timdrysdale / pyfd

simple python implementation of finite difference electrostatic calculations

Edit

Manage topics

7 commits | 1 branch | 0 packages | 0 releases | 1 contributor | GPL-3.0

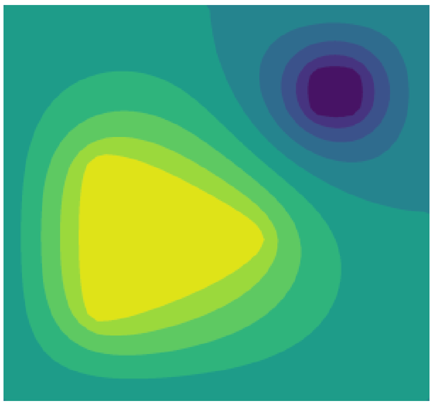
Branch: master | New pull request | Create new file | Upload files | Find file | Clone or download

timdrysdale fix img type in README Latest commit f26c9e6 17 seconds ago

img	modified readme	1 minute ago
.gitignore	added fd.py and example images	5 hours ago
LICENSE	Initial commit	6 hours ago
README.md	fix img type in README	17 seconds ago
demo.png	added face sim	3 hours ago
face.png	added pictures to README	1 hour ago
face_unsolved.png	added face sim	3 hours ago
fd.py	added pictures to README	1 hour ago
liveDemo.png	added face sim	3 hours ago
liveDemo0.png	added face sim	3 hours ago
liveDemo1.png	added face sim	3 hours ago

README.md

pyfd



A simple demonstration of finite differences for plotting potentials arising from 2D charge distributions, written for my EM3 class.

Features:

not very many! Uses fixed potential boundaries. Written in python.

Indexing is facaded behind compass directions for ease of writing and writing (avoids code with +/-1 all through it)

Usage:

Download this repository, then run from the command line (or from your IDE)

```
python fd.py
```

Assumes you have python3 installed, along with numpy and matplotlib (which you will if you use a complete distribution like Anaconda, and if not, can be easily rectified e.g. using `pip install numpy` at the command line.

Going further

Geometries

Implement your own geometry construction functions either by extending the existing Grid class, or by adding a new class that calls Grid.fixV().

Suggestions: circles, squares, rectangles, triangles, letters of the alphabet, cartoons

Accuracy

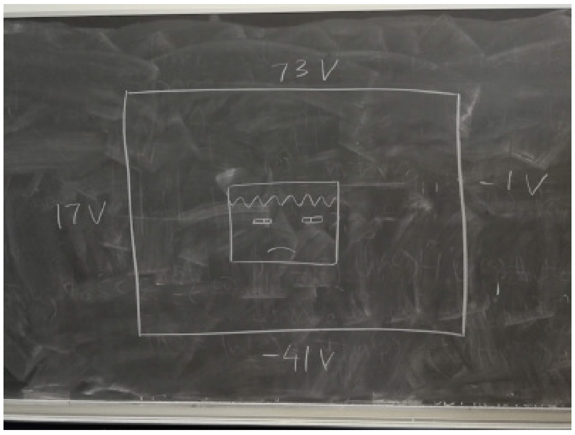
You might like to look more closely at convergece behaviour, and compare results to those obtained by hand, to understand just how much error was introduced by our assumption of an ideal capacitor.

Validation

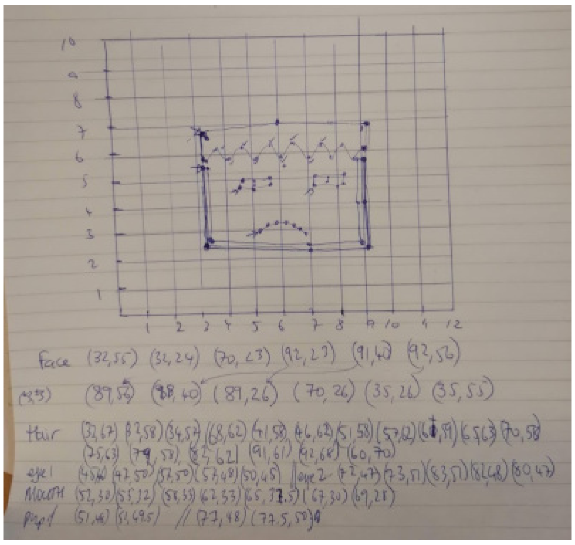
I've not done this yet, because things "look about right" - at least, so far! I've left it as an exercise for you. That might seem a bit cheeky, but actually in industry and research, any time you get a new simulation tool, you undertake a validation exercise to compare it to measurements (directly or indirectly).

Fun

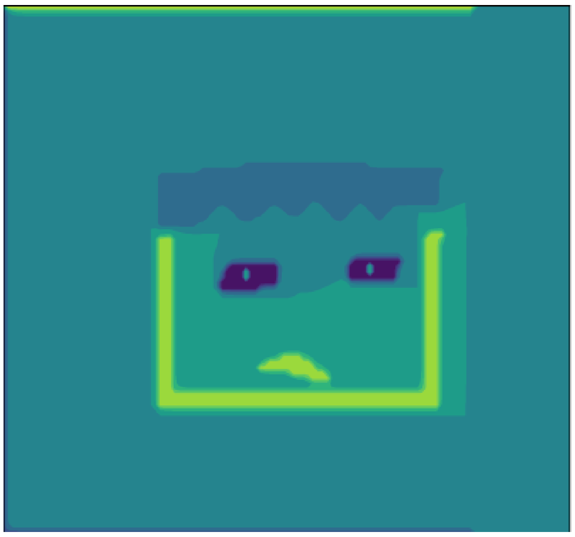
In Jan 21's lecture the class decided we should do a stylised face.



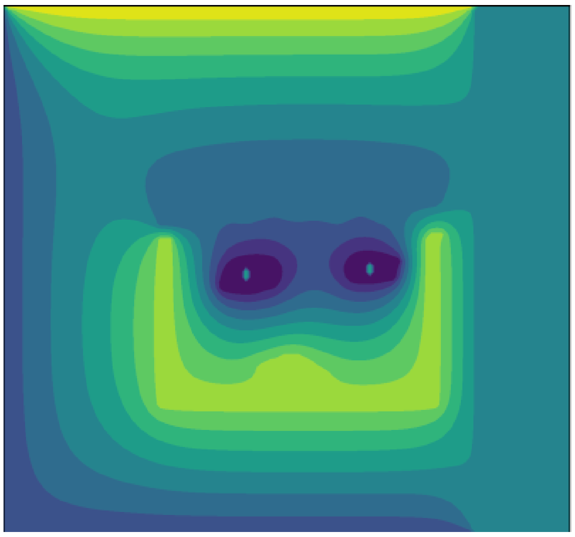
Here's a picture showing how I "digitised" it by hand (after tracing it onto paper from a photo)


Face (32,55) (32,24) (70,23) (92,27) (91,40) (92,56)
(89,55) (88,40) (89,26) (70,26) (35,26) (35,55)
Hair (32,67) (32,58) (34,57) (68,62) (41,58) (46,62) (51,58) (57,62) (61,59) (65,63) (70,58)
(75,63) (79,58) (85,62) (91,61) (92,68) (60,70)
eye1 (42,50) (57,50) (57,48) (50,45) eye2 (72,47) (73,51) (83,51) (84,48) (80,47)
Mouth (52,30) (55,32) (58,33) (62,33) (65,32.5) (67,30) (69,28)
Pupil (51,48) (51,49.5) // (77,48) (77.5,50)

Here's the geometry before solving:



Here's the voltage after solving:



Note that many of the sharper features do not translate into sharp fields in concave corners. On the other hand, we would see large field values at the tips of convex sharp structures, especially if we had a finer grid.

Implementation comments

Calculates the potentials in a four-sided box, on a unitary aspect-ratio rectangular grid.

The potentials on each wall are set to be constant, but need not be constant.

Aside from the challenge of numerical performance, one of the biggest time-soaks in making a modelling tool is handling the geometry input and mesh generation. It's no surprise that competing commercial tools use the same geometry library.

For simplicity - and to encourage you to get stuck in yourself - there are no such fancy graphical input tools with this code, but some convenience routines have been provided to get you started and show the way to automating other shaped fixed potentials.

Potentials can be specified within the domain as well, by adding an optional mask that is OR'd with the default update mask - although that particular implentation detail is hidden from the user via the fixV() method. Here's an example doUpdate array for a 3x5 region with potentials defined on all boundaries. FFFFF FTTTF FFFFF

Gotchas Trying to fix the potential in the corners throws an error because it won't affect anything (due to the way the rectange grid and cross-shaped stamp work - the stamp being the name to the update rule that we apply - the name coming from the idea of using a rubber stamp to highlight on a printed array which elements are used to update the element at the centre of the stamp) - here, X is updated using, N,S,E,W but not NW/SW/NE/SE. N W X E S

We don't define the size of the grid because it is not incuded in this static solution. This is an example of scale-invariance, although it only holds under the assumption that we are zoomed out enough we can't see the bumps of the individual atoms in our materials (i.e. macroscopic).