Computational Physics II — Project 2

Jan O. Haerter

February 20, 2022

Hand-in date: 6 a.m., Friday, Mar 4, 2022

In total, **100 points** can be achieved by solving the problems below (see annotation).

Numerical integration of the 1D diffusion equation

The diffusion equation is of fundamental importance in physics and serves as a continuum version of random walks (studied already in Project 1). A particular class of problems, widely relevant to biology, population dynamics and evolutionary theory, are first-passage problems. We will explore these topics here. To implement the Crank-Nicolson method to simulate 1D diffusion, beyond the usual numpy and matplotlib libraries, please also import a routine for solving a banded matrix equation

```
from scipy.linalg import solve_banded
```

Setting up the computation. (**30 pts for generally correct code**)

- Define the spatial domain size, n_x, to be sufficiently large, say, 1000 elements. Define the collision time Δt, mean free path $l = \Delta x$, and the diffusion coefficient D. From these, define the coefficient $r \equiv D\Delta t/\Delta x^2$. While the value of r, which could be called "Diffusion number,"[1] does not impact on numerical stability, it can affect numerical accuracy. A good ballpark is $r \approx 1$. Define a position vector, where $x_i = i\Delta x$, $i \in \{0, n_x - 1\}$. Also, define two arrays P_{curr} and P_{next} of length n_x to store the density $P(x,t)$ at the current and the subsequent timestep, respectively. Define boundary conditions (needed for the array $\mathbf{B}$), at $x = 0$ and $x = (n_x - 1)\Delta x$ (first and last elements of $P(x,t)$), such that $P(0,t) = P((n_x - 1)\Delta x, t) = 0$.

- Define the banded matrix $\mathbf{A}$ as a $3 \times n_x$-element array (three rows and n_x columns). Using the Crank-Nicolson scheme, discussed in class, populate the matrix $\mathbf{A}$. This matrix will consist of the diagonal, as well as one upper and one lower diagonal (beware of Python's convention on defining the three rows of the banded matrix). Note that defining $\mathbf{A}$ needs to be done only once, given that r is taken to be a constant. Set up a routine that evaluates the vector $\mathbf{B}$, which depends on $P(x,t)$ at the current time t.

[1] corrected typo

- Finally, solve the implicit matrix equation $\mathbf{AP} = \mathbf{B}$ using scipy's `solve_banded` routine, thus obtaining the vector P_{next} from P_{curr}.

Project tasks.

1. Set the density to be a numerical "Dirac Delta" in the center of your spatial domain.

$$P(x_i, 0) = \begin{cases} 1 \text{ if } x_i/\Delta x = n_x/2 \\ 0 \text{ otherwise .} \end{cases} \tag{1}$$

   Note that $P(x, 0)$ is normalized in units of system size, that is, $P_{tot} \equiv \sum_{i=0}^{n_x-1} P(x_i, 0) = 1$. Plot $P(x, 0)$. (**10 pts**)

2. Using your solver, obtain the subsequent density P_{next}, check and store the normalization and store the mean $\langle x \rangle$ and variance $\langle x^2 \rangle - \langle x \rangle^2$ of P_{next}. By overwriting $P_{curr} = P_{next}$, repeat the updating a large number of times n_t, e.g. $n_t \gtrsim \mathcal{O}(10^4)$. Finally, plot the statistical quantities as function of time. Also plot $P(x, t)$ for various values of t on the same graph. Is the total density constant? If it is changing, comment on this. Describe the mean and variance and compare them with theory. (**25 pts**)

3. Now shift your initial condition (Eq. 1) such that the "Dirac Delta" is located near one of the boundaries, say at $x_i/\Delta x = 10$. Repeat the computation from before. Besides the previous values, also plot the change in density, $\Delta P_{tot} \equiv P_{tot,curr} - P_{tot,next}$ as a function of time. Interpret your findings in terms of the Smirnov density. (**25 pts**)

4. By increasing and decreasing the value of r by an order of magnitude, discuss the numerical accuracy of your results. *Note:* When comparing to exact results, remember that a change to r means a change to at least one of the three parameters involved. (**10 pts**)

General remarks for all Projects. You will have to (i) analyze the problem, (ii) select an algorithm (if not specified), (iii) write a Python program, (iv) run the program, (v) visualize the data numerical data, and (vi) extract an answer to the physics question from the data. Which checks did you perform to validate the code? State the results you got for these tests. For each project you will submit a short report describing the physics problem, your way of attacking it, and the results you obtained. Provide the documented Python code in such a form that we can run the code. A Jupyter Notebook including the code and report is fine but not necessary.