MATH327: StatMech and Thermo, Spring 2025

Extra practice — Drift and diffusion

Last year there was an oil spill near the Caribbean island of Tobago. On 14 February 2024, changing currents caused several million litres of spilled oil to start moving towards Grenada's territorial waters, $144\ \mathrm{km}$ away. We can analyze the motion of the oil by treating each droplet as a random walker moving in one dimension — towards or away from Grenadan waters. Satellite images and ocean models were used to estimate the rate at which the oil was moving. Suppose they indicated a drift velocity $v_{\mathrm{dr}} = 6\ \mathrm{km/hour}$ towards Grenadan waters, with a diffusion constant $D = 8\ \mathrm{km}/\sqrt{\mathrm{hour}}$.

How many hours did Grenada have in which to take action before 1% of the spilled oil was inside its waters? How much additional time did it take for the amount of oil inside Grenadan waters to double to 2% of the total?

Suppose the oil were moving towards the island of Grenada itself, rather than its territorial waters. It is significantly more complicated to analyze the situation of oil washing up on Grenada's shores, because each droplet's random walk would *stop* once it reached the shore and left the water. This is known as a *first-passage process*. Without attempting this more complicated calculation, determine whether it will take more time, less time or the same amount time for the spilled oil to wash up on shore, compared to entering territorial waters, with everything else the same.

Hint: The error function

$$\mathrm{erf}(u) = \frac{1}{\sqrt{\pi}} \int_{-u}^{u} e^{-x^2}\, dx \equiv P$$

may appear in your work, with $u > 0$. SciPy is one tool you can use to invert the error function to find $u = \mathrm{erf}^{-1}(P)$ for a given $0 < P < 1$. Here are some examples:

```
>>> import math
>>> from scipy import special
>>>
>>> sigmas = [0.682689492, 0.954499736, 0.997300204, \
...                        0.999936656, 0.999999427]
>>> for P in sigmas:
...   u = special.erfinv(P)
...   n = round(u * math.sqrt(2.0))
...   print("u = %.4f for P=%.7f (%d sigma)" % (u, P, n))

u = 0.7071 for P=0.6826895 (1 sigma)
u = 1.4142 for P=0.9544997 (2 sigma)
u = 2.1213 for P=0.9973002 (3 sigma)
u = 2.8284 for P=0.9999367 (4 sigma)
u = 3.5356 for P=0.9999994 (5 sigma)
```