Lab assignment #3: Gaussian quadrature, numerical differentiation

Nico Grisouard (Instructor), Alex Cabaj (TA & Marker),
Pascal Hogan-Lamarre (TA)

Due Friday, October 2nd 2020, 5 pm

---

1. **More on the error function**

   (a) Dawson function.

   i. When evaluating $(2/\sqrt{\pi})\int_0^3 \exp(-x^2)\mathrm{d}x$ with SciPy, the answer is

```
D(4.0) = 0.1293480012360051
```

and when evaluating with the three integration rules we now know, we get:

```
For N = 8 slices/sample points:
  Trapezoidal says 0.2622478205347951
      Simpson says 0.18269096459712167
        Gauss says 0.1290106788170698

For N = 16 slices/sample points:
  Trapezoidal says 0.16828681895583716
      Simpson says 0.13696648509618448
        Gauss says 0.12934800119977766

For N = 32 slices/sample points:
  Trapezoidal says 0.1395800909267732
      Simpson says 0.13001118158375188
        Gauss says 0.1293480012360034
```

Note that at this point, the Gaussian quadrature agrees with SciPy's estimate within a relative error of about 3×10^{-16}, with is very, very close to the round-off error. While the two other methods will keep converging after this point, the Gaussian quadrature will tiptoe around SciPy's value: it has converged.

```
For N = 64 slices/sample points:
  Trapezoidal says 0.13194038496790614
      Simpson says 0.12939381631495048
        Gauss says 0.12934800123600462
```

```
For N = 128 slices/sample points:
  Trapezoidal says 0.1299983024925397
      Simpson says 0.12935094166741753
        Gauss says 0.12934800123600457

For N = 256 slices/sample points:
  Trapezoidal says 0.12951071531441982
      Simpson says 0.12934818625504652
        Gauss says 0.1293480012360041

For N = 512 slices/sample points:
  Trapezoidal says 0.12938868844305074
      Simpson says 0.12934801281926103
        Gauss says 0.1293480012360052

For N = 1024 slices/sample points:
  Trapezoidal says 0.1293581735809614
      Simpson says 0.12934800196026497
        Gauss says 0.12934800123600546

For N = 2048 slices/sample points:
  Trapezoidal says 0.12935054435619747
      Simpson says 0.12934800128127613
        Gauss says 0.12934800123600487
```

ii. For all methods, but particularly for the Gaussian Quadrature, the error falls by so many orders of magnitude that we have to use a lol-log scale. As noticed before, we see more clearly on fig. 1 that the error sort of stabilizes after about 32 sample points for the Gaussian quadrature. In this case, Gaussian quadrature beats the two other methods hands down for all number of slices.

(b) Not asked, but the change of variables

$$t = \frac{u - \bar{u}}{\sqrt{2}\delta}, \qquad \mathrm{d}t = \frac{\mathrm{d}u}{\sqrt{2}\delta} \tag{1}$$

yields, for the probability of blowing snow,

$$P(u_{10}, T_a, t_h) = \frac{1}{2}\int_{-\frac{\bar{u}}{\sqrt{2}\delta}}^{\frac{u_{10}-\bar{u}}{\sqrt{2}\delta}} \frac{2}{\sqrt{\pi}} \mathrm{e}^{-t^2} \mathrm{d}t, \tag{2}$$

which is the formula I implement in my code.

The probability increases sensitively to changes in wind and decreases moderately as the snow ages. Indeed, as the wind gets stronger, it is more able to provide lift to snow on the ground. As the snow ages, it gets harder and more compressed/less powdery, and therefore harder to lift. As the wind increases, the temperature at which blowing snow is most likely to occur decreases.

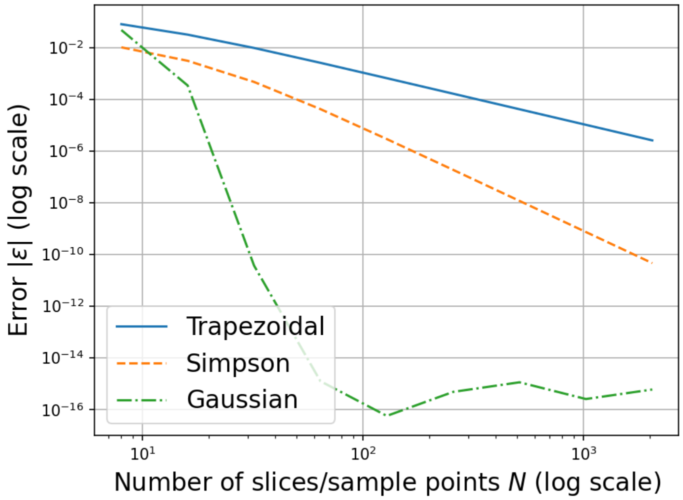

Figure 1: Plot for Q1a.ii

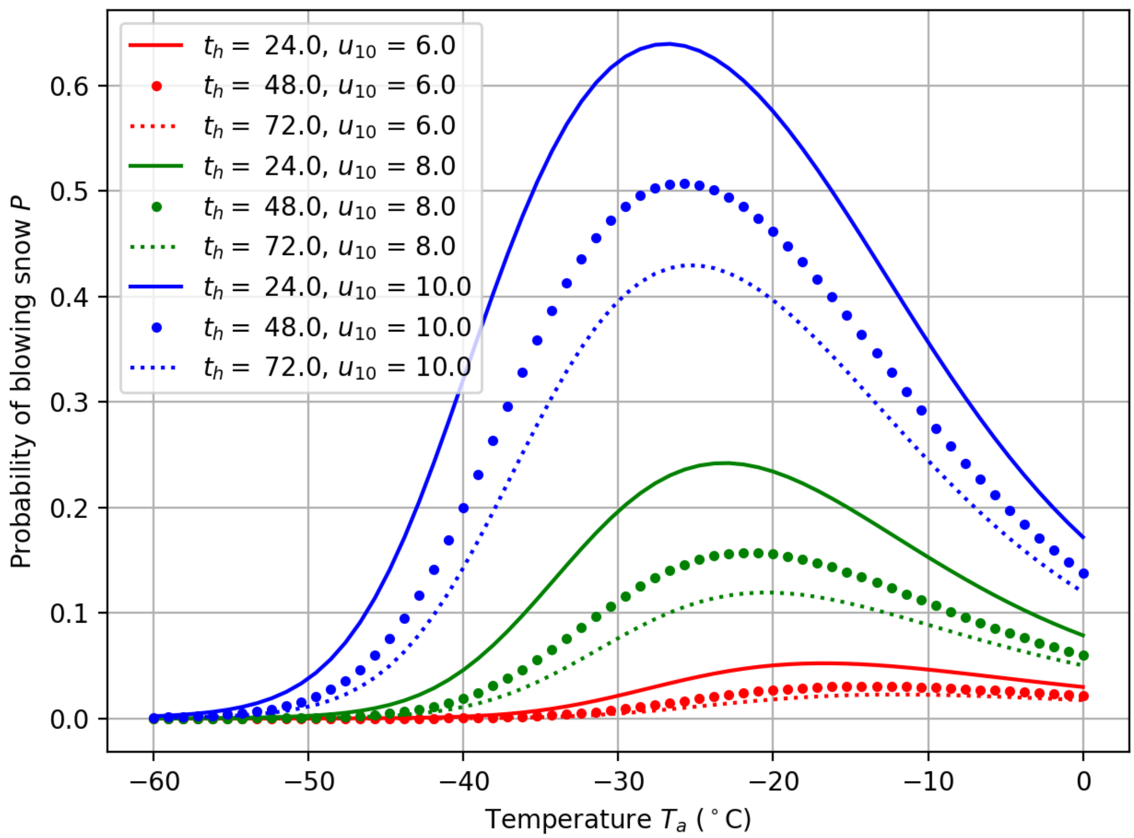

Figure 2: Plot for Q1b.

2. **Calculating quantum mechanical observables**

   (a) See fig. 3.

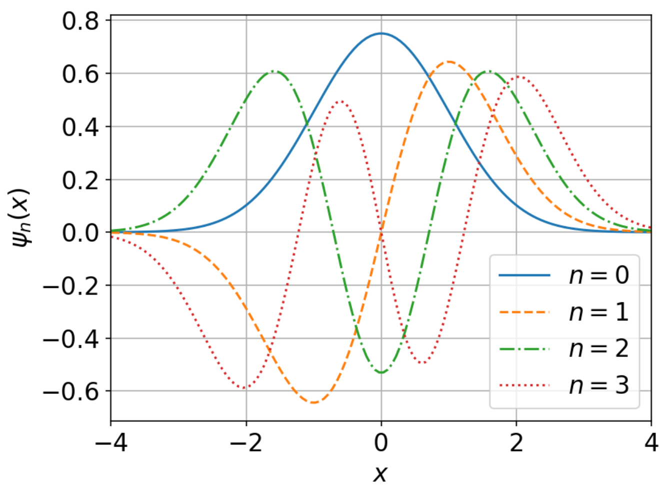

Figure 3: Plot for Q2a.

   (b) See fig. 4.

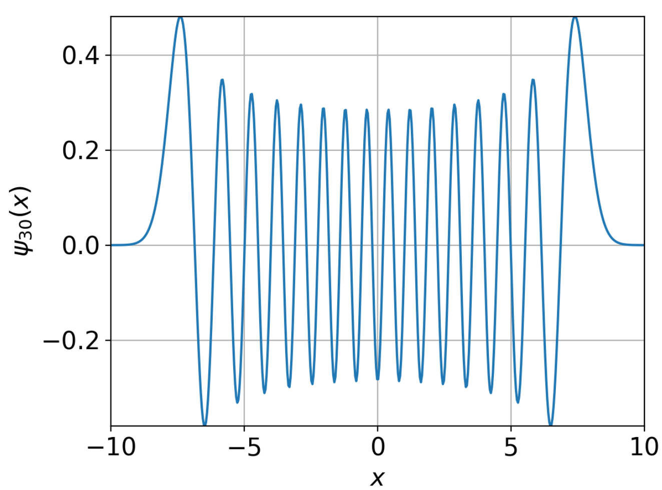

Figure 4: Plot for Q2b.

   (c) Code: See `L03-407-2020-Q2.py`. Here is some printed output:

```
Let's verify that for n=5, sqrt(<x**2>) = 2.3452078797796547
```

```
 n    sqrt(<x^2>)    sqrt(<p^2>)        E
---------------------------------------------
 0,    7.071e-01,     7.071e-01,     5.00e-01
 1,    1.225e+00,     1.225e+00,     1.50e+00
 2,    1.581e+00,     1.581e+00,     2.50e+00
 3,    1.871e+00,     1.871e+00,     3.50e+00
 4,    2.121e+00,     2.121e+00,     4.50e+00
 5,    2.345e+00,     2.345e+00,     5.50e+00
 6,    2.550e+00,     2.550e+00,     6.50e+00
 7,    2.739e+00,     2.739e+00,     7.50e+00
 8,    2.915e+00,     2.915e+00,     8.50e+00
 9,    3.082e+00,     3.082e+00,     9.50e+00
10,    3.240e+00,     3.240e+00,     1.05e+01
11,    3.391e+00,     3.391e+00,     1.15e+01
12,    3.536e+00,     3.536e+00,     1.25e+01
13,    3.674e+00,     3.674e+00,     1.35e+01
14,    3.807e+00,     3.808e+00,     1.45e+01
15,    3.937e+00,     3.938e+00,     1.55e+01
```

We can see energy equipartition ($\langle x^2 \rangle = \langle p^2 \rangle$) for every mode, and we can see that for a given mode,

$$E_n = \frac{1}{2} + n. \tag{3}$$

3. **Generating a relief map:**

   (a) For pseudo-code, see `L03-407-2020-Q3.py`.

   (b) Elevation map is in fig. 5, partial slopes ($\partial_x w$, $\partial_y w$) are in fig. 6, and the final calculation is in fig. 7.

   The main difference between w and I maps are that the w maps are more intense where there is a slope, rather than an elevation.

   On fig. 8, we compare the derivatives at the end points vs. the derivatives immediately next to them, because the values should be somewhat different, but not by much. That the partial slopes were sensible in fig. 6, and that the curves are now close-ish to each other means that we have not made any big mistake (wrong step, wrong side, wrong index...).

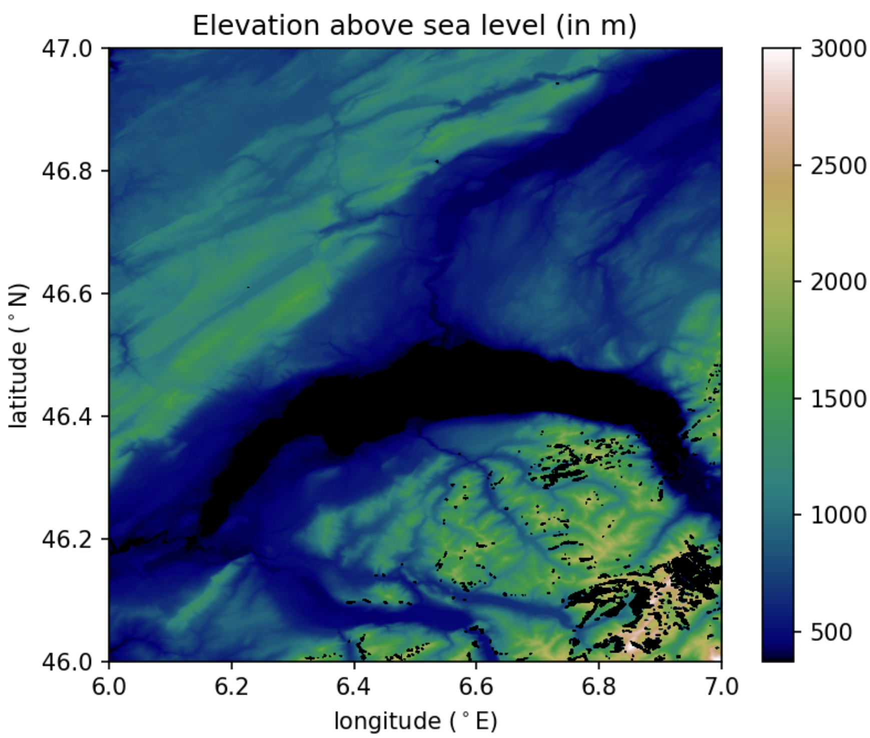

Figure 5: Plot of the elevation for Q3b.

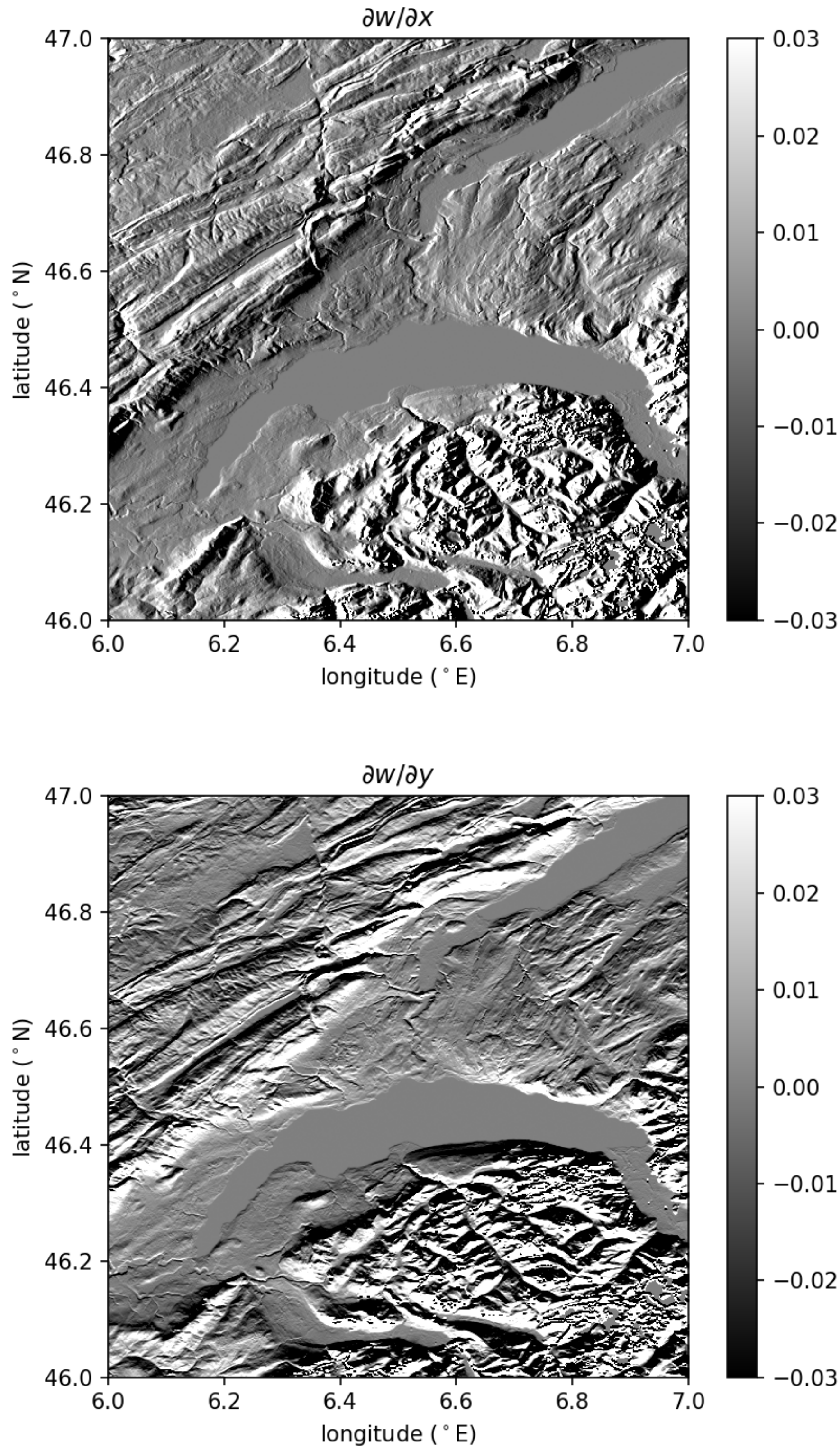

Figure 6: Plot of the partial slopes $\partial_x w$ and $\partial_y w$ for Q3b.

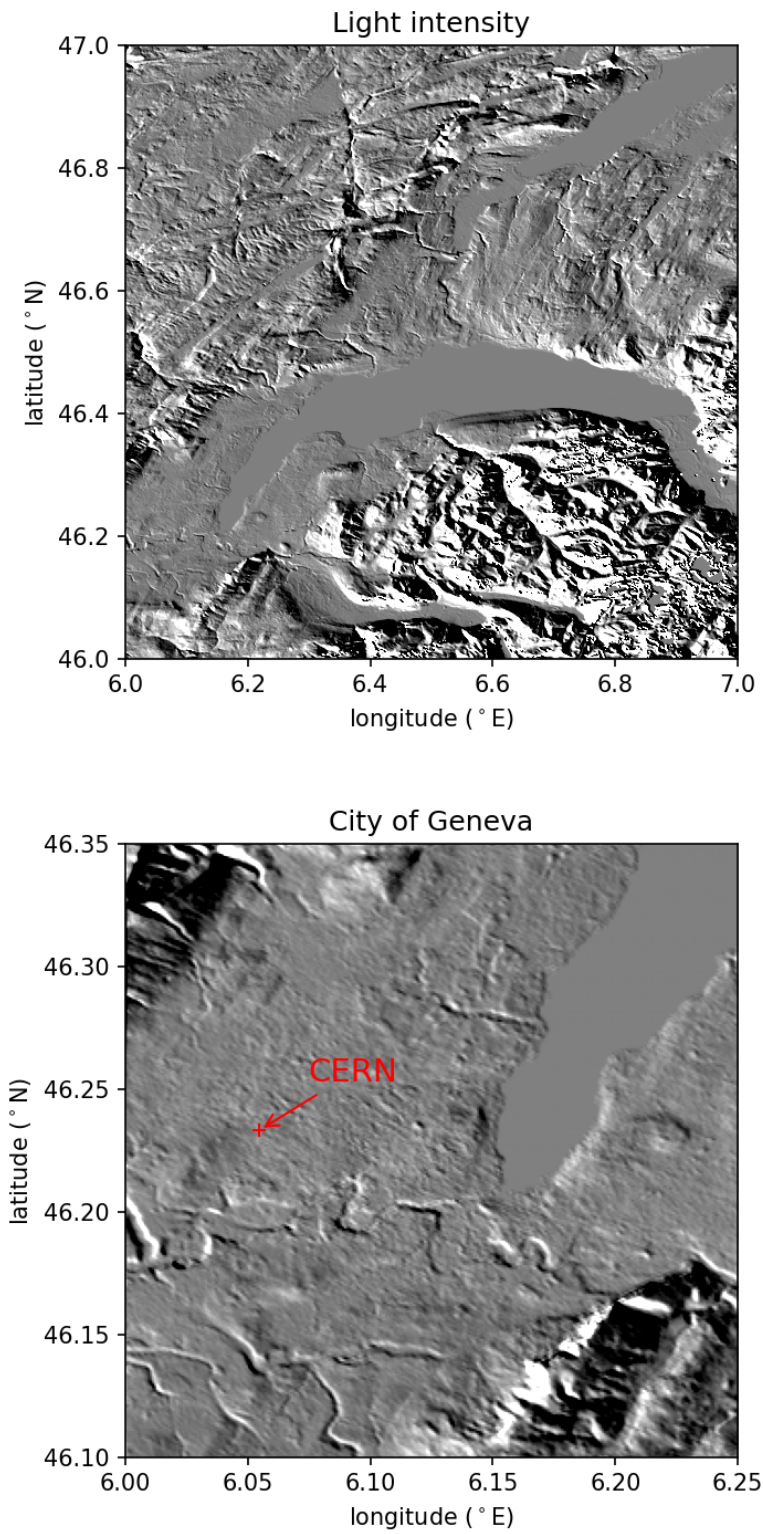

Figure 7: Top: Illumination of the map for $\varphi = -135°$. Bottom: close-up around city of Geneva.

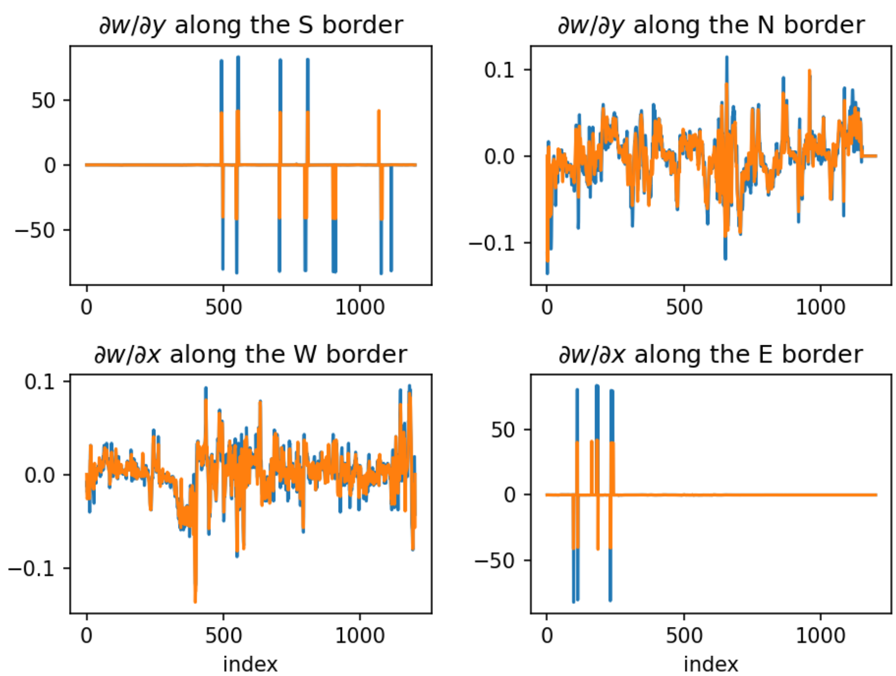

Figure 8: Rough check that the borders are correctly treated for Q3b. The blue curves are the derivatives at the borders, calculated via forward of backward differences. The orange curves are the derivatives, one cell immediately to the interior of the domain, calculated with central differences.