Integral Equations and Fast Algorithms (CS 598AK)

Homework Set 1

Due: September 5, 2013 · Out: August 27, 2013

Welcome to this semester's integral equations class! Remember that this class is about *you* and how much you learn. Please let me know what I can do to improve your experience.

This first assignment is meant to get you familiar with the mechanics of this class, from asking questions, to running code, to turning in homework. It'll also provide some practice with Python code. If you're unfamiliar with Python or the tools we are using, expect to spend a bit of time *now* to catch up. This will ensure that you won't be lost once we get to more complicated things.

After the description of the assignment, you will find a few pages that describe tools and procedures. Please make sure to read and follow those carefully.

Problem 1: Evaluate a layer potential slowly and inaccurately

Write a Python program to evaluate the double layer potential

$$D\sigma(x) := \int_0^{2\pi} \hat{n}_{\gamma(t)} \cdot (\nabla_2 G)(x, \gamma(t)) \sigma(t) \|\gamma'(t)\| \mathrm{d}t \tag{1}$$

directly using the ("uncorrected", i.e. textbook) N-point trapezoidal rule as quadrature, at an equidistant grid of $M \times M$ points covering the square $[-2, 2]^2 \subset \mathbb{R}^2$.

In the above, let G be the free-space Green's function for the Laplace equation in two dimensions,

$$G(x, x') := \frac{1}{-2\pi} \log(\|x - x'\|),$$

let $\hat{n}_{x'}$ be the unit normal to Γ at x', and let Γ be the closed curve given by the parametrization

$$\gamma(t) = \begin{pmatrix} \frac{3}{4}\cos(t - \pi/4)(1 + \sin(2t)/2) \\ \sin(t - \pi/4)(1 + \sin(2t)/2) \end{pmatrix},$$

with $\Gamma = \{\gamma(t) : 0 \leq t < 2\pi\} \subset \mathbb{R}^2$. The density σ will be specified below.

a) Plot the double layer potentials in the plane, on the square grid described above. Adjust the color scale so that the (smooth) behavior of the potential near the curve is visible. Note that in your initial plot, you will likely only see a few (spuriously singular) spikes near the curve.

   Submit a driver file `part-a.py` that takes three (command line) arguments N, M, and `density`. Recall that N is the number of source points on the curve, M is the number of target points (along one axis), and `density` is `"1"` for the constant density $\sigma(t) = 1$ and `"sin"` for the density $\sigma(t) = \sin(t)$. The driver file should then write the plot to a file whose name is given by `"part-a-{N}-{M}-{density}.png".format(N=N, M=M, density=density)`.

b) Study the convergence of the double layer potential at the point $(-0.5, -0.85)$ as you vary the number of points used in your trapezoidal rule.

   Compute a 'reference' value with a very high number of source points N_{ref} in your trapezoidal rule. Next, plot the error in a (properly labeled) log-log plot against N.

Submit a driver file `part-b.py` that takes two (command line) arguments N_{ref}, and `density`. Choose a range for N that shows the behavior from which you are drawing your conclusions to the question below. N and `density` are defined as above. N_{ref} is the number of source points in the reference calculation relative to which the error is computed. The driver file should then write the plot to a file whose name is given by

```
"part-b-{Nref}-{density}.png".format(Nref=Nref, density=density).
```

Does the error decrease with any fixed order[1]? Why/why not? (Answer these questions in a comment in your main driver file for this part.)

c) Use the same technique of computing a reference solution using very many source points to turn the plots from part a) into error plots.

Observe that there is a region near the curve where the potential is computed inaccurately. Also note that the computed potential is almost entirely useless on or near the curve. Use a logarithmic color scale and tweak the scaling so that the error behavior near the curve is clearly visible.

Submit a driver file `part-c.py` that takes four (command line) arguments N, N_{ref}, M, and `density`. defined as above, but instead outputs the spatial error plot according to the format

```
"part-c-{N}-{Nref}-{M}-{density}.png"
```

Empirically, how does the size of the region of the inaccuracy scale with N? (I.e. to which power of N is its size proportional?) Is this answer consistent with your answer for part b)? (Answer this question in a comment in your main driver file for this part.)

Please submit all of the above files in the subdirectory '`problem-1`' of your git repository. (See below for more details on git.)

Hints:

- Many of your driver files will likely use a shared set of functions. Investigate Python modules and imports to avoid having to copy and paste this code.
- `numpy.linspace`, `numpy.mgrid`
- `matplotlib.pyplot.imshow`, `matplotlib.pyplot.show`, `matplotlib.pyplot.savefig`
- Use the following code to adjust the range of the color palette:

```
import matplotlib.pyplot as pt
im = pt.imshow(...)
from matplotlib.colors import Normalize
im.set_norm(Normalize(vmin=min_value, vmax=max_value))
```

- Do not forget ds/dt in the path integral.
- If you code this using Python `for` loops, it will be unbearably slow. Use `numpy` vectorization (or perhaps `cython`, if you dare) to get your results more quickly.

[1] `https://en.wikipedia.org/wiki/Rate_of_convergence#Convergence_speed_for_discretization_methods`

- Do not check the generated `*.png` files into `git`. See `man gitignore` for more information.

Problem 2: Iterated integral operators

a) Consider the two integral operators

$$F\phi(x) := \int_a^x k_1(x,y)\phi(y)\mathrm{d}y,$$
$$G\phi(x) := \int_a^x k_2(x,y)\phi(y)\mathrm{d}y.$$

The 'output' of the operator F is a new function $F\phi$. We may now apply the operator G to that function. Show that the combined operator $G(F\phi)(x)$ (or, if you know the 'composition' symbol, $G \circ F$) can be recast in the same form as the above two, i.e.

$$(G \circ F)\phi(x) := \int_a^x k_3(x,y)\phi(y)\mathrm{d}y.$$

Give an expression for the kernel k_3.

Find assumptions on the kernels k_1 and k_2 under which your derivation of the new kernel is valid. (These can be arbitrarily strong.) Under the assumptions you specify, justify the validity of your derivation as rigorously as you can.

b) Use induction and the result from the previous part to show that n-fold application of the operator

$$D^{-1}\phi(x) := \int_0^x \phi(y)dy$$

is the same as the single application of the following operator

$$\underbrace{(D^{-}1 \circ D^{-1} \circ \cdots \circ D^{-1})}_{n \text{ times}}\phi = D^{-n}\phi(x) = \frac{1}{(n-1)!}\int_0^x (x-y)^{n-1}\phi(y)dy$$

Check the slide 'Linear ODE to Volterra' in the lecture for context. (Lecture 3 according to current plans.)

In the subdirectory '`problem-2`' of your repository, please submit a PDF file `answer.pdf` with your solution. The following ways of creating the PDF are acceptable:

- (OK) Write your solution using LaTeX and run the file through `pdflatex`. LaTeX is installed in your virtual machine.
- (OK) Use `texmacs` or `lyx`, both of which are installed in your virtual machine.
- (Avoid) Scan a handwritten answer. (If you use this option, please keep the file size below a megabyte or so while making sure that your work is still readable.)

Virtual machine and homework instructions

Helpful Hints

Getting Help

If you are having technical trouble, if some instructions here fail to work for you, or if things just seem confusing, don't despair: There's plenty of help available.

First of all, please make sure you are subscribed to the class mailing list. Someone might have had the same problem as you (and ideally already solved it). If you're either officially signed up or have added your name to the sign-up sheet in class, you should already be on the list–you can tell by whether you've received a welcome email on Tuesday night. If you aren't subscribed, you may do so yourself at the list's web page[2]. The list has archives[3] where you can check if you've missed anything important.

Send email to

`inteq13@tiker.net`.

to post to the list. (Note that your message may be delayed if you don't use the same email address with which you signed up.)

If you would like to discuss a technical issue (i.e. one that isn't directly related to you), please do not email me directly–I'll just tell you to ask on the list. So instead, please send a message to the mailing list, where I (and your peers) will be more than happy to assist. Thanks!

Again, welcome, and I hope you'll have an enjoyable and worthwhile experience in this class.

Note 1 *All work in this class will be UNIX-based. I've provided a virtual machine ('VM') that gives you easy access to just about all the software we'll need.*

First, install VirtualBox[4] and run it. Then, download the following machine image:

`http://tiker.net/debian-compute.ova`

The file is about 2 GB in size, so it'll take a while to download. If possible, grab it at school using a wired connection. If your browser didn't do this for you, you may have to rename the file so that it has a `.ova` file extension. If you're unable to download, ask me–I have a USB stick with the image and VirtualBox (the software that runs it).

Once downloaded, click "File — Import Appliance..." to import the VM into VirtualBox. This takes a few minutes, after which you can delete the file you downloaded.

Note 2 *Once the VM is done importing, you should change it to use the number of cores actually present in your machine. To do so, right-click the entry 'Debian Compute', then click 'Settings...', 'System', and 'Processor'. Then slide the 'Processor(s)' slider to the number of cores you'd like to use. Avoid the red area.*

[2] `http://lists.tiker.net/listinfo/inteq13`
[3] `http://lists.tiker.net/private/inteq13/`
[4] `http://virtualbox.org`

You can then start the VM by double-clicking on 'Debian Compute', and you'll be presented with a Linux desktop. You can make that desktop full-screen, if you wish. It should adapt itself to any window size you choose.

You can of course also use your own Linux (or OS/X) machine, but then you're on your own with respect to installing the software that we'll need.

Using Unix and the command line

While the VM provides a friendly point-and-click interface, we will be doing most of our work on the command line, which you can get by double-clicking the 'Terminal' icon on the desktop (and by many other ways).

While we hope you'll eventually agree that the command line is great for doing development work, it can be a bit intimidating at first.

Once you click the icon, you're greeted with a 'prompt':

```
owner@debian:~$
```

and a cursor next to it. `owner` is your user name, `debian` is the name of the VM. `~` is a shorthand for your home directory, and this spot in the prompt generally shows the current working directory you're in. `$` finally is the prompt itself.

Note 3 *We'll be a bit briefer from here on out. From now on, the dollar sign "`$`" represents the command prompt.*

If you are unsure of what to type next, consider taking a look at a friendly Unix tutorial[5] (recommended). (Googling for "unix tutorial" gives plenty more options.)

You will also need to edit plain text files. There are plenty of text editors that work in the terminal (nano, emacs, vim, all installed in the VM). If you've never used any of these, the command '`gedit`' in the VM provides provides a simple text editor that's not scary and works a bit like Microsoft Word. You can also start that editor on a specific file from the prompt:

```
$ gedit filename.py
```

(Remember that you don't have to type the `$`.)

Also, note that you will get a bash[6] shell by default. If you are used to the C shell, ask me on how to change that. (If this was mumbo-jumbo to you, forget about it.)

Using `git` for Homework and Collaboration

I expect you to use the distributed version control[7] system git[8] for developing and turning in solutions to your homework. Like compilers, debuggers, and build tools, version control systems are central to most present-day software development. By having you use `git`, I hope to be able to give you some familiarity with such tools.

[5] `http://www.ee.surrey.ac.uk/Teaching/Unix/`
[6] `http://en.wikipedia.org/wiki/Bash_(Unix_shell)`
[7] `http://en.wikipedia.org/wiki/Distributed_revision_control`
[8] `http://git-scm.org`

Along with the tool itself, we will be using a central collaboration space, `http://gitlab.tiker.net`, which works like the big, public development hubs Github, Google Code, or SourceForge. You will receive an account there and create a new "project" for every assignment (and your final project). The collaboration space provides the following functions in our class:

- I will grade what is on `gitlab` after the homework deadline. As a result, you definitely want to upload your finished assignment.
- It will allow you to collaborate on your final project with a friend. (if you choose)
- If you are having an issue with your current code, you may upload it and ask me to view it on `gitlab`.
- It provides an easy conduit to get code from one machine onto another and back while preserving changes made on both.

Note 4 *Your account on `gitlab` is initially limited to about 50 megabytes, with more available if needed. Please do not check in large data files before discussing your needs.*

Your accounts on `gitlab` and the data you store there will be removed on February 1, 2014. Please back up your code and data before this date.

Git is preinstalled on our virtual machine.

To start, open a terminal (use the icon on the VM desktop) and inform `git` of your name and email address:

```
$ git config --global user.name "Your Name"
$ git config --global user.email you@yourdomain.example.com
```

To access the class-wide collaboration space, go to `http://gitlab.tiker.net` and enter the user name and password you received. If you haven't received this information, please email me (andreask@illinois.edu). You can use the web browser in the VM to access `gitlab`. Make sure to check 'remember me' so that you don't have to keep entering your password.

Note 5 *The emails sent by `gitlab` have a habit of getting flagged as spam by a variety of spam filters.*

Next, type

```
$ ssh-keygen
Generating public/private rsa key pair.
# Just hit Enter for this prompt.
Enter file in which to save the key (/home/owner/.ssh/id_rsa):
Created directory '/home/owner/.ssh'.
# Pick a password here, or leave empty if you're feeling lucky.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/owner/.ssh/id_rsa.
Your public key has been saved in /home/owner/.ssh/id_rsa.pub.
The key fingerprint is:
df:0d:72:0b:19:f6:c4:ec:10:8b:0d:45:69:84:55:0a owner@debian
```

```
The key's randomart image is:
+--[ RSA 2048]----+
|       E**o.     |
|       .=o*      |
|       ..B +     |
|        . B      |
|       S + =     |
|        . = +    |
|         . o .   |
|                 |
|                 |
+-----------------+
$ cat $HOME/.ssh/id_rsa.pub
```

(Again, don't type the `$` signs...) This will print a few lines (technically, just one wrapped line) that looks like this:

```
ssh-rsa AAAABw...vM3XdIbZWwmXH/iNbFWEhZw== owner@debian
```

Once you are logged into `gitlab`, click the 'user' symbol in the top right corner. Click "Edit Profile" on the top right, then "SSH Keys" in the top row, then "Add SSH Key". Finally paste the line returned above into the field that says "Key". You can pick an arbitrary title, perhaps "Key from virtual machine". If you get errors, make sure that there are no explicit newlines in the key. Click "Add Key". The key update may take up to two minutes to fully complete, so if the '`git push`' below fails, wait for two minutes, and then try again. This completes the setup steps that are only required once.

Next, create a directory (on the VM) for your homework submission.

```
$ mkdir hw1
$ cd hw1
$ git init
Initialized empty Git repository in /home/owner/hw1/.git/
$ git status
On branch master

Initial commit

nothing to commit (create/copy files and use "git add" to track)
```

The '`init`' command told `git` that you would like it to manage this directory. Now go ahead and start on your project. Once you've gotten to a point that you would like to save, say

```
$ git add file1.py file2.pdf
$ git commit
```

It is easiest to `git add` one file at a time, from the directory in which the file resides.

Note 6 *You* can *add entire directories at once, but this is often a bad idea. The process typically picks up lots of files (binaries, backups, output, data...) that you did not mean to check in. If you add some of this stuff by accident, you can get rid of it by* `git rm filename`.

Upon entering `git commit`, `git` will ask you for a commit message in a new `gedit` window. (On other machines, this might drop you into `vi` or some other horror.)

Note 7 *You should not check in backups (ending in `~`), executable or binary files such as those ending in `.o` or `.pyc` or ones called `a.out`. These can easily be rebuilt from the source code and therefore do not need to be saved.*

After your first commit, let's save your repository to the class collaboration space. To do this, click on the '+' icon on the top right ('Create new project'). You may choose a name for the new repository. Make sure this name has no spaces and only dashes as special symbols. (similar to what you might pick for a filename)

Note 8 *To make sure I can find your homework, please choose this name following the pattern "`inteq13-hw1`". If you don't, I won't find your submission!*

If necessary, you may later add co-owners and collaborators by their NetID. (Note that you're free to discuss homework with your peers, but you *must* write your own code. In other words, collaboration will likely only become relevant once we get to the final project.) Once you click "Create Project", your repository will be created on `gitlab`. To establish the link with your local one, say

```
$ git remote add gitlab ssh://git@gitlab.tiker.net/<netid>/<name>.git
```

You may then say

```
$ git push gitlab master
# If you entered a password in ssh-keygen above, you'll be asked
# to enter it again now.
Counting objects: 34, done.
Delta compression using up to 8 threads.
Compressing objects: 100% (21/21), done.
Writing objects: 100% (34/34), 8.25 KiB, done.
Total 34 (delta 12), reused 30 (delta 11)
To ssh://git@gitlab.tiker.net/andreask/inteq13-hw1.git
 * [new branch]    master -> master
```

to upload your commits.

You may also navigate to `http://gitlab.tiker.net`, pick your project from the list, navigate to the "Files" tab and verify that the right version of the code is checked in.

From here on out, the cycle of add/commit/push just repeats. Note that you need to `git add` a file every time you've changed it.

Here are some more resources about git:

- An interactive practice session at `http://try.github.io`.
- A video explanation, by yours truly: `http://bit.ly/git-ak`. (Watch with Google Chrome)
- The 'official' git tutorial[9].

[9] `http://www.kernel.org/pub/software/scm/git/docs/gittutorial.html`

You should at least be familiar with the usage of the commands

- `git status`
- `git add`
- `git commit`
- `git log`
- `git diff`

One piece of advice: Commit frequently–at least once for every milestone you complete along the way. This allows you to go back to a working version or see what changes you've made since the last milestone.