author has verified it against the commonly used finite difference formulas with stated order of accuracy. It has also been conformed to empirically, as shall be seen shortly, for employed test cases. This relation and our earlier discussion offer us an elegant classification of finite difference formulas. A finite difference formula is defined by the number of distinct grid points n used to interpolate a polynomial function, the order of differentiation k, the pivot, or location where the derivative of the polynomial is evaluated, which, collectively, determine the order of accuracy.

[3] This can be easily realized using Cramer's rule.

Finally, we return to our question of applying finite difference formulas for numerical differentiation. For each grid point, we may elect to use some selection of neighboring points, not all the grid points, for interpolation. We'd like to apply finite difference formulas of the same order of accuracy. For these conditions, we define the differentiation matrix below.

DEFINITION. *For a set of n uniformly spaced abscissas, the **centric** differentiation matrix D of accuracy order e is a $n \times n$ matrix whose rows consist of coefficients in finite difference formulas.*

Specifically, for $2 \nmid e$, $\forall \left\lfloor \frac{e}{2} \right\rfloor \leq i \leq n - \left\lfloor \frac{e}{2} \right\rfloor + 1$, the i^{th} row of $\vec{d}_i$ has coefficients centered at index i corresponding to the central formula of accuracy order e, while all other elements of $\vec{d}_i$ is 0. For any other i, we use a formula of the same accuracy order that incorporates all the ordinates in the direction with fewer available data points, that is to either its right or its left, depending on the number of remaining points to the boundary.

For $2|e$, arbitrarily, we construct D using as the central formula the central difference expression of accuracy order e evaluated at pivot $e/2 + 1$. For an index where there is insufficient number of neighboring points in either direction, we similarly use a formula that incorporates all the available points in the deficient direction.

This definition is unclear on its own, and may not be so interesting to the reader. Thus we illustrate with an example. Below is the centric differentiation matrix of order $e = 4$ for the first derivative on a grid of 10 points, computed by our companion code *centricMatrix.m*.

$$\begin{bmatrix}
-2.0833 & 4.0000 & -3.0000 & 1.3333 & -0.2500 & 0 & 0 & 0 & 0 & 0 \\
-0.2500 & -0.8333 & 1.5000 & -0.5000 & 0.0833 & 0 & 0 & 0 & 0 & 0 \\
0.0833 & -0.6667 & 0 & 0.6667 & -0.0833 & 0 & 0 & 0 & 0 & 0 \\
0 & 0.0833 & -0.6667 & 0 & 0.6667 & -0.0833 & 0 & 0 & 0 & 0 \\
0 & 0 & 0.0833 & -0.6667 & 0 & 0.6667 & -0.0833 & 0 & 0 & 0 \\
0 & 0 & 0 & 0.0833 & -0.6667 & 0 & 0.6667 & -0.0833 & 0 & 0 \\
0 & 0 & 0 & 0 & 0.0833 & -0.6667 & 0 & 0.6667 & -0.0833 & 0 \\
0 & 0 & 0 & 0 & 0 & 0.0833 & -0.6667 & 0 & 0.6667 & -0.0833 \\
0 & 0 & 0 & 0 & 0 & -0.0833 & 0.5000 & -1.5000 & 0.8333 & 0.2500 \\
0 & 0 & 0 & 0 & 0 & 0.2500 & -1.3333 & 3.0000 & -4.0000 & 2.0833
\end{bmatrix}$$

This is a sparse matrix whose middle rows are a banded matrix with the $(0.0833, -0.6667, 0, 0.6667, -0.0833)$ entry repeated which corresponds to the central formula of 5^{th} order. These rows correspond to grid points with sufficient neighboring points on both sides, here 2 each, to apply the central formula. The first two and last two rows lack one or two points to the left and right respectively to apply the central formula. Thus for the second and second to last row, we use pseudo-centered formulas where one point is utilized on one side and three on the other. At the very boundaries, only a forward or backward formula can be applied that uses all points to the right or left.

Differentiation is performed by the multiplication

$$\hat{f}' = h^{-1} D \vec{y}$$

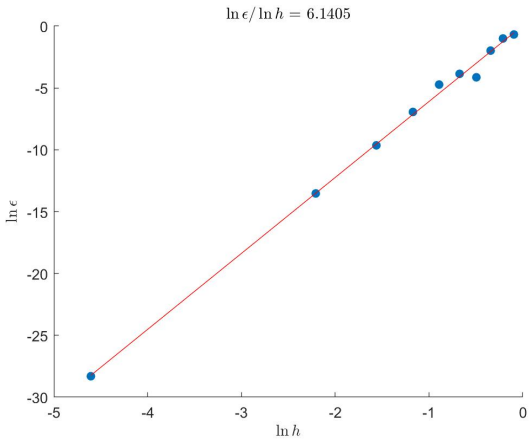

Figure 1. Empirical error order of first derivative.

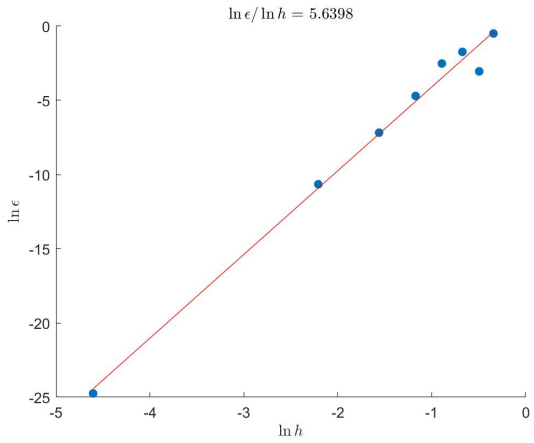

Figure 2. Empirical error order of second derivative.

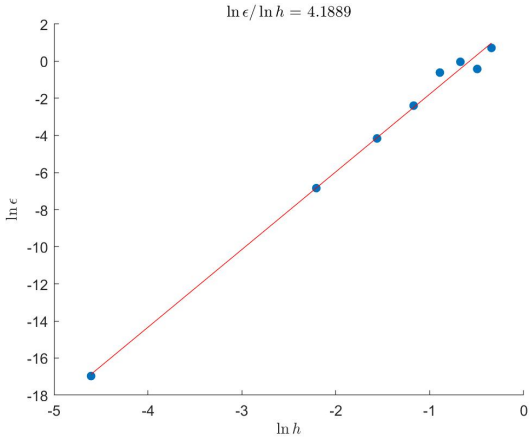

Figure 3. Empirical error order of third derivative.

Which computes the first derivative at all the grid points, all to the same theoretical order of accuracy. The motivation for preferring central formulas to construct the differentiation matrix is the idea that for differentiable functions, the behavior of change at a point is described by a two-sided locale, given sufficient resolution. This can be assessed more critically.

Below are several sample graphs showing the asymptotic convergence to the theoretical order of accuracy. We've chosen $f(x) = e^{\sin x}$ as the testing function, whose derivative is $e^{\sin x \cos x}$. Centric differentiation matrices of 6^{th}, 5^{th}, and 4th orders of accuracy for 1^{st}, 2^{nd}, and 3^{rd} derivatives. Here the error is defined as the mean absolute error of the derivatives estimated at all grid points used, which span $[0, 2\pi]$. The empirical orders of accuracy are printed, which agree generally with our expectation, though as seen for the second derivative, may not always be precise practically, depending on our choice of step sizes and our means of measuring error.

These graphs were produced by *orderAccuracy.m*.

3. Directional Differentiation in Multiple Dimensions

Succeeding our treatment of differentiation in one dimension, we can now fulfill the titular promise of this paper– generalization into multiple dimensions. For a multivariate vector function, the derivative is defined by component of the ordinate, and by direction of differentiation of the abscissa. Directional derivatives in all directions in the abscissa space can then be derived from the partial derivatives with respect to the standard basis vectors.

The procedure for computing partial derivatives are as follows. For a vector function $F : \mathbb{R}^n \longrightarrow \mathbb{R}^m$, with abscissa $\vec{x} = (x_1, x_2, ..., x_n)$, to compute $\frac{\partial y_j}{\partial x_i}$ at an arbitrary $\vec{x} = \vec{a} \equiv (a_1, a_2, .., a_n)$, consider the partial function $y(x_i) = F(a_1, a_2, ..., x_i, ...a_n)$, with all other $x_k \neq x_i$ held constant. Discretize this 1D function uniformly as $\vec{x}_i$ and $\vec{y}_i$; we thus have

$$\left.\frac{\partial y_j}{\partial x_i}\right|_{(a_1,a_2,...,\vec{x}_i,...a_n)} = h^{-1} D\vec{y}_i.$$

Hereby the problem is reduced to one dimension, where $\hat{x}_i$ is the direction of differentiation and $\vec{x}_i$ and $\vec{y}_i$ are the discretized abscissas and ordinates of a ray of grid points along that direction. We may then iterate through the components or the ordinate and the directions of differentiation to collect all the scalar partial derivatives.

This procedure may be referred to as *directional differentiation*, for only the functions values along the direction of differentiation are used by the finite difference formulas. More complex schemes of multidimensional differentiation can be devised assuming local differentiability of the function, that is, differentiability in all directions. Then, ordinate values not along the direction of differentiation may contribute to the estimated derivative, affording more continuity. This condition of differentiability may be important in various practical situations, though the simplest method of differentiation in multiple dimensions which assumes least of differentiability is as we have stated it.

Higher order derivatives are analogously obtained.

$$\left.\frac{\partial^q y_j}{\partial x_i^q}\right|_{(a_1,a_2,...,\vec{x}_i,...a_n)} = h^{-q} D^{(q)}\vec{y}_i \tag{8}$$

We note that mixed derivatives are obtained effectively by differentiating the derivative field along another direction of differentiation, so do not operate on the same ray of points. For this, the derivative field along the first direction is obtained and then differentiated with respect to the new direction.

Since at this stage, no symbolic operations are involved in differentiation, we consider the complexity of the algorithm. Differentiation of a ray of data, that is, the 1D problem, consumes $O(k + e)$ operations, which is the number of neighboring points involved in interpolation, per data point, so $O(n(k + e))$ for n total data points. For a vector grid of $n_1, n_2, \ldots, n_m$ dimension, the runtime complexity of computing the partial derivative of one of its components for the entire grid is $O(\prod_{i=1}^{m} n_i(k + e))$, equal to $O(n^m(k + e))$ for a symmetric grid. Such accounting is reasonable given that the finite differencing operations performed at each grid point is of order $O(k + e)$. Complexity is multiplied by the dimension of the abscissa and the dimension of the ordinate for a full computation of the k^{th} partial derivatives of accuracy order e.

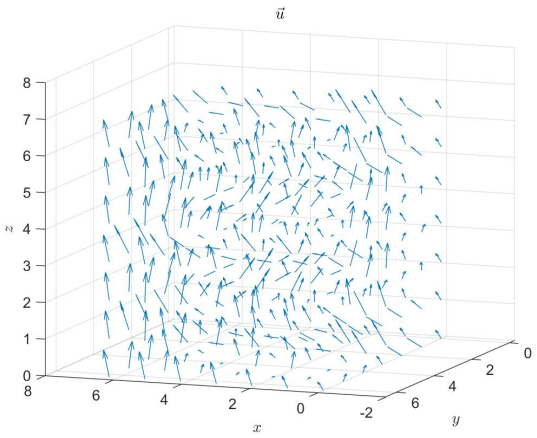

Figure 4. Low-resolution plot of synthetic field

Directional differentiation is implemented for vector fields $F : \mathbb{R}^3 \longrightarrow R^3$ in *VelocityField.diff()*, *VelocityField.gradient()*, and *VelocityField.vorticity()*, which we now utilize for a case study.[4]

4. A Periodic Field

$$\vec{\boldsymbol{u}} = \Big(\sin(y\cos z), \cos(x\sin z), e^{\sin(xy)}\Big) \tag{9}$$

We invent an infinitely differentiable vector field, which owns a certain periodicity in its structure. It can be instantiated with various resolutions, that is, spacing between vectors which will correspond to step size during differentiation. Below is a sample on $[0, 2\pi]$ with uniform increments of $\pi/3$ in all three dimensions, a coarsely resolved sample.

The analytical solutions of the first-derivative quantities are derived directly from the field's definition. We observe that each component of the field is independent of the variation of its corresponding abscissa dimension; hence the null divergence.

$$\nabla\vec{\boldsymbol{u}} = \begin{bmatrix} 0 & \cos z\cos(y\cos z) & -y\sin z\cos(y\cos z) \\ -\sin z\sin(x\sin z) & 0 & -x\cos z\sin(x\sin z) \\ y\cos(xy)e^{\sin(xy)} & x\cos(xy)e^{\sin(xy)} & 0 \end{bmatrix} \tag{10}$$

$$\nabla\times\vec{\boldsymbol{u}} = \begin{pmatrix} x\cos(xy)e^{\sin(xy)} + x\cos z\sin(x\sin z) \\ -y\sin z\cos(y\cos z) - y\cos(xy)e^{\sin(xy)} \\ -\sin z\sin(x\sin z) - \cos z\cos(y\cos z) \end{pmatrix} \tag{11}$$

$$\nabla\cdot\vec{\boldsymbol{u}} = 0 \tag{12}$$

[4] *VelocityField.m* is a data structure I'm developing for manipulating 3D vector fields interpreted as fluidic velocity fields. Its functions of numerical differentiation is implemented by the polynomial differentiation method we've described [2].

4.1. Accuracy of Partial Derivatives

Were we to compute the partial derivative of one component along a ray of grid points, say $u_x(x_0, y, z_0)$, the problem reduces to the 1D case, where the order of accuracy agrees with the degree of the interpolant polynomial, given differentiability of this synthetic field. Theoretically, since a partial derivative at a position is determined solely by the interpolation along the direction of differentiation, the order of error necessarily follows the declared order of the finite difference. Thus the order of accuracy of the partial derivative along any ray is identical to that on one ray.

Nevertheless, as the partial function $u_x(x_0, y, z_0)$ varies with x_0 and z_0, the rate of error's decrease varies amongst $u_x(x, y, z)$. Since the order of accuracy refers to *asymptotic* behavior of error as $h \to 0$, the characteristic decrease of error as a power function may or may not manifest depending on our choice of step sizes used. With many rays, thereby many partial functions, approaching asymptotic behavior at different rates, the problem is posed that, depending on the method by which error is measured, the theoretical order of accuracy may or may not be observed.

We shall substantiate these statements now with our vector field. Below are the log-error plots of the non-zero partial derivatives generated with a 7^{th} order accurate centric matrix. (Figures 5-10) The error here is measured by computing the mean absolute error[5] of the estimated partial derivative over all 3D grid points, normalized by the mean theoretical value of that partial derivative over all grid points. This is a reasonable way of measuring error: using the mean error scaled by the mean theoretical, which crudely corresponds to the percentage error.

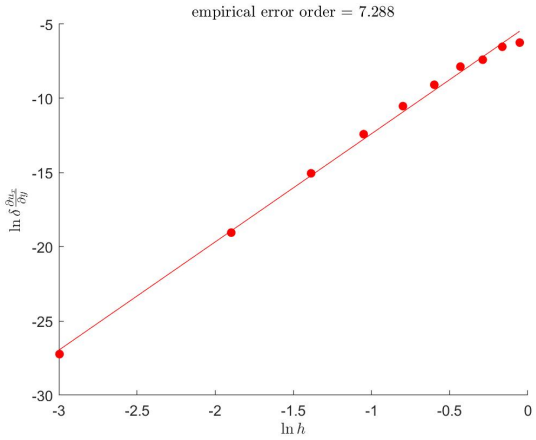

Figure 5. Error of $\frac{\partial u_x}{\partial y}$

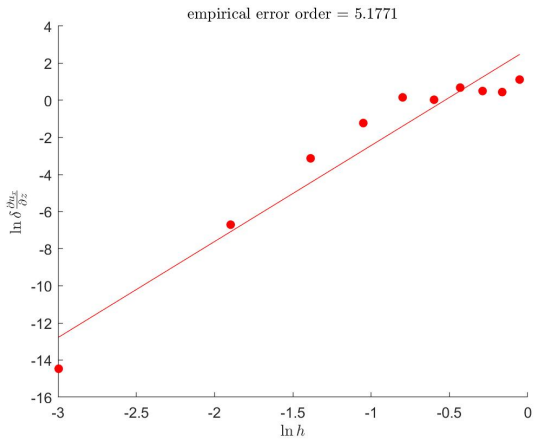

Figure 6. Error of $\frac{\partial u_x}{\partial z}$

We see that only in two of the dimensions does the empirical order of error matches the theoretical, namely for $\frac{\partial u_x}{\partial y}$ and $\frac{\partial u_y}{\partial x}$.[6] These two partial derivatives, we observe from (9), originate from basic sine and cosine oscillations whose derivative is not difficult to approximate. The other four partial derivatives, nonetheless, involve more complex variations that are composite, so generate more chaotic oscillations.

If we pick any of the irregular derivatives and compute error along a specific ray, we obtain

[5] All error discussed in this paper is absolute error, or the norm, when the error is a vector.

[6] These graphs, as well as the later one for vorticity, were generated by the script *orderAccuracy3D.m*.

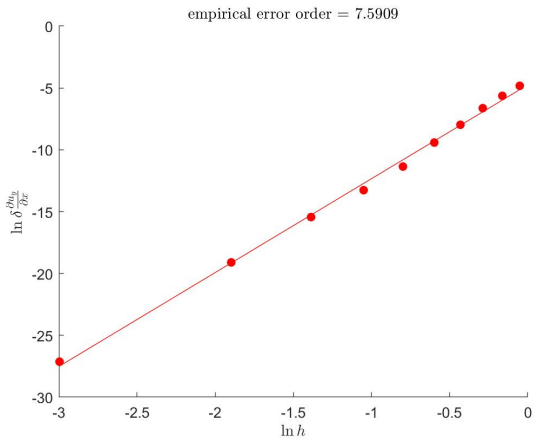

Figure 7. Error of $\frac{\partial u_y}{\partial x}$

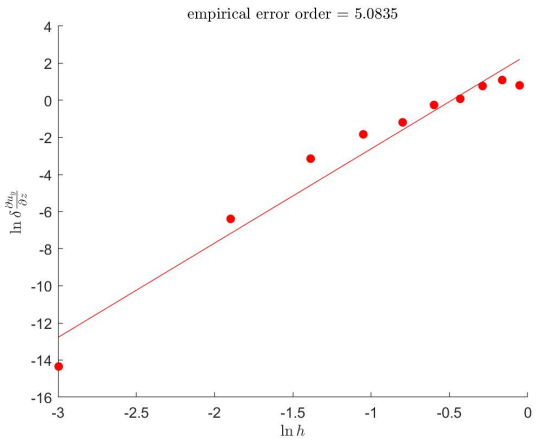

Figure 8. Error of $\frac{\partial u_y}{\partial z}$

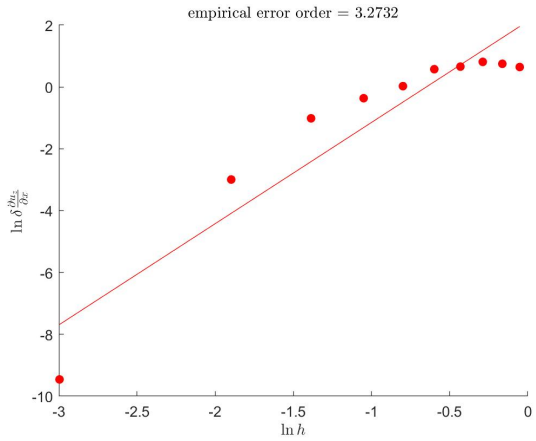

Figure 9. Error of $\frac{\partial u_z}{\partial x}$

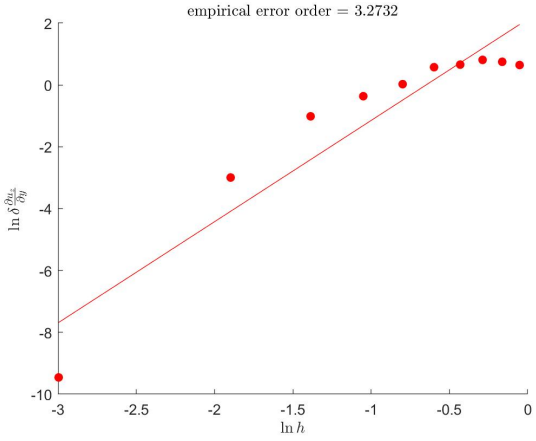

Figure 10. Error of $\frac{\partial u_z}{\partial y}$

the expected order of accuracy by diminishing the step size. In Figure 11, we have computed the error of $\frac{\partial u_z}{\partial y}$ along a ray with $x = 2\pi$, which causes quick oscillations.[7] The minimum step size employed here is about 2.5 orders smaller than that used in 3D. We might hope to procure a similar trend as well using these minuter step sizes. This, however, reveals a practical limitation of numerical differentiation in multiple dimensions. Our field spans from 0 to 2π in each dimension, so the amount of memory required for the construction of a field scales inverse-cubically with the size of the step. $\ln h = -4$ corresponds to a $348 \times 348 \times 348$ array, requiring more than 3 gigabytes in MATLAB. Decreasing the step size one order farther requires 27 gigabytes, implausible for a typical personal computer. Thus, though the theoretical order of accuracy is still attained at high enough resolution, practically there might be no way to approach the asymptope.[8] Thus, for an $O(h^p)$ accurate difference, it is more likely for multidimensional data that halving the step size will not produce the diminution of error by the factor of a desired power 2^p, depending on the nature of the data. Other irregular partial derivatives were analogously verified, showing the same effect.

[7] This figure was made with the same script *orderAccuracy.m* as used earlier for the 1D problem.

[8] Another practical constraint, though not significant here, is that when the variation of the function is ultraminute, values of error and step size might be in the range of machine precision, thus inaccurate for determining the order of accuracy.

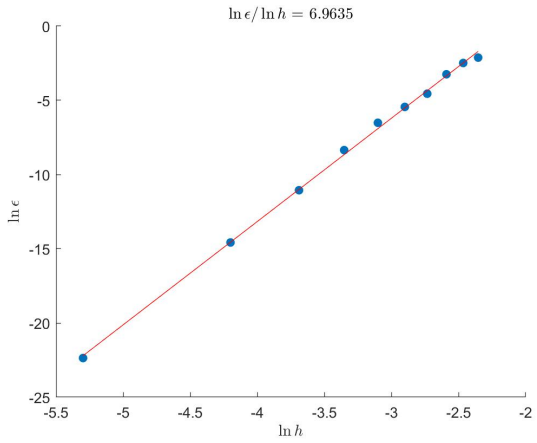

Figure 11. Irregular 3D error profile of $\frac{\partial u_z}{\partial y}$ becomes regular reduced to 1D.

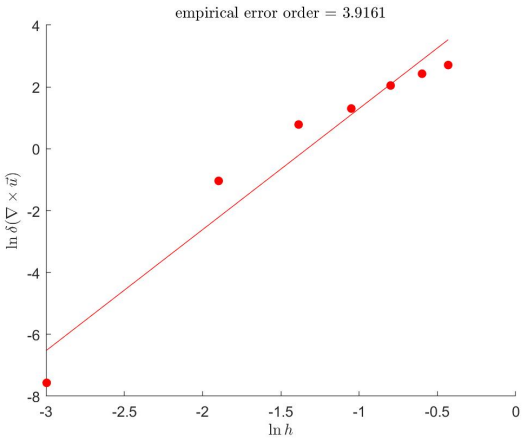

Figure 12. Empirical order of accuracy of $\nabla \times \vec{u}$

4.2. ACCURACY OF CURL

Having realized now that the partial derivatives involved in the gradient approach asymptotic behavior at different rates, we combine them in the computation of $\nabla \times \vec{u}$, known as the vorticity when $\vec{u}$ is a valid velocity field of a fluid. We expect that $\nabla \times \vec{u}$ to not reach asymptotic behavior, illustrated by a lower order of empirical accuracy.

Error is measured here as the mean magnitude of difference between the computed and theoretical curls, normalized by the mean theoretical curl magnitude in the 3D region. Still we have applied a 7^{th} order centric differentiation matrix and measure an empirical order around 4. (Figure 12) This is intermediate between the error orders of the partial derivatives, so appears reasonable given that the quicker decaying errors ameliorate the slower decaying ones in the average. We expect the theoretical order to evince with more minute step sizes, but again are unable to do so given limited computing power.

Similar to the 1D situation, we construct a plot for the error of curl approximation with different orders of accuracy. Here a vector field of the given resolution is created, on which finite differences of various orders are then applied to compute the vorticity. When the step size is small, an apparently exponential drop of mean error is observed, due presumably to the increasing power of h in $O(h^p)$. Again, we recognize the problem of "overfitting" at low

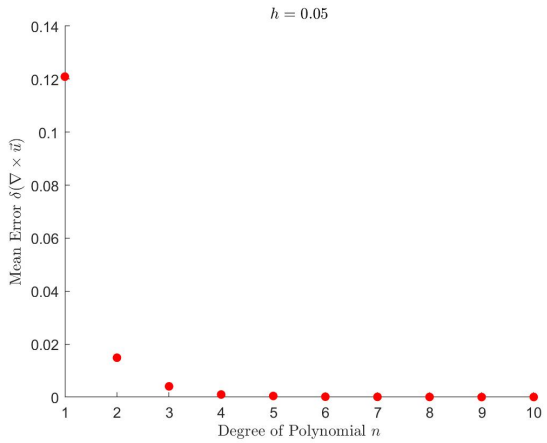

Figure 12. Variation of error with accuracy order for high-resolution data.

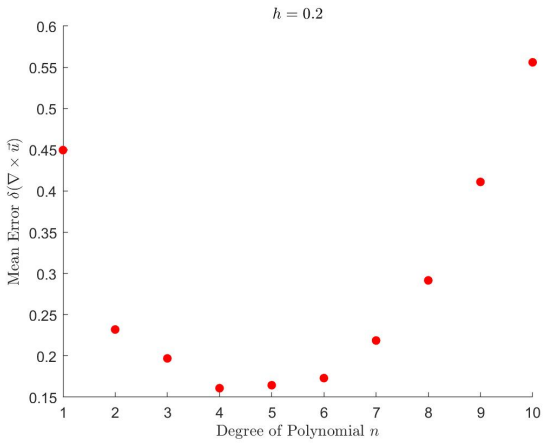

Figure 13. Variation of error with accuracy order for low-resolution data.

resolutions where a high degree polynomial interpolant introduces spurious patterns of change between given points.

These graphs were produced in *vort_polyorder.m*.

5. Conclusion and Critique

Differentiation by polynomial interpolation is a traditional method of numerical analysis that has been well-studied. The case study we have entertained here of deriving finite difference expressions and organizing them as a centric differentiation matrix is probably more for the student's edification than of any contribution. Implementing these as a computation module, nonetheless, has practical value since it allows for automated generation of finite differences that can be applied in related problems.

Numerical differentiation in higher dimensions is an intriguing and expansive field of study. We have observed here, through application of directional differentiation, that convergence patterns expected from 1D may not be as easily obtained given the added complexity of multiple dimensions and the additional variations these introduce. We've also realized, by experience, the constraint of computing power introduced in higher dimensions, which develops characteristically as $O(n^m)$ for memory usage, where m is the number of dimensions.

More importantly, practical factors inevitably involved in experimental studies utilizing numerical differentiation were not discussed. Perhaps the most significant factor, noise is not considered in this study, the treatment of which is indispensable in analysis of experimental data. Typical treatment of noise may be applying a smoother to the noisy data prior to differentiation. Nonetheless, the averaging involved in smoothing may destroy local features of change. More advanced techniques such as smoothing polynomials or smoothing splines can be applied in such situations [3]. A more thorough study of the finite difference approximation of derivatives needs to consider and address the effect of noise.

Maybe the absence of a module for finite differences in MATLAB and Python bespeaks the presence of superior techniques, such as spectral differentiation. Toward the prospect of studying these, I close our discussion presently.

6. REFERENCES

1. Martin Sleziak. *Why are Vandermone matrices invertible?*. 2014. "https://math.stackexchange.com/questions/426932/why-are-vandermonde-matrices-invertible
2. Derek Li & Leah Mendelson. Library for error analysis of 3D PIV velocity fields. https://github.com/epicderek/flow
3. Epps, B.P., Truscott, T.T., & Techet, A.H.. "Evaluating derivatives of experimental data using smoothing splines." 3rd Mathematical Methods in Engineering International Symposium (MME'10), Coimbra, Portugal, October 21-24, 2010.