7 — PHY 494: Homework assignment (35 points total)

Due Friday, April 7, 2017, 11:59pm.

Submission is to your **private GitHub repository**.

Read the following instructions carefully. Ask if anything is unclear.

1. `cd` into your assignment repository (change *YourGitHubUsername* to your GitHub username) and run the update script `./scripts/update.sh` (replace *YourGitHubUsername* with your GitHub username):

   ```
   cd  assignments-2017-YourGitHubUsername
   bash ./scripts/update.sh
   ```

   It should create three subdirectories[1] `assignment_07/Submission`, `assignment_07/Grade`, and `assignment_07/Work`.

2. You can try out code in the `assignment_07/Work` directory but you don't have to use it if you don't want to. Your grade with comments will appear in `assignment_07/Grade`.

3. Create your solution in `assignment_07/Submission`. Use Git to `git add` files and `git commit` changes.

   You can create a PDF, a text file or Jupyter notebook inside the `assignment_07/Submission` directory as well as Python code (if required). **Name your files `hw07.pdf` or `hw07.txt` or `hw07.ipynb`**, depending on how you format your work. Files with code (if requested) should be named exactly as required in the assignment.

4. When you are ready to submit your solution, do a final `git status` to check that you haven't forgotten anything, commit any uncommited changes, and `git push` to your GitHub repository. Check on *your* GitHub repository web page[2] that your files were properly submitted.

   You can push more updates up until the deadline. Changes after the deadline will not be taken into account for grading.

Homeworks must be legible and intelligible or may otherwise be returned ungraded with 0 points.

7.1 Square-root with Newton's method (15 points)

The square root function[3] $q(x)$ can be *defined* by the equation

$$q(x)^2 = x. \tag{1}$$

[1]If the script fails, file an issue in the Issue Tracker for PHY494-assignments-skeleton and just create the directories manually.

[2]`https://github.com/ASU-CompMethodsPhysics-PHY494/assignments-2017-`*`YourGitHubUsername`*

[3]The symbol $\sqrt{\cdot}$ is commonly used to denote the operation of q, so that $q(x) \equiv \sqrt{x}$. For the following it is more useful to think of the square root as a special function than just a calculation.

The goal is to develop and to implement an **efficient algorithm to compute square roots**.

The defining equation Eq. 1 can be rearranged as

$$q(x)^2 - x = 0 \tag{2}$$
$$q^2 - x = 0 \tag{3}$$
$$f_x(q) = 0 \tag{4}$$

where $f_x(q) = q^2 - x$ is now considered a function of the *variable* q and a given *parameter* x. Finding the square root $q(x)$ amounts to finding the root of $f_x(q)$, i.e., find that value q that makes Eq. 3 true for a given x.

(a) Use the iterative Newton-Raphson algorithm from the class to implement a function `sqrt(x, tol=1e-6, Nmax=100)` in a module `functions.py` that returns the square root of x to a tolerance of `tol` and uses at a maximum `Nmax` iterations. If `Nmax` is exceeded print a warning message and return `None`.[4] Your code should guess a good starting value for the Newton-Raphson scheme, e.g,. $x/2$.

In the Newton-Raphson scheme you have to calculate the update Δq to q in order to obtain the new best guess for the root

$$q \leftarrow q + \Delta q \tag{5}$$

with

$$\Delta q = -\frac{f_x(q)}{f'_x(q)} \tag{6}$$

In your code you may either use a finite difference scheme to calculate $f'_x(q)$ *or* (more efficiently), use the *analytical derivative* $f'_x(q) = 2q$ directly.[5]

Your code must produce correct results, as tested with `test_functions.py`.[6] You can run these tests yourself with

```
py.test -v test_functions.py
```

(in the same directory as your `functions.py`). **[10 points]**

(b) Show results for $x = 0, 0.456 \times 10^{-8}, 10^{-3}, 0.1, 0.64, 0.99, 1, 5, 9, 12.5, 10^3, 1.2345 \times 10^8$ for a tolerance of 10^{-6}. Put the results in a text file `sqrt.txt`. The results should be arranged so that each line contains x and $q(x)$. **[5 points]**

[4] In your code you may *not* use a library square root function such as `math.sqrt` or `numpy.sqrt` nor taking fractional powers such as `x**0.5` or `numpy.power(x, 0.5)`.

[5] Using the analytical derivatives makes for a handy algorithm to *manually* calculate square roots and indeed this is how Newton came up with the method.

[6] Some specific tests are allowed to fail and they are marked with `x` or `xfail` in the test output — this is ok.

Note: You don't have to submit any notebooks or written text. It will be sufficient to submit

- `functions.py`
- `sqrt.txt`

7.2 Numerical solution of the plate capacitor potential (20 points)

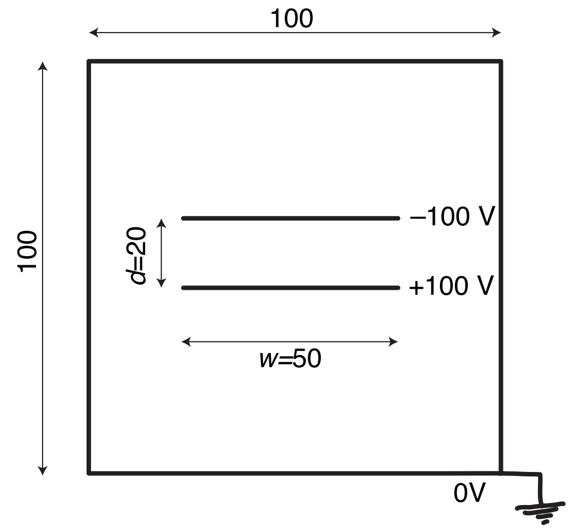

Figure 1: Schematic of the capacitor in the grounded box problem (2D). The capacitor is centered in the box. The capacitor plates are thin, conducting sheets of metal that are held at potential +100 V and -100 V, respectively.

Compute the electrostatic potential of a plate capacitor in a grounded box, shown in Fig. 1. Solve the problem in 2D. The square box has sides of length 100. Consider the capacitor plates to be conducting, very thin sheets of metal. The plates are held at an electric potential of +100 V and -100 V. The plates are $w = 50$ units wide and located at a distance $d = 20$. The capacitor is centered inside the box.

Submit your code as a python module `capacitor.py`. In particular, you should have the following functions:

`set_boundaries(Phi)` Given the potential on the lattice, enforce the boundary conditions by setting the appropriate lattice points to the prescribed values.

`calculate_phi_capacitor(Max_iter=10000, tol=1e-3)` This function should calculate the potential Φ on the lattice. It needs to take at least the two keyword arguments `Max_iter` to set the maximum number of iterations and `tol` to set the convergence criterion.

You must implement a convergence check: consider the solution converged when the Frobenius norm

$$||\Phi^{\text{new}} - \Phi^{\text{old}}|| := \sqrt{\sum_{i=1}^{M}\sum_{j=1}^{N}\left|\Phi_{ij}^{\text{new}} - \Phi_{ij}^{\text{old}}\right|^2} \tag{7}$$

is less than the set tolerance `tol`. Report the number of iterations that were required. Also report if convergence was not achieved within `Max_iter` iterations.

The function *must* return the potential Φ as a numpy array of dimensions $(100, 100)$.

It must be possible to run these functions as

```python
import capacitor
Phi = capacitor.calculate_phi_capacitor(Max_iter=2000, tol=1)
```

Start from the skeleton code in `capacitor.py` to ensure that your functions adhere to the prescribed interface.[7]

Specifically, you need to fullfil the following objectives:

(a) Your code must produce correct results, as tested with `test_capacitor.py`. You can run these tests yourself with

```
py.test -v test_capacitor.py
```

(in the same directory as your `capacitor.py`). **[15 points]**

(b) Converge your results to `tol = 1e-3`. Include in your submission:

a) Report how many iterations you required for convergence. Simply write the number in a *textfile* `iterations.txt` **[3 points]**

b) Make a 3D plot of the potential (you can use `capacitor.plot_phi(Phi)`) **[1 points]**

c) Make a 2D plot of the initial potential (boundary conditions) and of the final converged solution (you can use `capacitor.plot_panel(Phi)`) **[1 points]**

Note: You don't have to submit any notebooks or written text. It will be sufficient to submit

- `capacitor.py`

[7]The skeleton code contains a optimized function `Laplace_Jacobi()` which replaces the Gauss-Seidel code from the lecture. It runs 50 times faster so you are advised to use it... but you may use the slower code, if you prefer. The original code (as developed in class) is in the function `Laplace_Gauss_Seidel()`.

- `iterations.txt`
- `capacitor_potential_2d.pdf` and `capacitor_potential_3d.pdf` (these are the default filenames when using the functions in `capacitor.py`).