

Computing the Gradient: It's All Uphill From Here (Atmo 336, Fall 2007)

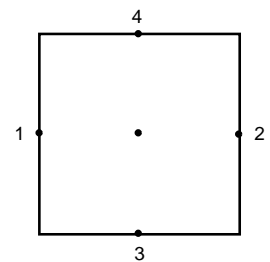
Getting Started. In a previous lab (see *Integrating the Hypsometric Equation—The Trapezoidal Rule*) we introduced a simple method for computing an integral when all you know is a discrete set function values at a finite number of points. Here we do the opposite—that is, instead of computing an integral we'll compute a derivative. And for our application we consider this: we suppose that we are given a gridded set of pressure measurements and we want to compute the gradient of the pressure from our data.

The Method. We've actually been computing derivatives from a gridded dataset already—in our discussion of kinematics. To be specific, we've been estimating the derivatives of the velocity components using approximations such as $\partial u/\partial x \approx (u_2 - u_1)/\Delta x$, $\partial u/\partial y \approx (u_4 - u_3)/\Delta y$, etc. Here we want to do the same thing....we just want to be a bit more careful about it.

$$\frac{\partial u}{\partial x} \approx \frac{u_2 - u_1}{\Delta x}$$

$$\frac{\partial u}{\partial y} \approx \frac{u_4 - u_3}{\Delta y}$$

etc.



A 1D example. Suppose we have a function $f(x)$ whose value is known at a set of discrete grid points

$$\dots, x_i - 2\Delta x, x_i - \Delta x, x_i, x_i + \Delta x, x_i + 2\Delta x, \dots$$

as illustrated below. We then want to approximate df/dx using only the known values of f on the grid.

$$\begin{array}{ccccccc} \times & & \times & & \times & & \times & & \times & & \times & & \times \\ & & x_i - 2\Delta x & & x_i - \Delta x & & x_i & & x_i + \Delta x & & x_i + 2\Delta x & & \end{array}$$

(a) Using Taylor series, show that

$$\frac{f(x_i + \Delta x) - f(x_i - \Delta x)}{2\Delta x} = \left. \frac{df}{dx} \right|_{x_i} + O((\Delta x)^2) \tag{1}$$

where the $O((\Delta x)^2)$ term refers to an error term whose size varies with the square of Δx (times some other stuff). As Δx is made smaller and smaller the error will also become smaller and smaller—quadratically in fact. We therefore expect that for sufficiently small Δx our formula (1) provides an accurate approximation of the derivative.

Often we try to simplify matters by using a shorthand notation in which our grid points are numbered from left to right, starting at one. The i^{th} point on the grid is then $x_i = (i - 1)\Delta x$ and the points to right and left of this point are $x_i + \Delta x = x_{i+1}$ and $x_i - \Delta x = x_{i-1}$. To further simplify things we let the function evaluated at the i^{th} point be denoted by $f(x_i) = f_i$. The

function values at the neighboring points are then $f(x_{i+1}) = f_{i+1}$ and $f(x_{i-1}) = f_{i-1}$. Putting this all together, a shorthand version of our approximation then looks like

$$\left. \frac{df}{dx} \right|_i \approx \frac{f_{i+1} - f_{i-1}}{2\Delta x} \quad (2)$$

The one-sided variants. The method described above is referred to as a *centered difference* approximation, since it approximates a derivative at the i^{th} point using the two grid points to either side. (That is, the approximation point x_i is centered between the two grid points used in the difference.) Unfortunately, sometimes the data on either side of our point of interest is not actually available. For instance, suppose we're at the leftmost edge of our 1D grid (i.e., at the point labeled x_1). At this point we have a neighboring grid point to our right but no neighboring point to the left. So what do we do now? Well, we'll just have to modify our approximation.

(b) Show using Taylor series that

$$\frac{f(x_i + \Delta x) - f(x_i)}{\Delta x} = \left. \frac{df}{dx} \right|_{x_i} + O(\Delta x)$$

where the notation $O(\Delta x)$ means that in this case our error varies linearly with Δx (times some other stuff). For small Δx this linear approximation is less accurate than our centered difference—since for small Δx an $O((\Delta x)^2)$ error is smaller than a $O(\Delta x)$ error—but it has the advantage of only requiring data to our right.

(c) Similarly, show also that

$$\frac{f(x_i) - f(x_i - \Delta x)}{\Delta x} = \left. \frac{df}{dx} \right|_{x_i} + O(\Delta x)$$

which then only requires data to the left.

The approximations described in (b) and (c) are referred to as *one-sided differences* since they require information only from the point to the left of our point of interest or else from the point to the right (but not from both). The advantage of these approximations is that we can use them at the grid edges. But since the one-sided approximations are less accurate than the centered difference (for small Δx at least), we'll want to stick with our centered approximation wherever possible.

Finally, in terms of our shorthand notation the one-sided differences look like

$$\left. \frac{df}{dx} \right|_i \approx \frac{f_{i+1} - f_i}{\Delta x} \quad \text{and} \quad \left. \frac{df}{dx} \right|_i \approx \frac{f_i - f_{i-1}}{\Delta x} \quad (3)$$

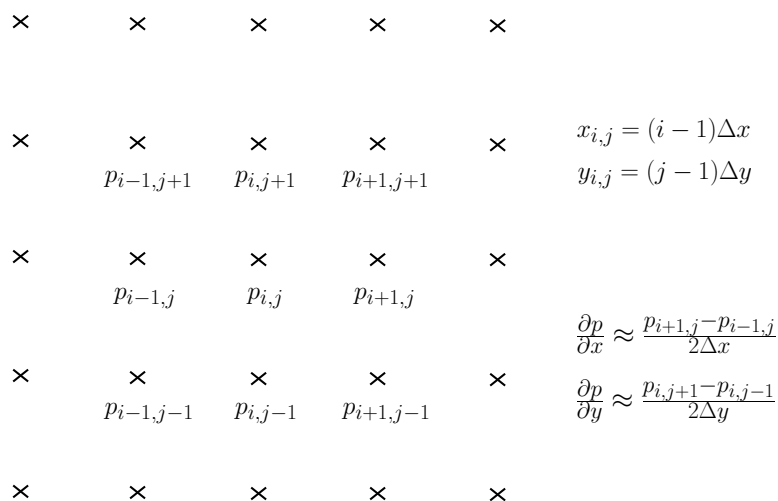
The 2D Case. The 1D method described above extends to the 2D (or even 3D) case more or less directly. The only change is that now when we take a derivative in the x -direction we

have to make sure to keep the y coordinate fixed (and vice-versa). (Of course, the notation changes a bit as well—that is, we use $\partial/\partial x$ instead of d/dx to denote the derivative. But this just reminds us that we’re keeping our other coordinate variables—in this case y —fixed.)

Now suppose we have a function $p(x, y)$ and that the value of this function is defined on an evenly spaced, rectangular set of grid points in 2D (as illustrated below). We’ll let the distance between any two adjacent grid points in the x -direction be Δx , while the distance between any two points in the y -direction is Δy . (In practice we’ll assume $\Delta x = \Delta y$, but in principle they could be different.)

Since our grid is two dimensional we’ll need two indices to keep track of the points. We’ll let the index i keep track of the position in the x direction (increasing eastward) and we’ll let j keep track of position in the y direction (increasing northward). The point labeled (i, j) on the grid then has the x and y coordinates given by

$$x_{i,j} = (i - 1) \Delta x \quad \text{and} \\ y_{i,j} = (j - 1) \Delta y$$



Once we have the grid set up, the 2D extensions for the 1D centered difference defined in (2) are relatively straightforward. Specifically

$$\left. \frac{\partial p}{\partial x} \right|_{i,j} \approx \frac{p_{i+1,j} - p_{i-1,j}}{2\Delta x} \quad \text{and} \quad \left. \frac{\partial p}{\partial y} \right|_{i,j} \approx \frac{p_{i,j+1} - p_{i,j-1}}{2\Delta y} \quad (4)$$

Note that we’ve kept our y -coordinate fixed when approximating $\partial p/\partial x$ and our x -coordinate fixed while approximating $\partial p/\partial y$. Similarly, the corresponding one-sided derivatives in x must be

$$\left. \frac{\partial p}{\partial x} \right|_{i,j} \approx \frac{p_{i+1,j} - p_{i,j}}{\Delta x} \quad \text{and} \quad \left. \frac{\partial p}{\partial x} \right|_{i,j} \approx \frac{p_{i,j} - p_{i-1,j}}{\Delta x} \quad (5)$$

while the one-sided derivatives in y are

$$\left. \frac{\partial p}{\partial y} \right|_{i,j} \approx \frac{p_{i,j+1} - p_{i,j}}{\Delta y} \quad \text{and} \quad \left. \frac{\partial p}{\partial y} \right|_{i,j} \approx \frac{p_{i,j} - p_{i,j-1}}{\Delta y} \quad (6)$$

Using (4)–(6) we can approximate both $\partial p/\partial x$ and $\partial p/\partial y$ at any point on our 2D grid (including both the interior points and the grid edges).

Error Testing. As before, we’ll determine the error in our calculations by first comparing against a known solution. To get started, copy the script files *get_field.m* and *get_error.m* from

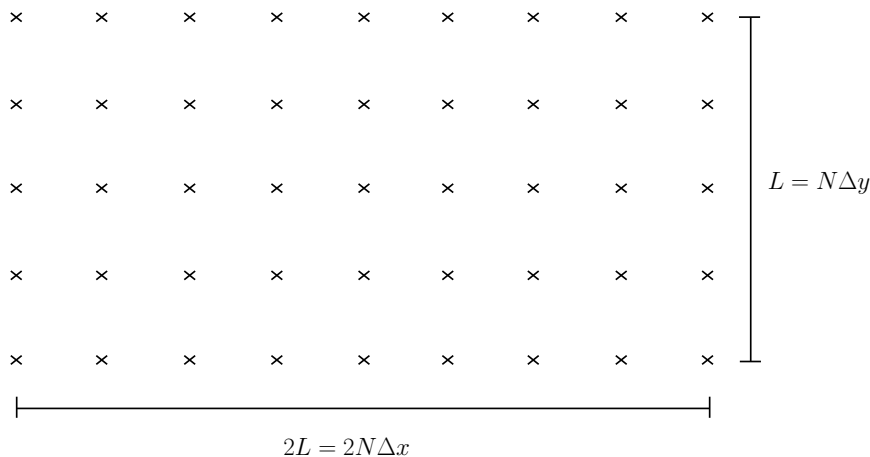
the shared ATMO336 directory to your home directory. The first script is used to define the pressure field, while the second script estimates the error in your gradient calculations. With a few exceptions the two scripts are already done for you and will not need to be modified (again, with a few exceptions).

(d) As our test pressure field we'll use

$$p(x, y) = \cos(x) \cos(y) \tag{7}$$

You'll need the gradient of this function later when computing the error, so go ahead and evaluate the gradient now [as a function of (x, y)] and add the result to the `get_error.m` script where indicated. Note that `get_error.m` already knows about the x and y coordinates on the grid. So to indicate the x -coordinate, for instance, you can just use `x(i,j)`. (And similarly `y(i,j)` gives the y -coordinate.)

Defining the grid. Now to set up our grid. For our computational domain we'll consider a rectangle given by $-3\pi/2 \leq x \leq \pi/2$ and $-\pi/2 \leq y \leq \pi/2$. Note that if $L = \pi$ is the domain width in the y -direction, then the domain width in the x -direction is $2L$. We'll want the grid spacing to be the same in both directions (i.e, we want $\Delta x = \Delta y$), so we'll end up using twice as many grid intervals in the x -direction as in the y -direction (as illustrated below).



Now suppose that the computational domain consists of N grid *intervals* in the y -direction (so that $L = N\Delta y$) and $2N$ grid *intervals* in the x -direction (so that $2L = 2N\Delta x$). And suppose that the number of grid *points* in the y -direction is ny and the number of grid *points* in the x -direction is nx . Then how are nx and ny related to N ? (No need to write this down.)

The discretized gradient. Your friendly neighborhood TA (FNTA) has given you a function script (defined in `get_fields.m`) that defines the pressure field for you on the grid described above. The function actually defines any of three different pressure fields, depending on the

value of a flag that's specified in the function call. (Our test problem (7) corresponds to *flag* = 1). The syntax for the function call looks something like

$$[x, y, p] = \text{get_fields}(N, \text{flag});$$

where N is the number of grid *intervals* in the y -direction (as above) and where *flag* is either 1, 2 or 3. The returned variables are then the x -coordinate, the y -coordinate and the pressure field. Each of the returned variables is defined on the grid points and is thus a function of the two indices i and j (i.e., $x(i, j)$, etc).

With that, we can now compute our discretized derivatives. You'll want to put the following in a script (say *gradient.m*) since you'll need to modify various aspects of this calculation later on. To be concrete we'll let our grid feature 32 intervals in the y -direction and 64 in the x -direction. A script to compute the derivatives might then look like

```
N = 32;
[x, y, p] = get_fields(N, 1);
dx = ; dy = ; nx = ; ny = ;

%% centered differencing for interior points
for i=2:nx-1
for j=1:ny
    dpdx(i, j) = ;
end
end
for i=1:nx
for j=2:ny-1
    dpdy(i, j) = ;
end
end

%% one-sided differencing for grid edges
for j=1:ny
    i = 1;
    dpdx(i, j) = ;
    i = nx;
    dpdx(i, j) = ;
end
for i=1:nx
    j = 1;
    dpdy(i, j) = ;
    j = ny;
    dpdy(i, j) = ;
end
```

where of course there are lots of things for you to fill in.

Making the plot. To plot your discretized gradient, first copy over the *plotgrad.m* script from the *ATMO336* directory. (No need to modify this one.) Calling *plotgrad* from the command line then does two things. First the isobars are plotted and the script then pauses for reflection. Hitting the spacebar (or any other key) then overlays the gradient vector.

Go ahead and take a look. (Remember to first run your *gradient.m* script so that you have a discretized gradient to plot!) Does the gradient vector look reasonable?

Convergence testing. Nice gal that she is, your FNTA has also given you a script—the *get_error.m* script—to compute the error in your computations. (You’ve already copied and edited this one previously, so don’t overwrite!) The error function takes x , y , dpx and dpy as arguments and returns a rough estimate of the fractional error. (You can think of this as roughly the average percentage error on the grid—divided by 100.) Find the error for your $N = 32$ case by adding

$$E = \text{get_error}(x, y, dpx, dpy)$$

to the end of your *gradient.m* script. Note that $N = 32$ is pretty good resolution, so your error should be small. In particular, if your error is bigger than 0.0025 then you should go back to your code and try, try again.

(e) Way back when we argued that the error in our centered difference approximation should depend roughly on the grid spacing squared. Since we’ve set $\Delta x = \Delta y$, we’ll just call this a $O((\Delta x)^2)$ error and forget about Δy for the moment. Test your predicted error dependence by computing the error for $N = 4, 6, 8, 10, 16, 24, 32$ and 64 . Then plot your results as a function of Δx and save to a jpg file. Does the error in fact go down quadratically as Δx decreases?

Interpreting the Gradient Geometrically. Your FNTA likes her calculations clean, so let’s say our error tolerance for this lab will be 1% (or fractional error of 0.01). Take a look at your error curve and pick the smallest value of N for which the error is less than 0.01. Then reset N to this value in your *gradient.m* script.

(f) Make the gradient plot, save the result to *grad_1.jpg* and copy to the figure directory to turn in. Then briefly answer the following:

- How is the direction of the gradient vector related to the pressure contours?
- How is the magnitude of the gradient related to the spacing of the contours?
- If the pressure contours were a topo map (with the terrain height determined by the value of p), would the gradient vector point uphill, downhill or along the line of constant height?

(g) Finally, go back and look at the other two pressure fields defined in *get_fields.m*. (To do this, just change the last argument in your *get_fields* call—at the top of your *gradient.m*

script—from 1 to 2 ... and then eventually to 3). Make a plot for each case and save the results to *grad_2.jpg* and *grad_3.jpg*. And copy to the figure directory to turn in.

Would your answers to the three questions given above be any different for these cases?

And as always, find the person two seats down to the right and argue over whose figures are best.