Numerical Analysis (CS 450)

Homework Set 1

Due: February 12, 2014 · Out: January 29, 2014

Welcome to this semester's numerical analysis class! Remember that this class is about *you* and how much you learn. Please let us (Andreas, Sweta, Kaushik) know what we 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: (15 points)

Consider the function $f : \mathbb{R}^2 \to \mathbb{R}$ defined by $f(x, y) = x - y$. Measuring the size of the input (x, y) by $|x| + |y|$, and assuming that $|x| + |y| \approx 1$ and $x - y \approx \varepsilon$, show that $\operatorname{cond}(f) \approx 1/\varepsilon$. What can you conclude about the sensitivity of subtraction?

What the problem means to say by '$x - y \approx \varepsilon$' is that the difference between x and y is small, where ε is that small difference. To compute the condition number, you'll be examining a perturbation $(x + \Delta x, y + \Delta y)$ of the input (x, y). The size of the perturbation is independent of ε, but also assumed small, so that you can assume that $(x + \Delta x) - (y + \Delta y)$ is also small, but not necessarily equal to ε

To simplify the analysis, the problem lets you also assume that $|x| + |y| \approx 1$. Since we assume $|\Delta x| + |\Delta y|$ is also assumed small, you may also assume $|x + \Delta x| + |y + \Delta y| \approx 1$ if needed.

(continued on next page. . .)

Problem 2: (15 points)

For computing the midpoint m of an interval $[a, b]$, which of the following two formulas is preferable in floating-point arithmetic?

(i) $m = (a + b)/2.0$

(ii) $m = a + (b - a)/2.0$

Why? When? (*Hint*: Devise examples for which the "midpoint" given by the formula lies *outside* the interval $[a, b]$.)
In addition, provide the following in your writeup:

(a) Specify the floating point system you are using.

(b) Specify values of a and b in floating-point representation (i.e. if you choose a base-2 system, your numbers should be binary) within your FP system.

(c) Write down all intermediate results for both cases in your chosen floating-point representation.

(d) Point out the steps where the problem occurs.

Providing one example suffices. The example you provide should support the point you are making about which formula is preferable and why.

(*For further study:* Entire articles (e.g. this one[a]) have been written on this problem.)

[a] `http://scholar.google.com/scholar?cluster=17564270959767398386`

(continued on next page...)

Problem 3: Bessel recurrence vs. floating point (20 points)

The Bessel functions[a] $J_n(\cdot)$ obey the recurrence relation

$$J_{n+1}(z) = (2n/z)J_n(z) - J_{n-1}(z). \qquad (1)$$

This means that, given $J_0(z)$ and $J_1(z)$ for a fixed $z \in \mathbb{R} \setminus \{0\}$, $J_n(z)$ for integer $n \geq 2$ can be computed–at least in theory.

You can evaluate Bessel functions as $J_n(z) =$ `scipy.special.jv(n, z)`.

(a) Write a program that tests the accuracy to which the values returned from `scipy` obey (1).

Print these values for each n, and also print difference between left and right hand side for each n.

(This should yield roughly machine precision in each case.) (5 points)

(b) Write a program that uses the values for $J_0(z)$ and $J_1(z)$ (obtained from `scipy`) and the so-called 'recurrence relation' (1) to compute the values of $J_2(z), \ldots, J_{50}(z)$.

Print these values for each n, and also print the relative error compared to the value returned by `scipy` and report your results.

For your experiments, fix $z = 20$. (5 points)

(c) You should find that the results from (1) rapidly start losing precision around $n = 30$.

Identify the reason for this loss of precision. (5 points)

(d) Observe that (1) can be rearranged to compute J_{n-1} from J_{n+1}:

$$J_{n-1}(z) = (2n/z)J_n(z) - J_{n+1}(z). \qquad (2)$$

Do you believe using (2) to compute $J_0, \ldots, J_{48}$ from J_{49} and J_{50} will encounter loss of precision? Why? (Run the experiment if you are not sure!) (5 points)

[a] `http://dlmf.nist.gov/10`

(continued on next page...)

Problem 4: Gaussian elimination and partial pivoting (25 points)

(a) Write a program implementing Gaussian elimination with no pivoting. (10 points)

(b) Write a program implementing Gaussian elimination with partial pivoting. (10 points)

(c) Test your code. For each of the following examples, report the following in a table:

- the condition number (`np.linalg.cond`)
- the error from un-pivoted solve (a)
- the residual from un-pivoted solve
- the error from partially-pivoted solve (b)
- the residual from partially-pivoted solve
- the error from `np.linalg.solve`
- the residual from `np.linalg.solve`

Also report if one of the solves failed. Write your code so that it prints this table automatically, without user interaction.

For each example, write two or three sentences on your observations in regards to conditioning, the necessity of pivoting, and the reasons for your observed results.

Use the following examples of size $n = 100$ to test your code:

1. a 'random' matrix (from `np.random.randn`),
2. the matrix given by
$$A_{ij} = \begin{cases} 5 & (i+2) \bmod n = j, \\ 10^{-3} & \text{otherwise}, \end{cases}$$
3. the matrix given by
$$A_{ij} = \frac{1}{(1+|(i+1) \bmod n - j|)^4}.$$

In each case, generate random vector $x^* \in \mathbb{R}^n$. Now, compute b as the matrix-vector product $A\,x^*$ and solve the linear system $A\,x = b$. (5 points)

Coding suggestions: You may find the following functions useful for this problem: `np.array`, `np.dot` and `np.random.rand`, `np.linalg.norm`, `np.roll`, `np.fill_diagonal`.

(continued on next page...)

Problem 5: Scaling a linear system (25 points)

Multiplying both sides of a linear system $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ by a nonsingular diagonal matrix $\boldsymbol{D}$ to obtain a new system $\boldsymbol{D}\boldsymbol{A}\boldsymbol{x} = \boldsymbol{D}\boldsymbol{b}$ simply rescales the rows of the system and in theory does not change the solution. Such scaling *does* affect the condition number of the matrix and the choice of pivots in Gaussian elimination. As a result, it may affect the accuracy of the solution in finite-precision arithmetic.

Using a linear system with the following code to set up $\boldsymbol{A}$:

```
n = 100
np.random.seed(0)
A = np.random.randn(n, n)
```

$\boldsymbol{x} = \begin{bmatrix} 1 & 1 & \cdots & 1 \end{bmatrix}^T$ and $\boldsymbol{b} = \boldsymbol{A}\boldsymbol{x}$ experiment with the following scaling matrices $\boldsymbol{D}$ to see what effect they have on the condition number of the matrix $\boldsymbol{D}\boldsymbol{A}$ and the solution given by `numpy.linalg.solve` for solving the linear system $\boldsymbol{D}\boldsymbol{A}\boldsymbol{x} = \boldsymbol{D}\boldsymbol{b}$.

(a) No scaling.

(b) `D = np.diag(2*np.ones(n))`

(c) `D = np.diag(np.linspace(1, 100, 100))`

(d) `D = np.diag(np.linspace(1, 10000, 100))`

(e) `D = np.diag(2**-np.arange(-n//2, n//2, dtype=np.float64))`

Report the *relative residuals*, the *relative error*, and the *condition number* given by the various scalings. Which of the these scalings gives a very poor accuracy? Is the residual still small in this case? How does the condition number correlate with your observations?

By 'relative residual', the problem means $\|r\|/\|b\|$, where b is the right-hand side, and r is the residual. Use a norm of your choosing. When in doubt which to use, use the 2-norm.

Virtual machine and homework instructions

Helpful Hints

Submission guidelines

- All files should be submitted through Illinois Moodle. A link to Moodle is provided on the course website.
- The answer to each problem should be submitted as a separate PDF file.
- Important: Make sure to drop the PDF parts of your submission into the 'PDF' bin (the bottom one) in Moodle.
- Each problem should have a clear and concise write-up.
- Refer to any Python code that you write and submit it as a separate file with a name starting with `problem`n_.
- All figures should be included in your submitted PDF.
- All code output should be clearly distinguished in your PDF.
- Modules should be imported as follows :

```
import numpy as np
import numpy.linalg as la
import scipy as sp
import matplotlib.pyplot as pt
```

  (Only import the modules you need, though.)

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 enrolled in the Piazza site. Someone might have had the same problem as you (and ideally already solved it). See this URL:

`https://piazza.com/class/hnwbbd0nykn6jm`

If you would like to discuss a technical issue (i.e. one that isn't directly related to you), please do not email us directly–We'll just tell you to ask on Piazza. So instead, please use Piazza, where we (and your peers) will be more than happy to assist. Thanks!

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

Setting up the class virtual machine

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

First, install VirtualBox[1] and run it. Then, download the machine image from the following page and follow the instructions:

`http://wiki.tiker.net/ComputeVirtualMachineImages`

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 (Andreas)–I have a USB stick with the image and VirtualBox (the software that runs it).

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 you'll need.

Using Unix and the command line

While the VM provides a friendly point-and-click interface, try to also get familiar with on the command line, which you can access by double-clicking the 'Terminal' icon on the desktop (and by many other ways).

We hope you'll eventually agree that the command line is great for doing development work, but 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 2. *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[2] (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 `$`.)

[1] `http://virtualbox.org`
[2] `http://www.ee.surrey.ac.uk/Teaching/Unix/`