Stencils without Pencils

A short write up explaining how I treat spatial derivatives in one-dimensional systems of Partial Differential Equations (PDE) using finite difference approximations based on Taylor series expansions. The extension to multiple dimensions is straightforward but is a topic for another day.

I will:

- Describe discretization grids and how derivative approximation sets fit on the grid.
- Derive the finite difference stencils starting from Taylor expansions.
- Show how to calculate stencils in general and how to use Taylor Tables.
- Show how to apply stencils across a discretization grid.
- Briefly discuss challenges at the boundaries of a grid and the accuracy bonus when using centered stencils.

At the end of the paper is a simple Python program that calculates finite difference stencils of any size and shape avoiding tedious paper calculations - stencils without pencils.

I hope you enjoy it!

1. Discretization Grids and Approximation Sets

Many engineering problems require that we solve partial differential equations of the form:

$$\frac{\partial y}{\partial t} = f(y, y', y'', y''', \ldots, t, x)$$

where y', y'', etc. represent spatial derivatives of various orders.

A classic example is the one-dimensional heat equation:

$$\frac{\partial T}{\partial t} = \alpha \frac{\partial^2 T}{\partial x^2}$$

where T is temperature, t is time, x is the spatial coordinate, and α is thermal diffusivity.

Real world problems tend to be more complicated with first, second, or higher order spatial derivatives, nonlinear terms, multiple PDEs with coupling relationships, multiple dimensions, or a combination of PDE and ODE. Equations like these are found in fluid flow phenomena, in mixing problems, heat and mass transfer, the reaction kinetics and transport equations that describe catalytic chemical reactors… to name but a few.

The usual solution strategy is to discretize in the spatial domain and then approximate the spatial derivatives using the function values (y values) at a set of adjacent grid points that I call the approximation set.

Figure 1 is an illustration of a 1-dimensional discretization grid. At each grid point (x_i, x_{i+1}, etc) there is an associated function value, ($y(x_i)$, $y(x_{i+1})$, etc.) referred to in shorthand as y_i, y_{i-1}, etc. Notation becomes more complicated for multidimensional problems, but the extension is straightforward, and we need not worry about that now.

It's probably worth pointing out the initial conditions provided in our problem will give as an initiaol set of values for the y_i in our grid.

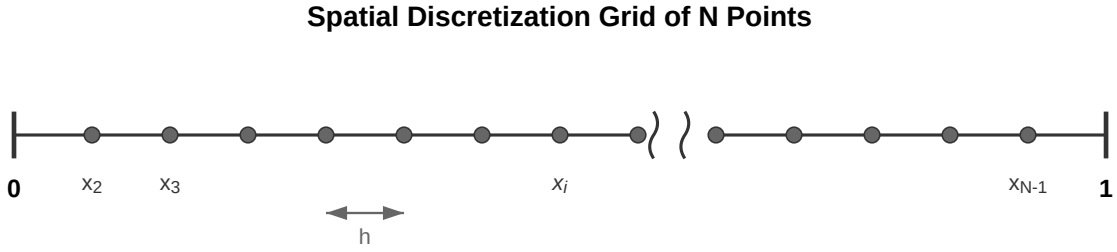

Figure 1: Illustration of a one-dimensional spatial discretization grid. The domain extends from x=0 to x=1 with uniform spacing, h, between adjacent grid points. The wavy lines indicate that the actual grid contains many more points. At each grid point there is an associated y value

Figure 2 is a 1-dimensional grid illustrating a 5-point centered approximation set. The spatial derivatives at point x_i are approximated by using the y-values at x_i plus two points upstream and two points downstream $(y_{i-2}, y_{i-1}, y_i, y_{i+1}, y_{i+2})$.

The number of points in the approximation set can vary (more points means more accuracy, as we will see) and the principle point, x_i (the point for which we are approximating derivatives), can be shifted to put more points before than after or vice-versa. In my notation the points in the approximation set are $[x_i]^m$ and the set of corresponding y values is $[y_i]^m$, where m is the number of points in the set.

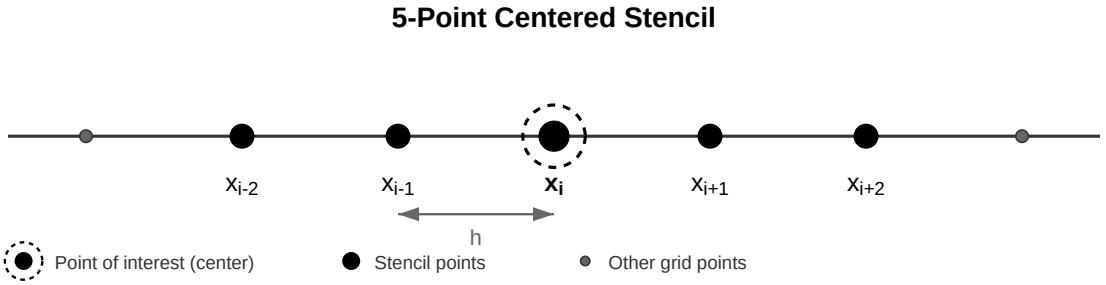

Figure 2: Illustration of a 5-point centered approximation set. The circled point is the location where the derivative is being approximated, using function values at the five highlighted points (two upstream, center, and two downstream). The spacing between adjacent grid points is h.

Figure 3 is a 3-point forward approximation.

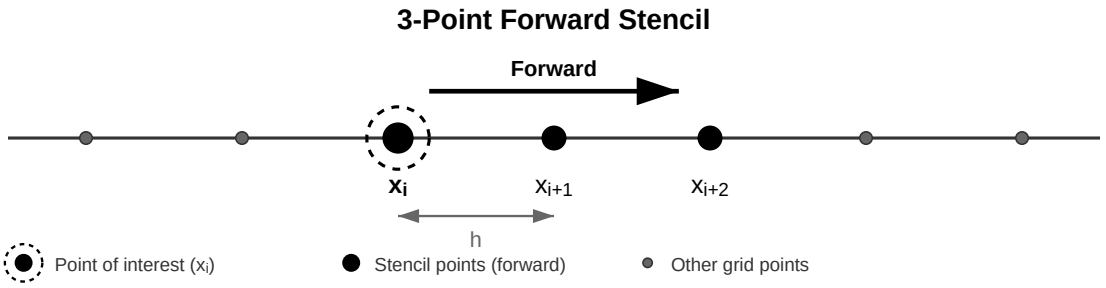

Figure 3: Illustration of a 3-point forward approximation set. The circled point is the location where the derivative is being approximated using function values at x_i plus the two adjacent points immediately downstream. The spacing between adjacent grid points is h.

Discretizing in the spatial domain simplifies the PDE to a set of Ordinary Differential equations (ODE) - one at each grid point - with derivatives only in time:

$$\frac{\partial y_i}{\partial t} = f(y_i, y_i', y_i'', y_i''', \ldots, t, x)$$

but, because the spatial derivatives are approximated using $[y]^m$:

$$\frac{dy_i}{dt} = f([y_i]^m, t, x)$$

and $[y^i]^m$ are the function values at the points in the approximation set.

This system of ODE can be solved using an appropriate numerical ODE solver of which there are many.

Some problems include both PDE (usually time and spatial derivatives) and ODE (usually only spatial derivative). Discretization converts the PDE to ODE and the ODE to algebraic equations. An ODE after discretization becomes an algebraic equation of the form.

$$0 = f([y_i]^m, t, x)$$

Simultaneously solving differential/algebraic equation sets (DAE) requires a more specialized numerical routine, but they are readily available, if not as numerous as ODE solvers.

The next section shows how spatial derivative approximations arise from Taylor expansions and introduces stencils.

2. Stencils arise from Taylor Series Expansions

From calculus, we know that $y(x + a)$ can be approximated using the function and derivatives values at x, as shown below:

Forward expansion (Estimating $y(x+h)$ from the $y^{(n)}(x)$):

$$y(x+h) = y(x) + hy'(x) + \frac{h^2}{2!}y''(x) + \frac{h^3}{3!}y'''(x) + \frac{h^4}{4!}y^{(4)}(x) + \cdots$$

Backward expansion (for $x-h$):

$$y(x-h) = y(x) - hy'(x) + \frac{h^2}{2!}y''(x) - \frac{h^3}{3!}y'''(x) + \frac{h^4}{4!}y^{(4)}(x) - \cdots$$

Multi-step expansions:

$$y(x+2h) = y(x) + 2hy'(x) + \frac{(2h)^2}{2!}y''(x) + \frac{(2h)^3}{3!}y'''(x) + \cdots$$

$$y(x-2h) = y(x) - 2hy'(x) + \frac{(2h)^2}{2!}y''(x) - \frac{(2h)^3}{3!}y'''(x) + \cdots$$

(trivial) Zero step expansion

$$y(x) = y(x)$$

And we can treat the function value as the zero-th derivative, $y(x) = y^{(0)}(x)$

Notice that for positive steps (h and $2h$) the expansion terms are all positive, but for negative steps (-1 and $-2h$), the terms alternate sign - odd powers of negative numbers are negative. These five equations are the taylor expansions we would use for a 5-point centered approximation.

To approximate spacial derivatives using an arbitrary approximation set of m points, we start by writing Taylor expansions for the function value at each point. Next, (and this is the trick) we find linear combinations of the expansions that cancel out all but one of the derivatives of order $m-1$ and lower. The remaining derivative is the one we want to approximate.

Strictly speaking, Taylor expansions are infinite series but we will truncate the expansions after the $y^{(m)}$ term. The remaining terms, collectively, are the truncation error, which we say is $\mathcal{O}(h^{m+1})$ - a shorthand way of saying error is loosely proportional to h^{m+1}. This truncation error propagates to our derivative approximations and is important in deciding what kinds of approximation sets to use for a desired accuracy.

I think an example is the best way to understand this, so let's look at derivative approximations using 3-point centered and 5-point centered sets. Along the way I will introduce stencils - the vector of coefficients we need to estimate derivatives along our discretization grid:

3-point centered stencils

A three-point centered approximation set uses points at $[x_{i-1},\ x_i,\ x_{i+1}]$, and the Taylor expansions for the function values at those points is our starting place:

$$y(x_i + h) = y_i + hy_i' + \frac{h^2}{2}y_i'' + \frac{h^3}{6}y_i''' + \mathcal{O}(h^4)$$

$$y(x_i) = y_i$$

$$y(x_i - h) = y_i - hy_i' + \frac{h^2}{2}y_i'' - \frac{h^3}{6}y_i''' + \mathcal{O}(h^4)$$

A 3-equation system is simple enough that we can see by inspection the linear combination we want - multiply the expansions by $-\frac{1}{2}$, 0, $\frac{1}{2}$ respectively:

$$-\frac{1}{2} \cdot \left[y(x_i - h) = -\frac{1}{2}y_i + \frac{h}{2}y_i' - \frac{h^2}{4}y_i'' + \frac{h^3}{12}y_i''' + \mathcal{O}(h^4) \right]$$

$$0 \cdot y(x_i) = 0$$

$$+\frac{1}{2} \cdot \left[y(x_i + h) = +\frac{1}{2}y_i + \frac{h}{2}y_i' + \frac{h^2}{4}y_i'' + \frac{h^3}{12}y_i''' + \mathcal{O}(h^4) \right]$$

Adding these up cancels the y_i and y_i'' terms, leaving only the y' term:

$$\frac{1}{2}\left[y_{i+1} - y_{i-1}\right] = \left[-\frac{1}{2} + \frac{1}{2}\right] y_i + \left[\frac{h}{2} + \frac{h}{2}\right] y_i' + \left[-\frac{h^2}{4} + \frac{h^2}{4}\right] y_i'' + \left[\frac{h^3}{12} + \frac{h^3}{12}\right] y_i'''$$

$$\frac{y_{i+1} - y_{i-1}}{2} = hy_i' + \frac{h^3}{6}y_i''' + \mathcal{O}(h^4)$$

Dividing by h and rearranging gives centered difference formula for y':

$$y_i' \approx \frac{y_{i+1} - y_{i-1}}{2h} + \mathcal{O}(h^2)$$

Notice that the $y^{(m)}$ (here $m = 3$) term did not cancel. So, after dividing through by h, we find that the truncation error has propagated to be $\mathcal{O}(h^2)$ for our first derivative approximation.

The set of coefficients $[-\frac{1}{2}, 0, \frac{1}{2}]$, is the stencil, so for this example:

$$y_i' = \frac{1}{h}\left[-\frac{1}{2}y_{i-1} + 0y_i + \frac{1}{2}y_{i+1}\right]$$

When estimating derivatives about point x_i, the stencil coefficients applied to function values at points behind x_i have a negative index and the coefficients for points ahead have a positive index - so the coefficient applied to the value at x_{i-3} is c_{-3}, and at x_i is c_0, and so on. The stencil coefficients for an m-point approximation I will represent as $[c_i]^m$, or in expanded form for a 3-point centered stencil $[c_{-1}, c_0, c_1]$. For a 5-point centered stencil, the stencil coefficients are applied as:

$$c_{-2}y_{i-2} + c_{-1}y_{i-1} + c_0y_i + c_1y_{i+1} + c_2y_{i+2}$$

or

$$\sum_{s=a}^{b} c_s y_{i+s}$$

Where a and b are the starting and ending values of the stencil index s.

In linear algebra notation:

$$\mathbf{c} \cdot \mathbf{y}_i$$

which is the dot product of the coefficient vector $\mathbf{c}$ and the vector $\mathbf{y}$ that contains the function values at the $[x_i]^m$ points in the approximation set positioned at x_i. The dot product yields a scalar and the approximation for $y_i^{(k)}$ would be

$$y^{(k)} = \frac{1}{h^k}]\mathbf{c} \cdot \mathbf{y}_i]$$

Returning to our examples, if we want a 3-point centered approximation for the second derivative, we multiply the expansions by $1, -2$ and 1, respectively (so the stencil vector $\mathbf{c}$, is [1, −2, 1]).

$$1 \cdot y(x_i - h) = y_i - hy_i' + \frac{h^2}{2} y_i'' - \frac{h^3}{6} y_i''' + \mathcal{O}(h^4)$$

$$-2 \cdot y(x_i) = -2y_i$$

$$+1 \cdot y(x_i + h) = y_i + hy_i' + \frac{h^2}{2} y_i'' + \frac{h^3}{6} y_i''' + \mathcal{O}(h^4)$$

Adding cancels the y and y' terms:

$$y_{i-1} - 2y_i + y_{i+1} = h^2 y_i'' + \mathcal{O}(h^4)$$

And this time the y''' term has canceled

Dividing by h^2 and rearranging:

$$y''(x_i) \approx \frac{y_{i-1} - 2y_i + y_{i+1}}{h^2} + \mathcal{O}(h^2)$$

Both approximations from our 3-point centered stencil are second-order accurate, $\mathcal{O}(h^2)$.

Hopefully, the reader can anticipate these two features

- Increasing the number of points in our discretization grid improves accuracy because h (the distance between grid points) is inversely proportional to N (the number of grid points) and a smaller h means a smaller truncation error.

- Increasing the number of points in the approximation set improves accuracy because more points give us more equations (more Tayler expansions) to eliminate even higher order terms leaving an even higher exponent applied to h in $\mathcal{O}(h^e)$.

5-point centered stencils

For a y' approximation using a 5-point centered stencil, We start with Taylor expansions about all the points in the approximation set, then find stencil coefficients that cancel y_i, y_i'', y''', and y_i^{iv}. I leave the algebra to the reader; the stencil is $[\frac{1}{12}, -\frac{2}{3}, 0, \frac{2}{3}, -\frac{1}{12}]$; the Taylor expansions with stencil coefficients applied is:

$$\frac{1}{12} \cdot y(x_i - 2h) = \frac{1}{12}y_i - \frac{h}{6}y_i' + \frac{h^2}{3}y_i'' - \frac{2h^3}{9}y_i''' + \frac{2h^4}{9}y_i^{(4)} - \frac{4h^5}{45}y_i^{(5)} + \mathcal{O}(h^6)$$

$$-\frac{2}{3} \cdot y(x_i - h) = -\frac{2}{3}y_i + \frac{2h}{3}y_i' - \frac{h^2}{3}y_i'' + \frac{h^3}{9}y_i''' - \frac{h^4}{36}y_i^{(4)} + \frac{h^5}{180}y_i^{(5)} + \mathcal{O}(h^6)$$

$$0 \cdot y(x_i) = 0$$

$$\frac{2}{3} \cdot y(x_i + h) = \frac{2}{3}y_i + \frac{2h}{3}y_i' + \frac{h^2}{3}y_i'' + \frac{h^3}{9}y_i''' + \frac{h^4}{36}y_i^{(4)} + \frac{h^5}{180}y_i^{(5)} + \mathcal{O}(h^6)$$

$$-\frac{1}{12} \cdot y(x_i + 2h) = -\frac{1}{12}y_i - \frac{h}{6}y_i' - \frac{h^2}{3}y_i'' - \frac{2h^3}{9}y_i''' - \frac{2h^4}{9}y_i^{(4)} - \frac{4h^5}{45}y_i^{(5)} + \mathcal{O}(h^6)$$

Adding these, cancels the y_i, y_i'', y''', and y_i^{iv} terms leaving:

$$\frac{1}{12}y_{i-2} - \frac{2}{3}y_{i-1} + \frac{2}{3}y_{i+1} - \frac{1}{12}y_{i+2} = hy_i' - \frac{h^5}{30}y_i^{(5)} + \mathcal{O}(h^6)$$

Dividing by h and rearranging:

$$y'(x_i) \approx \frac{y_{i-2} - 8y_{i-1} + 8y_{i+1} - y_{i+2}}{12h} + \mathcal{O}(h^4)$$

The y^m term did not cancel so our approximation is fourth-order accurate, $\mathcal{O}(h^4)$, a significant improvement over our $\mathcal{O}(h^2)$ three-point approximation. In general, the approximation error will be $\mathcal{O}(h^{m-k})$ where m is the number of points in the approximation set and k is the order of the derivative we want to approximate. Under certain conditions, the $y^{(m)}$ term will cancel and we get a bonus accuracy improvement to h^{m-k+1}

I can generalize the equation for our derivative approximation at x_i to:

$$y_i^{(k)} \approx \frac{1}{h^k}\sum_{s=a}^{b} c_s y_{(i+s)}$$

where the $[y_i]^m$ are the values at the points in our approximating set, k is the order of the approximated derivative, a and b are the starting and ending stencil index values for our m point approximation set. The index interval [a,b] always includes 0 - c_0 is applied at x_i - and $(a-b) = (m-1).

3. Stencil Coefficients and Taylor Series Tables

In this section I'll develop the technique for approximating derivatives of arbitrary order (though, most engineering problems only need the first two) using an arbitrary number of points, which allows us arbitrary accuracy. Then I'll introduce Taylor Series Tables which are a handy tool for constructing the equation set we solve to find the stencil coefficients.

Finding stencils

In general, the Taylor expansion about any point $y(x + sh)$ is:

$$y(x+sh) = y(x) + sh\,y'(x) + \frac{(sh)^2}{2!}y''(x) + \frac{(sh)^3}{3!}y'''(x) + \cdots = \sum_{n=0}^{\infty}\frac{(sh)^n}{n!}y^{(n)}(x)$$

where h is the distance between points in our spatial grid and s is the index telling us the number of steps ahead (or behind) of x_i.

The form of the approximation we want for the k^{th} derivative at point x_i using m points is:

$$y^{(k)}(x_i) = y_i^{(k)} \approx \frac{1}{h^k}\sum_{s=a}^{b} c_s y_{(i+s)}$$

Inside the summation, for each of the y values, we can substitute the taylor series expansion - think of this as a compact way of writing the taylor expansions for all the points in our approximation set multiplied by the yet to be determined stencil coefficients c_j:

$$\sum_{s=a}^{b} c_s y_{i+s} = \sum_{s=a}^{b} c_s\left[y_i + shy_i' + \frac{(sh)^2}{2!}y_i'' + \frac{(sh)^3}{3!}y_i''' \cdots\right]$$

Next, we collect terms that are factors against the same derivative - think of this as adding the taylor expansions like we did to cancel terms in the previous examples, but this time the stencil coefficients, $[c_s]^m$, aren't yet known. So we re-write the equation as:

$$\sum_{s=a}^{b} c_s y_{i+s} = y_i\sum_{s=a}^{b} c_s + hy_i'\sum_{s=a}^{b} sc_s + \frac{h^2}{2!}y''\sum_{s=a}^{b} s^2c_s + \frac{h^3}{3!}y'''\sum_{s=a}^{b} s^3c_s + \cdots$$

To get the stencil coefficients for approximating $y^{(k)}$, we want the sum $\sum_{s=a}^{b} s^k c_s$ to equal $k!$ (to cancel out the $k!$ in the denominator) and we want the summation for all other derivatives through $y^{(m-1)}$ to be zero. So, for a second derivative approximation, the non-zero summation would be:

$$\sum_{s=a}^{b} s^2 c_s = 2!$$

which is just 2. For a first derivative it would be:

$$\sum_{s=a}^{b} s c_s = 1$$

For a third derivative it would be:

$$\sum_{s=a}^{b} s^3 c_s = 3! = 6$$

So, in general, when estimating the k^{th} derivative:

$$for\, n \;=\; 0 \;\; to \;\; m-1 \quad \sum_{s=a}^{b} j^n c_s = \begin{cases} n! & \text{if } n = k \\ 0 & \text{if } n \neq k \end{cases}$$

We can rewrite equations for the coefficients in matrix notation as

$$\mathbf{A}\mathbf{c} = \mathbf{r}$$

where:

$$\mathbf{A} = \begin{bmatrix} 1 & 1 & 1 & \cdots & 1 \\ a & a+1 & a+2 & \cdots & b \\ a^2 & (a+1)^2 & (a+2)^2 & \cdots & b^2 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ a^{m-1} & (a+1)^{m-1} & (a+2)^{m-1} & \cdots & b^{m-1} \end{bmatrix}, \quad \mathbf{c} = \begin{bmatrix} c_1 \\ c_2 \\ c_3 \\ \vdots \\ c_m \end{bmatrix}, \quad \mathbf{r} = \begin{bmatrix} 0 \\ 0 \\ \vdots \\ k! \\ \vdots \\ 0 \end{bmatrix}$$

And this is the linear equation set we solve to find the stencil coefficients.

The matrix $\mathbf{A}$ is a Vandermonde matrix where each column is a geometric progression of the coefficient index, and the coefficient index ranges from a to b, $\mathbf{c}$ is the vector of stencil coefficients we want to find, and the vector $\mathbf{r}$ has all zeros except for $k!$ in the $(k+1)^{th}$ position.

I hope this formulation brings out that for an m point approximation set, we get m equations and we can impose conditions on the coefficient summations for $y^0 - y^{(m-1)}$ only - we can't control the fate of $\$y\hat{}(\{m\})$, but it sometimes serendipitously cancels.

The vandermonde matrix is, apparently, important for polynomial interpolation and the reader can google to learn more about its origins. Now would be a good time for the reader to make sure that all the procedure - from writing Taylor series expansions, to the compact summation notation and collecting the terms, to the equation set we solve for the stencil coefficients - is clear.

Taylor tables

A Taylor table is a handy tool for making the Vandermonde matrix, **A**. Figure 4 is a Taylor table for a 5 point centered y'' approximation.

	$y(x)$	$y'(x)\cdot(sh)$	$y''(x)\cdot(sh)^2/2!$	$y'''(x)\cdot(sh)^3/3!$	$y^{(4)}(x)\cdot(sh)^4/4!$
$c_{-2}\cdot y(x-2h)$	c_{-2}	$-2c_{-2}$	$4c_{-2}$	$-8c_{-2}$	$16c_{-2}$
$c_{-1}\cdot y(x-h)$	c_{-1}	$-c_{-1}$	c_{-1}	$-c_{-1}$	c_{-1}
$c_0\cdot y(x)$	c_0	0	0	0	0
$c_1\cdot y(x+h)$	c_1	c_1	c_1	c_1	c_1
$c_2\cdot y(x+2h)$	c_2	$2c_2$	$4c_2$	$8c_2$	$16c_2$
Σ above terms	0	0	2!	0	0

Figure 4: A Taylor table for a 5-point centered stencil calculation. The table can be expanded to include more points for more accuracy. The location of x_i can shift to have more points before or after. Once you are familiar with the pattern, these are very easy to layout.

The procedure to create the table is as follows:

- In the first column, write all the points in the approximation set (x_i doesn't have to be centered).
- In the top row, write the Taylor series term to which the stencil coefficients apply starting with $y(x)$.
- Populate the table so that each row has the $c_s s^n$ terms where s runs from a to b.
- In the bottom row, write the column summations we need according to the derivative we want to approximate.

The bottom row is the vector **r**. To approximate y_i' we need the second column to add to 1 with zeros elsewhere; to approximate y_i'' we need the third column to add to 2! with zeros elsewhere; and so on. The example in Figure 4 is for estimating y'').

The multipliers next to the stencil coefficients in each row are the elements in the columns of the Vandermonde matrix **A** from ealier. For the $f(x-2h)$ row, these would be $[1, -2, 4, -8, 16, -32]$.

In Figure 5 I have created the Vandermonde matrix by isolating the multipiers in the Taylor table and then taking the transpose so that the rows become columns. This is the Matrix **A** in the linear equation set we solve to find the stencil coefficients, **c**.

$$\mathbf{Ac} = \mathbf{r}$$

$\partial^2 f/\partial x^2$	c_{-2}	c_{-1}	c_0	c_1	c_2
$f(x)$	1	1	1	1	1
$f'(x)\cdot(sh)$	-2	-1	0	1	2
$f''(x)\cdot(sh)^2/2!$	4	1	0	1	4
$f'''(x)\cdot(sh)^3/3!$	-8	-1	0	1	8
$f^{(4)}(x)\cdot(sh)^4/4!$	16	1	0	1	16

Figure 5: The multipliers have been isolated and the entries transposed relative to our original Taylor table. Taylor tables are a handy tool for creating the Vandermode matrix $\mathbf{A}$*, and for visualizing the matrix multiplication we solve for the stencil coefficients.*

I hope the reader can easily see how the table changes for larger approximation sets and for uncentered stencils. Once you know how to lay these out, you can very quickly construct the Vandermonde matrix and then solve the linear equation set for the stencil coefficients using any of many readily available matrix calculation tools.

Now that we know how to find the stencil coefficients for any approximation set, in the next section I'll show the linear algebra techniques for calculating derivatives across our discretization grid using our stencils. It turns out that the sparse nature of these system can be leveraged to accelerate our numerical calculations.

4. Calculating Derivatives Across a Grid

It is worthwhile to visualize the approximation calculation in the context of a large grid. So, imagine a large 1 dimensional grid of 1000 points wherein the spatial derivative at each point is a function of just a few adjacent points and then imagine how the approximation set moves with us as we go from point to point across the grid estimating derivatives.

I want the reader to realize:

- As we move across the grid, the derivatives at the current point, x_i, do not depend on most of the other grid points - only 5 out of 1000, for a 5-point stencil on a 1000 point grid. When we formulate this set of equations as matrix multiplication, we can intuitively expect that matrix to be sparse.
- As we approach the boundary (the edge of our grid), we run out of points. For an N point grid, we can't estimate the derivative at point x_N using points beyond N, like x_{N+1}, because they don't exist! They are outside the domain of our problem.

Matrix Formulation of the Derivative Approximation Calculation

The general equation for approximating derivatives at x_i using our stencil is:

$$y^{(k)}(x_i) \approx \frac{1}{h^k} \sum_{s=a}^{b} c_s y_{i+s}$$

For a first derivative approximation using a 5-point centered stencil:

$$y'(x_i) \approx \frac{1}{h} \left[c'_{-2} y_{i-2} + c'_{-1} y_{i-1} + c'_0 y_i + c'_1 y_{i+1} + c'_2 y_{i+2} \right]$$

and for a second derivative approximation:

$$y''(x_i) \approx \frac{1}{h^2} \left[c''_{-2} y_{i-2} + c''_{-1} y_{i-1} + c''_0 y_i + c''_1 y_{i+1} + c''_2 y_{i+2} \right]$$

The first and second derivative stencil coefficients are different (I've denoted them as c' and c''), but we use the same points $[y_i]^m$ for approximating both derivatives at x_i.

To compute derivatives across the entire grid we do this calculation at each grid point and we can write that sequence of calculations as:

$$y_i^{(k)} \approx \frac{1}{h^k} \sum_{s=a}^{b} c_s^k y_{i+s} = \frac{1}{h^k} \mathbf{D}^k \mathbf{y}, \quad i = 1, \cdots, N$$

where $\mathbf{y} = [y_1, y_2, \ldots, y_N]$ is the vector of function values at the N grid points and $\mathbf{D}$ is a matrix where each row contains a properly positioned stencil - row i contains the stencil for approximating the derivatives at x_i, row $i+1$ for derivatives at point x_{i+1}, and so on. We can use $\mathbf{D}'$, $\mathbf{D}''$, $\mathbf{D}^{(k)}$ to specify first, second, k^{th} derivative matrices, respectively.

For a 3-point centered stencil with coefficients $[c_{-1}, c_0, c_1]$ for interior grid points (special conditions apply at the boundaries), the derivative matrix has the form:

$$\mathbf{D} = \begin{bmatrix} c_{-1} & c_0 & c_1 & 0 & 0 & \cdots & 0 \\ 0 & c_{-1} & c_0 & c_1 & 0 & \cdots & 0 \\ 0 & 0 & c_{-1} & c_0 & c_1 & \cdots & 0 \\ \vdots & \vdots & \vdots & \ddots & \ddots & \ddots & \vdots \\ 0 & 0 & 0 & \cdots & c_{-1} & c_0 & c_1 \end{bmatrix}$$

Ignoring for the moment the difficulties at the boundary, the matrix $\mathbf{D}$ is size $N \times N$. Each row represents the equation for one derivative approximation and each of the grid points in discretization requires one column. If we were to write out $\mathbf{D}$ for a 1000 point grid, it would extend far beyond the edges of this page (and a few more pages as well), yet each row would have only 5 non-zero entries. This is a very sparse pentadiagonal matrix, and that works to our advantage because we can accelerate the numerical calculation by using methods tailored for sparse systems.

As alluded to earlier, there are difficulties near the boundaries where the points we need for the approximation calculation lie outside the domain of our problem - we can't use points that come before x_0 or after x_N because they they don't exist!. There are several ways to handle boundary issues and I will very briefly describe one of the most common.

Ghost Points

Ghost points are fictitious points that lie outside the physical domain of our problem. They allow us to use our stencil, unmodified, to approximate derivatives for points near the domain boundary. Ghost point values are chosen to satisfy the specific boundary conditions (or coupling relationships) of the problem.

For centered approximations using $2k+1$ points, k points are tacked on to each side of our grid, so we have k points before x_0 and k points after x_N. When we include the ghost points the matrix $\mathbf{D^{(k)}}$ becomes a $(N+2k) \times N$ system calculating derivatives at N points using $N+2k$ points and we would write:

$$\mathbf{y}^{(k)} \approx \frac{1}{h}\hat{\mathbf{D}}^{(\mathbf{k})}\hat{\mathbf{y}}$$

Where $\hat{\mathbf{D}}$ and $\hat{\mathbf{y}}$ indicates they have been modified with ghost points to accommodate calculations near the boundary.

The Accuracy Boost for Centered Stencils

When we looked at the example systems for finding stencils, I hope the reader noticed that sometimes the y^{(m)}$ term canceled, and sometimes it didn't. It turns out that when approximating even numbered derivatives using centered stencils the $y^{(m)}$ term cancels and we get an accuracy boost. But, when approximating odd numbered derivatives it does not cancel.

To see why this is so, remember that centered stencils always have an odd number of points. It turns out that when centered stesncils are used to approximate even numbered derivatives, the coefficients are symmetric around the center point, and wWhen used to approximate odd numbered derivatives, the coefficients are symmetrical in absolute value, but opposite in sign. Fpr example, the 7-point centered stencil for y'' is

$$[\frac{1}{90},\ -\frac{3}{20},\ \frac{3}{2},\ -\frac{49}{45},\ \frac{3}{2},\ -\frac{3}{20},\ \frac{1}{90}]$$

and the stencil for y' is

$$[-\frac{1}{60},\ \frac{3}{20},\ -\frac{3}{4},\ 0,\ \frac{3}{4},\ -\frac{3}{20},\ \frac{1}{60}]$$

If we look again at Figure 5 we see that the row elements in the Vandermonde matrix are also symmetrical for even numbered derivatives and symmetrical in absolute value but opposite in sign for odd numbered derivative. I hope the reader can see the implication.

When using centered stencils to approximate even order derivatives, all of the odd order derivatives cancel, which gives us an accuracy boost by canceling out the $y^{(m)}$ term. when using centered stencils to estimate odd order derivatives, all of the even ordered derivatives cancel, but that doesn't help us because the $y^{(m)}$ term remains.

If this isn't yet clear, imagine the matrix multiplication of a symmetric stencil against the row entries in Figure 5.

We don't get bonus accuracy using non-centered stencils. If we want the same error for approximatinig y_i' and y_i'', we can use an $[x_i]^m$ non-centered approximation set for y_i', but we have to add a point and use $[x_i]^{(m+1)} set for y_i''

Attached at the end of this article is a simple python program that will calculate stencils for an arbitrary number of points and an arbitrary configuration. It will also calculate the m^{th} derivative multiplier. I encourage the reader to play with different kinds of stencils to see how this plays out.

In summary

The objective of this article was to introduce the discretization and approximating techniques used to solve the system of PDE characteristic of many engineering problems. What I hope we learned:

- Why we convert spatial dimensions to a grids of N points.
- What is an approximation set and how does that fit (and move) on a grid
- How stencil coefficients arise from Taylor expansions
- How the number of points in the approximation set affects the approximation error
- How the number of points in the grid affects the approximation error
- How to calculate stencil coefficients for arbitrary approximation sets and to arbitrary accuracy
- How stencil coefficients are placed in a derivative matrix to approximate derivatives across the entire grid
- The bonus accuracy of centered stencil approximations
- An awareness of difficulties at the edges of our grid and ghost points

If the reader can take these learnings forward, then our time together was well spent.

'Till next time,

Mike

Appendix: Python Program for Calculating Stencils

(with sample output)

Below is the afore mentioned python code. It requires that the user manually update three entries: a, b, and k where a and b are the starting and ending indices for an m point approximation set, $[x_i]^m$ (that includes x_o), and k is the order of the derivative we wish to approximate (and must be m-1 or lower).

The output is fairly self explanatory, but it will repeat the input, show the Vandermonde matrix, $\mathbf{A}$, and the transpose $\mathbf{A^T}$. The vector $\mathbf{b}$ and the stencil vector $\mathbf{c}$. It performs a consistency check to ensure the reverse calculation reproduces $\mathbf{b}$. Finally, it checks to see if the m^{th} derivative term cancels by calculating the summation below (which will be zero if $y^{(m)}$ cancels, otherwise it does not)

$$\sum_{s=a}^{b} s^m c_s$$

```
import numpy as np
from scipy.linalg import solve
import math
from math import factorial


# program to calculate stencils
# for an arbitrary number of points
# in an arbitrary configuration
# via the Vandermonde matrix
# using np.vander
#=================================================================
#=================================================================
# the user entries a and b that define the approximation set.
# a is the rear index and b is the front index of the stencil
# for a 3-point centered stencil: a = -1, b = 1
# for a 5-point forward stencil: a = 0, b = 4
# k is the order of the derivative we want 1 for first, 2 for second....
#-----------------------------------------------------------
a = -3  #<<<<<<<<<<<<
b = 3 #<<<<<<<<<<<<
k = 2   #<<<<<<<<<<<<

n = (b-a) + 1

if k >= n:
    print(f"approximation set too small to the {k}th derivative \n\n)")

#  repeating the input
print(f"stencil indices are a:{a}, and b: {b}\n\n")
print(f"there are {n} points in the approximation set \n\n")
print(f"we want to find the {k}th derivative \n\n")
#=================================================================

#=================================================================
#---------------Calculating the Vandermonde----------------
indices = np.arange(a, b + 1)
A = np.vander(indices, N=len(indices), increasing=True)
print(f"The Vandermonde Matrix, A, is \n {A} \n\n")

#=================================================================
#----------- creating vector b - we place k! in position k+1 --
B = np.zeros(n)
B[k] = math.factorial(k)
print(f"the b vector is: {B} \n\n")

#=================================================================
```

```
#----------  take transpose of A---------
A_T = np.transpose(A)
print(f"A transpose is: \n {A_T} \n\n")

#=========================================================
# ----------solve for the stencil coefficients ----------
x = solve(A_T,B)

#=========================================================
#--------- print output --------------------------
print(f"the stencil coefficients are \n {[f'{val:.4f}' for val in x]} \n\n")

#=============================================================================
# ---------- check by reversing the calc to reproduce b --------------------
check = np.dot(A_T,x)
print(f"this check should reproduce the b vector --> {[f'{val:.2f}' for val in check]} \n\n")

#=====================================================
#  Did we eliminate the mth derivative term

app_set = np.arange(a, b+1)
m_vec = app_set**n
m_check = np.dot(m_vec, x)

print(f" the mth derivative term cancels (we get bonus accuracy) if this sum is ~0:    {m_chec

#   END



#================================
#Sample output
#================================

stencil indices are a:-3, and b: 3

there are 7 points in the approximation set

we want to find the 2th derivative

The Vandermonde Matrix, A, is

 [[   1   -3    9  -27   81 -243  729]
 [   1   -2    4   -8   16  -32   64]
 [   1   -1    1   -1    1   -1    1]
 [   1    0    0    0    0    0    0]
 [   1    1    1    1    1    1    1]
 [   1    2    4    8   16   32   64]
```

```
 [   1    3    9   27   81  243  729]]


the b vector is: [0. 0. 2. 0. 0. 0. 0.]


A transpose is:

 [[   1    1    1    1    1    1    1]
 [  -3   -2   -1    0    1    2    3]
 [   9    4    1    0    1    4    9]
 [ -27   -8   -1    0    1    8   27]
 [  81   16    1    0    1   16   81]
 [-243  -32   -1    0    1   32  243]
 [ 729   64    1    0    1   64  729]]


the stencil coefficients are
 ['0.0111', '-0.1500', '1.5000', '-2.7222', '1.5000', '-0.1500', '0.0111']


this check should reproduce the b vector --> ['-0.00', '0.00', '2.00', '-0.00', '0.00', '-0.00

 the mth derivative term cancels (bonus accuracy) if this sum is ~0:    0.0000
```