FYS-4096 Computational Physics: Exercise 2

Ilkka Kylänpää

Week 3

General:

- You should have access to help files on teacher's repo `Comp Phys Spring 2021`. If not, please provide your username and associated email to the teacher.
- In the repo you will find a folder `ex2_help` with files `num_calculus.py`, `linear_interp.py` and `spline_class.py`. Of those files at least the last two are needed in these exercises.

Problem 1: Integration
You can use the integration function you made yourself in Exercise 1 or, e.g., trapezoid or Simpson rule from scipy.integrate (i.e., from scipy.integrate import simps):

- Idea: Get numerical estimates for the integrals show in (a), (b), and (c) using trapezoidal or Simpson integration. In a hypothetical project your code would encounter huge amounts of similar integrals than are shown in (a), (b) and (c), but with small changes. Here you want to make sure that you get a single integration done accurately enough, yet not spending too much time for it. You have (somehow) found out that a four decimal accuracy would be enough for the success of the grand project.

a) Integrate the integrals below numerically.

$$\int_0^\infty r^2 e^{-2r} dr, \qquad \int_0^1 \frac{\sin(x)}{x} dx, \qquad \int_0^5 \exp(\sin(x^3)) dx$$

b) Calculate with your method of choice the integral

$$\int_{y_0}^{y_1} \int_{x_0}^{x_1} f(x,y) dx dy,$$

where $f(x,y) = x \exp(-\sqrt{x^2+y^2})$, $x_0 = 0$, $x_1 = 2$, $y_0 = -2$, and $y_1 = 2$.

c) Calculate with your method of choice the integral

$$j = \int \frac{|\Psi(\mathbf{r} - \mathbf{r}_A)|^2}{\|\mathbf{r} - \mathbf{r}_B\|} d\mathbf{r},$$

where $\Psi(\mathbf{r}) = e^{-r}/\sqrt{\pi}$, where $r = \|\mathbf{r}\| = \sqrt{x^2+y^2+z^2}$. Compare this at a few points (~ 5 points) with the analytic result $j(R) = (1 - (1+R)e^{-2R})/R$, where $R = \|\mathbf{r}_A - \mathbf{r}_B\|$.

- Notice 1: In (a) and (b) you should be getting values close to 0.25, 0.95, 6.65, 1.57. In (c) the exact answer for different R values is already given.
- **Notice 2: You can make an estimate on the accuracy, e.g., by varying the grid spacing from coarse to finer. For infinite limit you could look at the shape of the integrand etc.**

Problem 2: Interpolation (Getting familiar with details)

- Download modules `linear_interp.py` and `spline_class.py`.
- Add appropriate basis functions to both modules, e.g., $l_1(t)$ and $l_2(t)$ for the linear case. See lecture slides for the basis functions.
- After adding the basis functions run the modules.
- Add comments and explanations in the code. This should reflect your understanding / knowledge of the details.
- Make appropriate test functions for 1D, 2D and 3D cases. In the tests compare to analytical values at some non-grid points. Use for example, average of squared difference etc. in assessing the accuracy, i.e., $1/N \sum_i (f(x_i) - S(x_i))^2$.
- Make presentable figures either from the provided examples in the modules or according to your desire.
  - Food for thought (no need to do anything related to this "bullet point"): How to use splines to smooth noisier data? In general, merging all interpolation routines (linear, splines, etc.) in one interpolation module could be desirable.

Problem 3: Root search

- Consider writing your own routines for finding indices needed in interpolation when

  a) your grid is linear such as given by `numpy.linspace`.

  b) your grid generation function takes in values `r_0`, `h`, and `dim` and is given as

```
r[0]=0.
for i in range(1,dim):
    r[i]=r_0*(exp(i*h)-1)
```

  where $h = \log(r_{\max}/r_0 + 1)/(\dim - 1)$. Choose for example, $r_{\max} = 100$, $r_0 = 1e-5$ and $\dim = 100$.

Problem 4: Gradient and steepest descent

a) Add an implementation for N-dimensional numerical gradient in your `num_calculus.py` or in a new module. Remember testing and commenting. For testing use the function $f(\mathbf{x}) = \sin(x_1) + \cos(x_N) + \sum_{i=2}^{N-1} x_i^2$, where $N \geq 3$. Notice also that your gradient should work for $N = 1$, i.e., in 1D.

b) Make a simple (i.e., no need to fine tune) steepest/gradient descent function based on Eqs. (31) and (32) on lecture slides. Consider the absolute value in Eq. (31) as length of a vector. Remember testing and commenting. As your test, considerin finding the minimum of $f(\mathbf{x}) = \sum_i x_i^2$, where $\mathbf{x}$ is a 10-dimensional vector. As your initial guess for the solution use $\mathbf{x} = \mathbf{1}$, i.e., $x_i = 1$ for all i. Despite your test case is for a N-dimensional vector where $N = 10$, your steepest descent code should work for arbitrary N.

Returning your exercise

1. Create a new folder called "exercise2" to your existing Computational Physics repo.
2. Create a file solvedProblems.txt into the "exercise2" folder. Inside it, write a comma separated list of problems you have solved, e.g., 1,2,3. Remember the more detailed instructions given in Exercise 1. Make sure all your source files are under version control and push them to GitLab.
3. If you desire, tag your final solution (git commit) with "final" tag: `git tag final`
4. Push your commits (and the final tag) to GitLab **before noon on Monday January 25**:
   `git push --all && git push --tags`
5. If you haven't yet shared your GitLab repo with the teacher, do it now. Provide proper permissions (Reporter, Developer or Maintainer), so that the teacher can also access the data. (Teacher's GitLab account: kylanpaait, ilkka.kylanpaa@tuni.fi).