Fundamental Methods of Numerical Extrapolation With Applications

Eric Hung-Lin Liu
ehliu@mit.edu

Keywords: numerical analysis, extrapolation, richardson, romberg, numerical differentiation, numerical integration

Abstract

Extrapolation is an incredibly powerful technique for increasing speed and accuracy in various numerical tasks in scientific computing. As we will see, extrapolation can transform even the most mundane of algorithms (such as the Trapezoid Rule) into an extremely fast and accurate algorithm, increasing the rate of convergence by more than one order of magnitude. Within this paper, we will first present some background theory to motivate the derivation of Richardson and Romberg Extrapolation and provide support for their validity and stability as numerical techniques. Then we will present examples from differentiation and integration to demonstrate the incredible power of extrapolation. We will also give some MATLAB code to show some simple implentation methods in addition to discussing error bounds and differentiating between "'true"' and "'false"' convergence.

1 Introduction

In many practical applications, we often find ourselves attempting to compute continuous quantities on discrete machines of limited precision. In several classic applications (e.g. numerical differentiation), this is accomplished by tending the step-size to zero. Unfortunately as we will see, this creates severe problems with floating point round-off errors. In other situations such as numerical integration, tending the step-size toward zero does not cause extreme round-off errors, but it can be severely limiting in terms of execution time: a decrease by a factor of 10 increases the required function evaluations by a factor of 10.

Extrapolation, the topic of this paper, attempts to rectify these problems by computing "weighted averages" of relatively poor estimates to eliminate error modes–specifically, we will be examining various forms of Richardson Extrapolation. Richardson and Richardson-like schemes are but a subset of all extrapolation methods, but they are among the most common. Moreover they have numerous applications and are not terribly difficult to understand and implement. We will demonstrate that various numerical methods taught in introductory calculus classes are in fact insufficient, but they can all be greatly improved through proper application of extrapolation techniques.

1.1 Background: A Few Words on Series Expansions

We are very often able to associate some infinite sequence $\{A_n\}$ with a known function $A(h)$. For example, $A(h)$ may represent a first divided differences scheme for differentiation, where h is the step-size. Note that h may be continuous or discrete. The sequence $\{A_n\}$ is described by: $A_n = A(h_n)$, $n \in \mathbb{N}$ for a monotonically decreasing sequence $\{h_n\}$ that converges to 0 in the limit. Thus $\lim_{h\to 0+} A(h) = A$ and similarly, $\lim_{n\to\infty} A_n = A$. Computing $A(0)$ is exactly what we want to do–continuing the differentiation example, roughly speaking $A(0)$ is the "point" where the divided differences "becomes" the derivative.

Interestingly enough, although we require that $A(y)$ have an asymptotic series expansion, we are only curious about its form. Specifically, we will examine functions with expansions that take this form:

$$A(h) = A + \sum_{k=1}^{s} \alpha_k h^{p_k} + \mathrm{O}(h^{p_{s+1}}) \qquad \text{as } h \to 0+ \tag{1}$$

where $p_n \neq 0\ \forall n$, $p_1 < p_2 < ... < p_s + 1$, and all the α_k are independent of y. The p_n can in fact be imaginary, in which case we compare only the real parts. Clearly, having $p_1 > 0$ guarantees the existence of $\lim_{h\to 0+} A(h) = A$. In the case where that limit does not exist, A is the *antilimit* of $A(h)$, and then we know that at $p_i \leq 0$ for at least $i = 1$. However, as long as the sequence $\{p_n\}$ is monotonically increasing such that $\lim_{k\to\infty} p_k = +\infty$, (1) exists and $A(h)$ has the expansion $A(h) \sim A + \sum_{k=1}^{\infty} \alpha_k h^{p_k}$. We do not need to declare any conditions on the convergence of the infinite series. We do assume that the h_k are known, but we do not need to know the α_k, since as we will see, these constant factors are not significant.

Now suppose that $p_1 > 0$ (hence it is true $\forall p$), then when h is sufficiently small, $A(y)$ approximates A with error $A(h) - A = \mathrm{O}(h^{p_1})$. It is worth noting that if p_1 is rather large, then $A(h) - A \approx 0$ even for values of y that are not very small. However in most applications, h_1 will be small (say $p_1 < 4$), so it would appear that we must resort to smaller and smaller values of h. But as previously noted, this is almost always prohibitive in terms of round-off errors and/or function evaluations. But we should not despair, because we are now standing on the doorstep of the fundamental idea behind Richardson Extrapolation. That idea is to use equation (1) to somehow eliminate the h^{p_1} error term, thus obtaining a new approximation $A_1(h) - A = \mathrm{O}(h^{h_2})$. It is obvious that $\mid A_1(h) - A \mid < \mid A(h) - A \mid$ since $p_2 > p_1$. As we will see later, this is accomplished by taking a "weighted average" of $A(h)$ and $A(\frac{h}{s})$.

2 Simple Numerical Differentiation and Integration Schemes

Before continuing with our development of the theory of extrapolation, we will briefly review differentiation by first divided differences and quadrature using the (composite) trapezoid and midpoint rules. These are among the simplest algorithms; they are straightforward enough to be presented in first-year calculus classes. Being all first-order approximations, their error modes are significant, and they have little application in real situations. However as we will see, applying a simple extrapolation scheme makes these simple methods extremely powerful, convering difficult problems quickly and achieving higher accuracy in the process.

2.1 Differentiation via First Divided Differencs

Suppose we have an arbitrary function $f(x) \in C^1[x_0 - \epsilon, x_0 + \epsilon]$ for some $\epsilon > 0$. Then apply a simple linear fit to $f(x)$ in the aforementioned region to approximate the $f'(x)$. We can begin by assuming the existence of additional derivatives and computing a Taylor Expansion:

$$f(x) = f(x_0) + f'(x_0)(x - x_0) + \frac{1}{2}f''(x_0)(x - x_0)^2 + \mathrm{O}(f(h^3))$$

where $h = x - x_0$ and $0 < h \leq \epsilon$

$$f'(x_0) = \frac{f(x + h) - f(x)}{h} - \frac{1}{2}f''(x)h + \mathrm{O}(h^2)$$

Resulting in:

$$f'(x_0) = \frac{f(x_0 + h) - f(x_0)}{h} \tag{2}$$

From the Taylor Expansion, it would seem that our first divided difference approximation has truncation error on the order of $\mathrm{O}(h)$. Thus one might imagine that allowing $h \to 0$ will yield $\mathrm{O}(h) \to 0$ at a rate linear in h. Unfortunately, due the limitations of finite precision arithmetic, performing $h \to 0$ can only decrease truncation error a certain amount before arithmetic error becomes dominant. Thus selecting a very small h is not a viable solution, which is sadly a misconception of many Calculus students. Let us examine exactly how this arithmetic error arises. Note that because of the discrete nature of computers, a number a is represented as $a(1 + \epsilon_1)$ where ϵ_1 is some error.

$$\begin{aligned}(a - b) &= a(1 + \epsilon_1) - b(1 + \epsilon_2) \\ &= (a - b) + (a\epsilon_1 - b\epsilon_2) + (a - b)\epsilon_3 \\ &= (a - b)(1 + \epsilon_3) + (\mid a \mid + \mid b \mid)\epsilon_4\end{aligned}$$

Thus the floating point representation of the difference numerator, $fl[f(x + h) - f(x)]$ is $f(x + h) - f(x) + 2f(x)\epsilon + ...$ where $2f(x)\epsilon$ represents the arithmetic error due to subtraction. So the floating point representation of the full divided difference is:

$$f'(x) = fl[\frac{f(x + h) - f(x)}{h}] + \frac{2f(x)\epsilon}{h} - \frac{1}{2}f''(x)h + ... \tag{3}$$

Again, $\frac{2f(x)\epsilon}{h}$ is the arithmetic error, while $\frac{1}{2}f''(x)h$ is the truncation error from only taking the linear term of the Taylor Expansion. Since one term acts like $\frac{1}{h}$ and the other like h, there exists an optimum choice of h such that the total error is minimized:

$$h \approx 2\left(\frac{f(x)}{f''(x)}\epsilon\right)^{\frac{1}{2}} \tag{4}$$

Implying that $h_{optimum} \sim \sqrt{eps}$ where eps is machine precision, which is roughly 10^{-16} in IEEE double precision. Thus our best error is $4(f''(x)f(x)\epsilon)^{\frac{1}{2}} \sim 10^{-8}$. This is rather dismal, and we certainly can do better. Naturally it is impossible to modify arithemtic error: it is a property of double precision representations. So we look to the truncation error: a prime suspect for extrapolation.

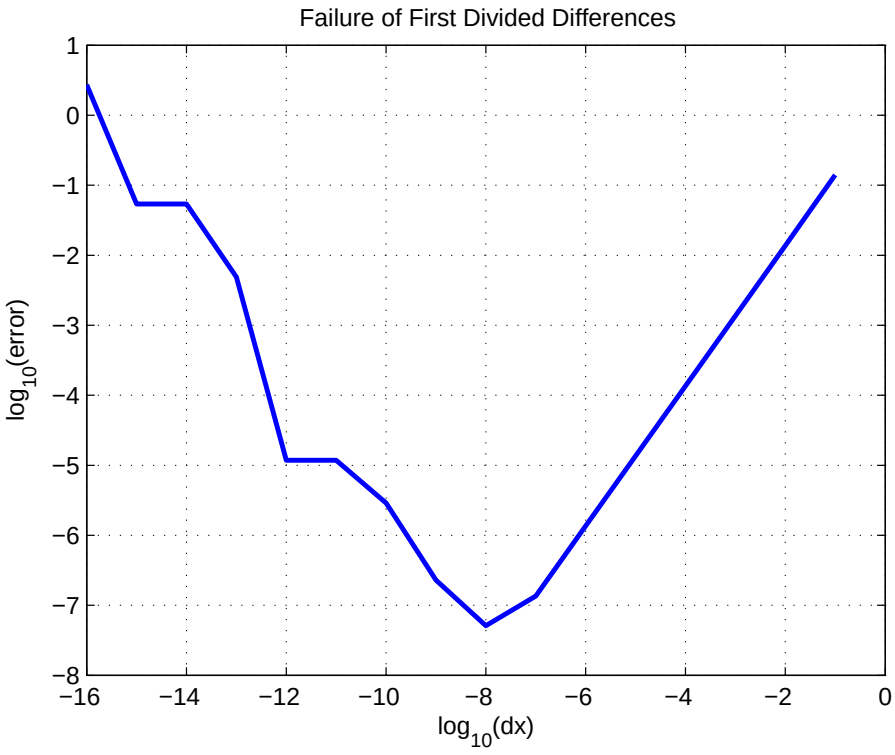

Figure 1: Approximation errors from $\frac{de^x}{dx} \mid_{x=1}$

The extent of this problem can be seen in Fig. 1. It displays the numerical errors resulting from the first divided difference approximation of $\frac{de^x}{dx} \mid_{x=1}$. The y-axis represents the logarithm of the error, while the x-axis shows the approximate step size. The log-log scale is convenient for visualizing rough order of magnitude approximations for numerical errors with the additional benefit of linearizing functions with polynomial error expressions. This method is commonplace in numerical analysis.

Note that for practical purposes, we should not be applying first divided differences to estimate derivatives. The previous method is $\mathrm{O}(h)$. By using centered divided differences, we can obtain $\mathrm{O}h^2$ without any additional work. We initially avoided this because the example of numerical errors is most severe using first differences. The formula for centered differences approximation is:

$$\delta_0(h) = \frac{f(x_0 + h) - f(x_0 - h)}{2h} \qquad \text{for } 0 < h \leq a \tag{5}$$

The derivative approximation then has error:

$$\delta_0(h) - f'(x_0) = \frac{f'''(\xi(h))}{3!} h^2 = \mathrm{O}(h^2) \tag{6}$$

Sidi provides the full expansion, obtained with a Taylor series expansion around x_0:

$$\begin{aligned} \delta_0(h) - f'(x_0) =& \sum_{k=1}^{s} \frac{f^{2k+1}(x_0)}{(2k+1)!} h^{2k} + R_s(h) \quad \text{where} \\ R_s(h) =& \frac{f^{(2s+3)}(\xi(h))}{(2s+3)!} h^{2s+2} = \mathrm{O}(h^{2s+2}) \end{aligned} \tag{7}$$

2.2 Integration via the Trapezoid and Midpoint Rules

First we should note that since the integration schemes presented are used over a range of interpolation points $\{x_j\}$, the given formulas correspond to the Composite Trapezoid and Midpoint schemes, although we will drop the "composite" label.

$$\text{Composite Trapezoid Rule: } \int_a^b f(x)\,dx = \frac{h}{2}\left(f(a) + 2\sum_{j=1}^{n-1} f(x_j) + f(b)\right) + \mathrm{O}(h^2) \qquad (8)$$

For a uniformly chosen set of points $\{x_j\}$ in the range $[a, b]$ such that $x_j = a + jh$ and $j = 1/n$ where n is the total number of interpolation points. Observe that all the Trapezoid Rule really does is fit a line to the function between each pair of points in the interval.

The Trapezoid Rule is a special case ($n = 1$) of the *(n+1)-point closed Newton-Cotes formula*. Also, Simpson's Rule is another special case ($n = 2$) of this formula. The composite rules arise from iterating the Newton-Cotes expressions over a range of points. The following is adapted from Burden and Faires:

Theorem 1. *Suppose that* $\sum_{i=0}^{n} a_i f(x_i)$ *denotes the (n+1)-point closed Newton-Cotes formula with* $x_0 = a$, $x_n = b$, *and* $h = \frac{b-a}{n}$. *Then there exists* $\xi \in (a, b)$ *such that*

$$\int_a^b f(x)dx = \sum_{i=0}^{n} a_i f(x_i) + \frac{h^{n+3} f^{n+2}(\xi)}{(n+2)!} \int_0^n t^2 \prod_{j=1}^{n}(t-j)dt$$

if n *is even and* $f \in \mathrm{C}^{n+2}[a, b]$, *and*

$$\int_a^b f(x)dx = \sum_{i=0}^{n} a_i f(x_i) + \frac{h^{n+2} f^{n+1}(\xi)}{(n+1)!} \int_0^n t \prod_{j=1}^{n}(t-j)dt$$

if n *is odd and* $f \in \mathrm{C}^{n+1}[a, b]$.

Proof. As implied in the introduction, we are assuming the validity of the following expansion:

$$\int_a^b f(x)dx \approx \sum_{i=0}^{n} a_i f(x_i)$$

where

$$a_i = \int_{x_0}^{x_n} L_i(x)dx = \int_{x_0}^{x_n} \prod_{\substack{j=0 \\ j\neq i}}^{n} \frac{x - x_j}{x_i - x_j} dx$$

with L_i being the i-th term of the *Lagrange Interpolating Polynomial*. Then we can exploit the properties of the Lagrange Interpolating Polynomial along with Taylor Expansions to arrive at the result. The proof is omitted here, but a very detailed explanation can be found in Issacson and Keller along with more details about the specific error analysis. An alternative, more difficult derivation that avoids the Lagrange Interpolating Polynomial can be found in Sidi. □

There is a parallel *(n+1)-point open Newton Cotes formula* that does not evaluate the endpoints a, b, which is useful for avoiding endpoint singularities as in $int_0^1 \frac{1}{x} dx$. The Midpoint Rule is a member of this group.

Theorem 2. *Suppose that* $\sum_{i=0}^{n} a_i f(x_i)$ *denotes the (n+1)-point closed Newton-Cotes formula with* $x_{-1} = a$, $x_{n+1} = b$, *and* $h = \frac{b-a}{n+2}$. *Then there exists* $\xi \in (a, b)$ *such that*

$$\int_a^b f(x)dx = \sum_{i=0}^{n} a_i f(x_i) + \frac{h^{n+3} f^{n+2}(\xi)}{(n+2)!} \int_{-1}^{n+1} t^2 \prod_{j=1}^{n} (t-j)dt$$

if n *is even and* $f \in \mathrm{C}^{n+2}[a, b]$, *and*

$$\int_a^b f(x)dx = \sum_{i=0}^{n} a_i f(x_i) + \frac{h^{n+2} f^{n+1}(\xi)}{(n+1)!} \int_{-1}^{n+1} t \prod_{j=1}^{n} (t-j)dt$$

if n *is odd and* $f \in \mathrm{C}^{n+1}[a, b]$.

Proof. The proof is analagous to Theorem 1, employing Lagrange Interpolating Polynomials in the same fashion. Again, the full proof can be found in Issacson and Keller. □

For completeness, here is the Composite Midpoint Rule. Again, the error term has been simplified because the exact coeffcients are not of interest, as we will soon see. Additionally, notice that as with the Composite Trapezoid Rule, the error term power is one less than the one expressed by the Newton-Cotes formulas. This is simply because the Composite rule has summed n "copies" of the simple Newton-Cotes rules.

$$\text{Composite Midpoint Rule: } \int_a^b f(x)\ dx = h \sum_{j=1}^{n-1} f(a + \frac{jh}{2}) + \mathrm{O}(h^2) \tag{9}$$

Now, having reviewed the basics of numeric integration and differentiation, we will finally introduce the Richardson (differentiation) and Romberg (integration) Extrapolation procedures. The reader may find it surprising that only these most basic of numeric algorithms are necessary to procede, but the reason for this will soon become clear. The two methods are in fact essentially the same, so we will only derive the Romberg algorithm. Then we will present numeric results demonstrating the superiority of extrapolation over the basic algorithms. We will also show that one can arrive at Simpson's rule simply by performing extrapolation on the Trapezoid rule. (Note how we have avoided the most simple schemes of left and right hand sums. The reasoning is left to the reader.)

3 The Richardson and Romberg Extrapolation Technique

3.1 Conceptual and Analytic Approach

The fundamental idea behind these extrapolation schemes is the usage of multiple, low-accuracy evaluations to eliminate error modes and create a highly accurate result. As mentioned earlier, we are assuming that the working functions have the following property: $A(h) - A = \mathrm{O}(h^{p_1})$, where p_1 is known or obtainable.

To motivate our understanding of extrapolation, we will write some arbitrary quadrature rule I as a function of its associated step-size, h: $I(h)$. For example, the Trapezoid Rule could be written as:

$$I(h) = \frac{h}{2}\left(f(a) + 2\sum_{j=1}^{n-1} f(x_j) + f(b)\right) + \mathrm{O}(h^2)$$

As previously mentioned, suppose the expansion $I(h) = I_0 + h^{p_1}k_1 + hp_2k_2 + \ldots$ exists, where $\{p_i\} \subset \mathbb{R}$ are the set of error powers and $\{k_i\} \subset \mathbb{R}$ are some known or unknown constants. Now consider:

$$I(\frac{h}{s}) = I_0 + \frac{h^{p_1}k_1}{s^{p_1}} + \frac{h^{p_2}k_2}{s^{p_2}} + \ldots$$

To eliminate the h^{p_1} error term (because it is the dominant error power), we will compute $-I(h) + s^{p_1}I(\frac{h}{s})$.

$$\begin{aligned}
I'(h) &= s^{p_1}I(\frac{h}{s}) - I(h) = s^{p_1}I_0 - I_0 + \boxed{0 * \mathrm{O}(h^{p_1})} + \frac{h^{p_2}k_2}{s^{p_1-p_2}} - h^{p_2}k_2 + \ldots && (10)\\
&= \frac{s^{p_1}I(\frac{h}{s}) - I(h)}{s^{p_1} - 1} = I_0 + \frac{s^{p_1-p_2} - 1}{s^{p_1} - 1}h^{p_2}k_2 + \ldots && (11)\\
&= I_0 + h^{p_2}k_2' + \ldots && (12)\\
&= I(\frac{h}{s}) - \frac{I(h) - I(\frac{h}{s})}{s^{p_1} - 1} && (13)
\end{aligned}$$

And that is the basic idea underlying Romberg Extrapolation. Note that s is the weight by which we decrease the step-size in each new evaluation of I. $s = 2$ is a very common choice. From (14), we can continue iteratively (or recursively) to elimate $\mathrm{O}(h^{p_2})$. Using that result, we can eliminate $\mathrm{O}(h^{p_3})$, and continue on indefinitely, in theory. In reality, we are limited by floating point precision.

The following table presents a graphical representation of the extrapolation process. Notice that the left-most column is the *only* column that involves actual function evaluations. The second column represents estimates of I with the $\mathrm{O}(h^{p_1})$ term eliminated using the algorithm given above. The third column elmiminates $\mathrm{O}(h^{p_2})$ and so forth. The actual approximations that should be used for error checking can be found by reading off the entries on the main diagonal of this matrix.

Table 1: Iteration of a Generic Extrapolation Method

$I_0(h)$				
$I_0(\frac{h}{s})$	$I_1(h)$			
$I_0(\frac{h}{s^2})$	$I_1(\frac{h}{s})$	$I_2(h)$		
$I_0(\frac{h}{s^3})$	$I_1(\frac{h}{s^2})$	$I_2(\frac{h}{s})$	$I_3(h)$	
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\ddots$

Note: more generally, we need not decrease h geometrically by factors of $\frac{1}{s}$, but this approach is often easiest to visualize, and it simplifies the algebra in most applications without damaging performance. Common choices for s are small integers (e.g. $s = 2$) because larger integers may cause the number of function evaluations to grow prohibitively quickly.

3.2 A Few Words on Implementation

When implementing Richardson or Romberg extrapolation, it is most common to do it iteratively, unless storage is an extreme issue. Although the problem can be solved just as easily using recursion, iteration is generally preferred. In an iterative method, storage of the full table/matrix is not neccessary. Conveniently, it is only necessary to store the last two rows. Although one should keep at least a short history (1 or 2 entries) from the main diagonal for error comparisions. That is to say, when the difference between successive estimates decreases to some given tolerance (the smallest possible tolerance being machine-zero), we should terminate the algorithm because we are done. However it is possible the extrapolation method to stall locally and generate two successive entries that are within tolerance even though the result may be relatively far from the desired precision. Thus it is common practice to check errors across every second or third term to avoid this problem. The following example demonstrates a simple implementation that avoids storing the full table using MATLAB. It only checks for termination in successive terms, and it assumes the error powers follow the arithmetic progression 2, 4, 6, ...

```
while log10(abs(diff(I)))>log10(eps) %Only checks error of successive terms
    h(k)=(b-a)/2^k; %step-size for the next row
    R(2,1)=midpoint(a,b,h(end)); %Uses midpoint rule to obtain the first entry
    for j=2:k  %perform extrapolation steps
        R(2,j)=R(2,j-1)+(R(2,j-1)-R(1,j-1))/(4^(j-1)-1);
    end
    I(k)=R(end); %store latest result in an array
    for m=1:k %copy the second row to the first row
        R(1,m)=R(2,m);
    end %the next iteration will overwrite the current second row
    k=k+1;
end
```

If the full matrix is desired, the for loop would be similar to the following. Here, we allow for arbitrary error powers.

```
%xs are the x-locations along the axis where the function is evaluated
%ys are the corresponding y-values
%p is the array of error powers
  n = length(xs);
  R = zeros(n); %create the table
  R(:,1) = ys;
  for j=1:n-1,
    for i=1:(n-j),
      R(i,j+1)=R(i,j)+(R(i,j)-R(i+1,j))/((xs(i+j)/xs(i))^(p(j)/j)-1);
    end
  end
```

4 Graphical Demonstration of Results

4.1 Differentiation and Integration Revisited

To see the power of extrapolation, we present two illustrative examples. First, we return to $\frac{de^x}{dx}\ |_{x=1}$. Figure 2 shows the original plot of the first divided difference approximation followed by the un-extrapolated centered difference (red). The two black lines represent one and two extrapolation steps with the single step being naturally less accurate. Thus we can see that extrapolation increased the rate of convergence here by an incredible amount. Not only that, but it also handed us 14 decimal places of accuracy, compared to the mere 7 achieved by first differences. Even the single step showed great improvements over the non-extrapolated centered difference. But unfortunately, not even extrapolation can eliminate the problems of floating point errors. What it can do and has done here is eliminate truncation error due to "poor" initial approximation methods.

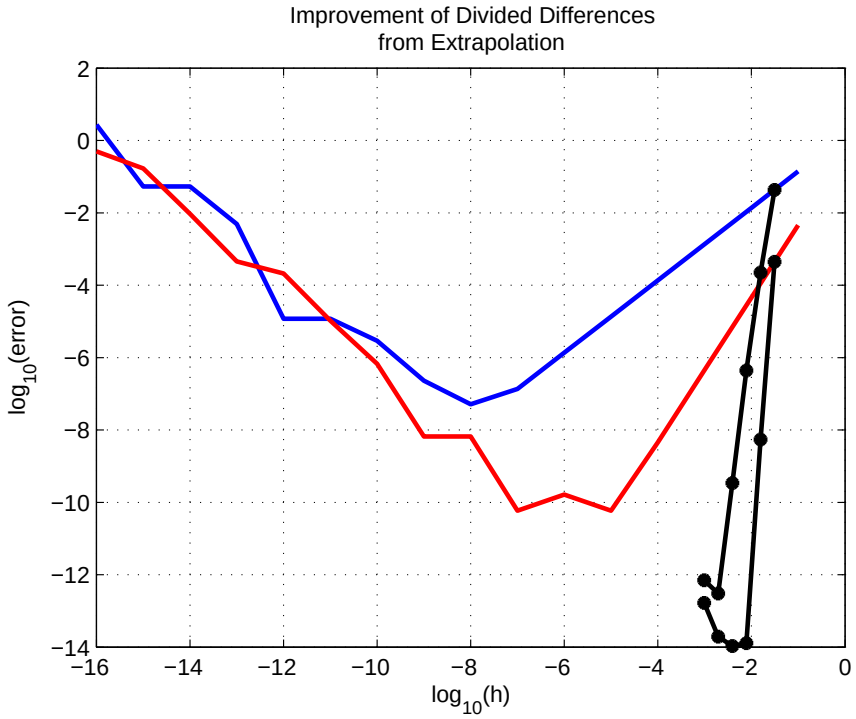

Figure 2: First and Extrapolated Centered Differences for $\frac{de^x}{dx}\ |_{x=1}$

Figure 3 displays the LHS, Trapezoid, Midpoint, and Simpson Rules in their standard forms (dotted lines). If it is not clear, LHS is the black line marked by asterisks, the midpoint/trapezoid rules virtually lie on top of each other, and the pink like marked by Xs is Simpson's Rule. Once extrapolated (solid lines), Simpson, Midpoint, and Trapezoid are nearly indistinguishable. (Why does this imply that we should never start with Simpson's Rule when extrapolating?) Even the lowly Left Hand Sum is now performing at a decent level. But clearly, the Midpoint or Trapezoid Rule is the best choice here. The deciding factor between them is typically where we want the function evaluations to occur (i.e. Midpoint avoids the endpoints).

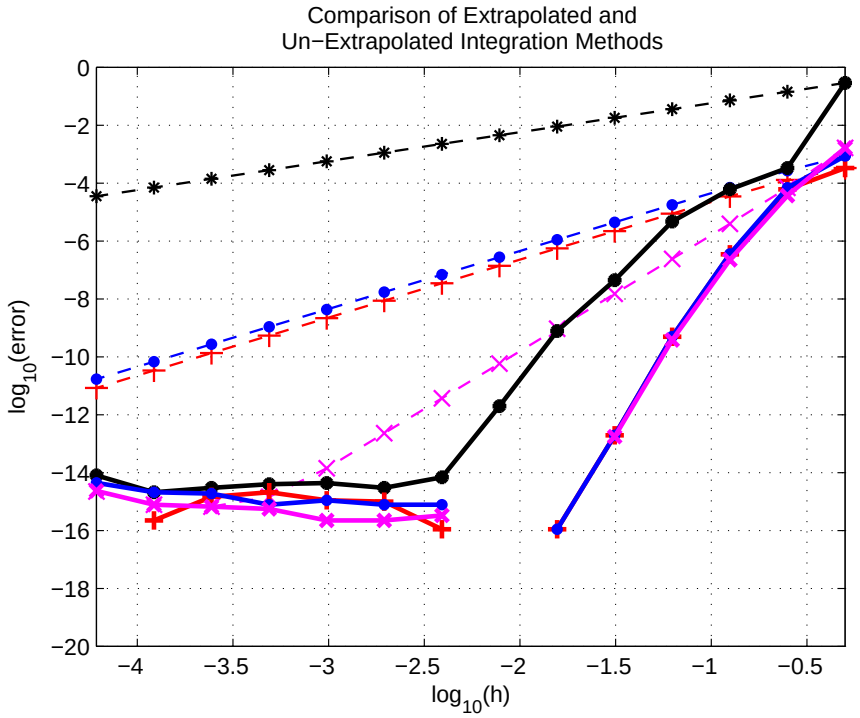

Figure 3: LHS, Trapezoid, Midpoint, and Simpson extrapolated and Un-Extrapolated

4.2 Finding Error Powers Graphically

Oftentimes, we find ourselves attempting to extrapolate expressions where the error powers have not been analytically pre-derived for our convenience. We have already seen a few examples of this; other situations include most "canned" methods for numerically solving ODEs. However not all cases are so nice. For example, it is easy to apply extrapolation to evaluate infinite series or sequences, but the error terms are usually not obvious in those cases. Differentiation and integration are special in that it is relatively easy to analytically show how to form the $\{p_i\}$. In these situations, we are left with two options. We can either attempt to analytically derive the error terms or we can use graphical methods.

This is another situation where using log-log plots is *ideal*. We begin with:

$$\begin{aligned} I(h) - I(\frac{h}{s}) &= (1 - s^{-p_1})h^{p_1}k_1 + (1 - s^{-p2})h^{p_2}k_1 + \dots \\ &= (1 - s^{-p_1})h^{p_1}k_1 + \mathrm{O}(h^{p_2}) \end{aligned}$$

From here, we would like to determine p_1. We can do this by first assuming that for sufficently small values of h (in practice, on the order of $\frac{1}{100}$ and smaller is sufficient), we can make the approximation $\mathrm{O}(h^{p_2}) \approx 0$. Now we apply $\log_{10}$ to both sides; note that we will now drop the subscript p_1 and assume that log means $\log_{10}$. So now we have:

$$\log(I(h) - I(\frac{h}{s})) = p\log(h) + \log(1 - s^{-p}) + logk_1 \tag{14}$$

At this point, we can taylor expand $\log(1 - s^{-p}) = \frac{1}{\ln(10}\left(-s^{-p} - \frac{1}{2}s^{-2p} - \frac{1}{3}s^{-p} + \dots\right)$. At this point we have two options. If we believe that the error powers will be integers or "simple" fractions (e.g. $\frac{1}{2}, \frac{1}{3}, \frac{1}{4}$ and similarly simple ones), then we can make the additional approximation

$\log(1 - s^{-p}) \approx \log(1) = 0$. As we will see, this series of approximations actually does not really harm our ability to determine p in most situations. The alternative is to retain more terms from the expansion of $\log(1 - s^{-p})$ and solve for p using the differences $\log(I(h) - I(\frac{h}{s}))$. Proceeding with the first option, we have:

$$\log(I(h) - I(\frac{h}{s})) = p \log(h) + C \tag{15}$$

Figure 4 begins with the Trapezoid Rule extrapolated once using the known first error term, $p_1 = 2$. The slope of this (red) line shows the next error power we would need to use in order to continue with the extrapolation process. Not surprisingly, the plot shows that $p_2 = 4$. Table 2 displays the successive points between step-sizes. As predicted, the slope values converged to approximately 4 fairly quickly. In the first few entries, the choice of s is too small, and in the last few values, it begins to diverge from 4 as floating point errors come into play. But in the center, there is little doubt that the correct choice is $p_2 = 4$, if you did not believe the plot.

The other lines in figure 4 demonstrate what happens as continue setting successive error powers. Here it is clear that the progression for the trapezoid rule goes in multiples of 2.

5 Conclusion

Extrapolation allows us to achieve greater precision with fewer function evaluations than un-extrapolated methods. In some cases (as we saw with simple difference schemes), extrapolation is the only way to improve numeric behavior using the same algorithm (i.e. without switching to centered differences or something better). Using sets of poor estimates with small step-sizes, we can easily and quickly eliminate error modes, reaching results that would be otherwise unobtainable without significantly smaller step-sizes and function evaluations. Thus we turn relatively poor numeric schemes like the Trapezoid Rule into excellent performers.

Of course the material presented here is certainly not the limit of extrapolation methods. The Richardson and Romberg techniques are part of a fundamental set of methods using polynomial approximations. There exist other methods using rational function approximations (e.g. Pade, Burlirsch Stoer) along with generalizations of material presented here. For example, Sidi gives detailed analysis of extrapolating functions whose expansions are more complicated than $A(h) = A + \sum_{k=1}^{s} \alpha_k h^{p_k} + \mathrm{O}(h^{p_{s+1}})$. The next order of complexity can solve problems where we can write $A(h) = A + \sum_{k=1}^{s} \alpha_k \phi_k(y) + \mathrm{O}(\phi_{s+1}(y))$ where $A(y)$ and $\phi_k(y)$ are assumed known, while again the α_k are not required.

Still, the methods presented here are more than sufficient for many applications including evaluating integrals, derivatives, infinite sums, infinite sequences, function approximations at a point (e.g. evaluating π or γ), and more.

5.1 From Trapezoid to Simpson and Beyond

On a closing note, we will now quickly outline how Romberg extrapolation can be used to easily derive Simpson's rule from the Trapezoid rule. More generally, extrapolating the $n = i$ Closed Newton-Cotes Formula results in the $n = i + 1$ version. By now, this should not be all that surprising. We start with $s = 2$ and $p = 2$; the choice of p is based on the error term from the Closed Newton-Cotes Formula. Alternatively, it could have been determined graphically, as shown earlier.

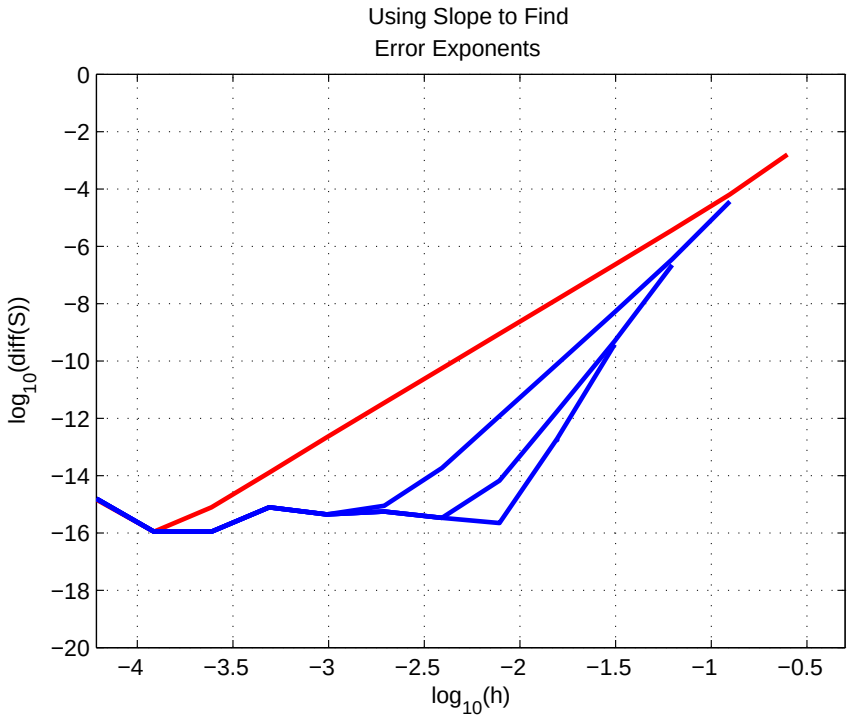

Figure 4: Trapezoid Rule applied to $\int_{0.25}^{1.25} \exp(-x^2)$

Table 2: Slope between step-sizes

4.61680926238264
4.12392496687081
4.02980316622319
4.00738318535940
4.00184180151224
4.00045117971785
4.00034982057854
3.99697103596297
4.08671822443851
4.02553509210714

$$\begin{aligned}\frac{4trap(\frac{h}{2}) - trap(h)}{2^2 - 1} &= Simp(\frac{h}{2}) \\ &= \frac{h}{3}\left(f(a) + 2\sum_{j=1}^{n-1} f(x_j) + f(b)\right) - \frac{h}{6}\left(f(a) + 2\sum_{j=1}^{n-1} f(x_j) + f(b)\right) \\ &= \frac{h}{3}\left(f(a) + 2\sum_{j=1}^{\frac{n}{2}-1} f(x_j) + 4\sum_{j=1}^{\frac{n}{2}} f(x_j) + f(b)\right)\end{aligned}$$

Thus Simpson's Rule is:

$$\text{Simpson's Rule: } \int_a^b f(x)\,dx = \frac{h}{3}\left(f(a) + 2\sum_{j=1}^{\frac{n}{2}-1} f(x_j) + 4\sum_{j=1}^{\frac{n}{2}} f(x_j) + f(b)\right) + \mathrm{O}(h^4) \quad (16)$$

References

[1] Burden, R. and D. Faires, *Numerical Analysis 8TH Ed*, Thomson, 2005.

[2] Issacson, E. and H. B. Keller. *Analysis of Numerical Methods*, John Wiley and Sons. New York, 1966.

[3] Press, et. al. *Numerical Recipes in C: The Art of Scientific Computing*, Cambridge University Press, 1992.

[4] Sidi, Avram, *Practical Extrapolation Methods*, Cambridge University Press, 2003.

[5] Stoer, J. and R. Burlirsch, *Introduction to Numerical Analysis*, Springer-Verlag, 1980.

[6] Sujit, Kirpekar, Implementation of the Burlirsch Stoer Extrapolation Method, U. Berkeley, 2003.