Math 302 - Numerical Analysis
Recitation #1 appendix - LaTeX, MatLab & GPTs

Objective: This handout serves as supplemental material for recitation #1. It is just for your interest and not required in any form. Here, you will learn about different TeX compiling environments available to you and practice using them. You'll also set up MatLab and get familiar with it through some examples. Last but not the least, you'll be prepared to "lead" GPTs. ~~You will need these so that you can complete course projects~~, and they're also useful if you plan do any research.

Complete as much of this as you can during recitation. If you run out of time, please complete the rest at home.

Note: The auto-magic power of VS code or JetBrains will not be here to help you. But it does not mean you need to remember every detailed command line we used ---- LLMs always ready to help you. But what you do need is understanding from the top level (to give clear instructions as "leader").

There's lots of IMPORTANT explanation here, for this course and beyond. Actual tasks you're asked to do are colored cyan for your convenience, but read everything. Thanks!

1. Your choice of computing environments

We'll use a variety of software in the course, and you have your choice of three ways to get work done. Each have their tradeoffs, and you'll become familiar with both of them in this recitation. Your choices:

- **Sever-like** (not yet available through this course)

  Whether you use Google Colab, a school cluster, or a rented server, high-performance computing resources can handle more intensive tasks and larger projects. These environments are typically well-configured and ready to use, allowing you to tackle substantial jobs with ease. However, they may be less user-friendly for those unfamiliar with them, and the queuing times or costs can sometimes be less than ideal. Give it a try, but don't worry if it doesn't work out—we're here to support you.

- **Local tools** on your own computer

  With the right software and setup, your Windows, Mac, or Linux machine can handle most tasks you'll need for this class. While the setup process might take some time, it provides the best user experience since you're not relying on a network. However, please keep in mind that local computational resources may sometimes be insufficient for certain tasks.

If you anticipate working on larger projects, it's advisable to have an alternative solution ready. *Note: support for this option is best-effort, as computing system vary, and not every member of the teaching staff is familiar with every system.* (As long as it's not something like a Raspberry Pi😅, we can likely make it work for you.)

First, we'll get the local tools working and come back to server-like later on if necessary.

For both approaches, we will be using *git* source code control to both back up our data, track its changes, and smoothen collaboration.

2. Command Line (Terminal) warm-up

Poke around your environment, then open up a terminal (iTerm or zsh preferred). You'll pick up more in Homework 2 if you are going to run it on server-like environment, but for now, try out these commands.

Useful Commands

- `ls`: list directory contents
- `pwd`: print name of current/working directory
- `cd path/to/directory`: change the working directory
  - Note, `~` is shorthand for the home directory. For example, the Desktop is at path `~/Desktop`.
- `cd ..`: go up one directory level
- `cp` *`src dst`*: copy files and directories from *`src`* to *`dst`* (use `-a` flag for directories)
- `mv` *`src dst`*: move and rename files by moving from *`src`* to *`dst`*
- `mkdir` *`dir`*: make directory *`dir`*
- `rm` *`filename`*: remove file *`filename`*
- `rm -r` *`dir`*: recursively remove directory *`dir`*
- `touch` *`filename`*: create file with name *`filename`*
- `cat` *`filename`*: print contents of file *`filename`* to the console
- `history`: print previous command

Two things you NEED to do on the command line to survive and thrive

1. **Tab completion**: You can use **tab** to complete directory paths and filenames. For example, try typing `cd ~/Desk` and hit tab. This will autocomplete the path as `~/Desktop/`. If the completion is ambiguous (e.g., "pot<TAB>" when there's "potato.jpg" and "potato.txt", it will complete as much as it can, then beep or flash. Hit tab again for a list of the choices, then type a few characters to disambiguate, and hit tab again. Tab completion is one of the best things ever. (If someone is watching you type and you don't use it, they'll get physically agitated.)

2. **Arrow history**: Use the **up arrow** to access recently used commands (and down arrow go to the other way, too). This can save a lot of time retyping long commands![1]

READ THE ABOVE AGAIN. PRACTICE DOING IT CONSTANTLY.
NEVER NOT TAB COMPLETE. NEVER RETYPE A COMMAND.
I KNOW THE FUTURE. YOU *WANT* TO PRACTICE THIS.

[1] If you want to be really efficient, you can hit Ctrl+R in the shell, then type part of a command you want to go back to. This is a reverse-search of your command history, basically searching the up arrow for a given string.

3. Introducing the local tools and *git*

With the current need we will start with compilers-like tools for LaTeX, s.t. all drafting, editing, and even presenting (with Beamer, to be mentioned later) can be done with LaTeX, which offers powerful typesetting capabilities for creating professional-quality documents with complex mathematical equations, bibliographies, and seamless formatting, making it ideal for academic and technical writing.

Additionally, we will be using GitLab to distribute templates and examples from recitations. You can fork an example (details later on) to get started. **Make sure that your forked project is private! Not doing so is considered a violation of the Duke Community Standard**. Once you fork an assignment, you can clone the project to your development environment, make changes, commit them and push changes back to GitLab. You are expected to regularly commit changes and push them to GitLab. (Your use of GitLab and the number of commits you choose to make, however, will not be graded.) Because this is a very easy way to keep up-to-date backups, corrupted or lost files will not warrant an extension for homework assignments.

NOTE: File corruption and loss is not a hypothetical scenario, it happens everywhere in this field. You should take backing up your work seriously.

Setting up the environment for LaTeX

Follow the table below, install what you needed (process didn't fully tested due to content creator's limited access to all the OS), **If you want a easier/faster way of writing, see "APPENDIX: Using Overleaf for LaTeX content creating" (HIGHLY RECOMMENDED)**:

Your OS	LaTeX Distribution	LaTeX Editor	PDF Viewer
Windows	MiKTeX	TeX Live	SumatraPDF
MacOS	MacTeX	TeXShop (included)	Skim
Linux (Ubuntu/Debian)	sudo apt-get install texlive-full	TeXworks (included)	sudo apt-get install evince

After installing your LaTeX distribution and editor, test the setup by **creating and compiling a simple .tex file**:

```
\documentclass{article}

\begin{document}

Hello, LaTeX World!

\end{document}
```

Compile the document in your editor (TeXworks, TeXShop, or VS Code) and ensure the output is a correctly formatted PDF. This setup will prepare you to work effectively with LaTeX on your local machine, giving you a powerful tool for academic writing and document preparation. **Recompile every time when you made any change!** ***Note**: If you think this stinks, you're right! See "APPENDIX*: **Using Overleaf for LaTeX content creating"** ***to write with a realtime preview and avoid this issue entirely.***

Git and GitLab (GitHub)

Git is a source code control tool that will allow you to track changes over time. GitLab is a central repository for Git projects; Duke has a deployment of it here: https://gitlab.oit.duke.edu/[2] You should click the button to sign in using Duke OAuth2.

[2] Note: You may find outdated documentation referring to "coursework.cs.duke.edu" – ignore that in favor of

First, we need to make sure git is setup properly in your environment. **Open the terminal and enter the following commands, replacing <NetID> and <Your Name> appropriately**:

```
git config --global user.email "<NetID>@duke.edu"

git config --global user.name "<Your Name>"
```

We now need to set up SSH keys so you can access GitLab. An SSH key is a cryptographic pair of data files called the "public key" and "private key"; these files are mathematically related. We provide the public key to Gitlab, then we can use our corresponding private key to login to Gitlab in the future. Don't sweat the details – the tools do most of this for you[3]. **To make a pair of keys, run the following command in the terminal, replacing <NetID> appropriately**:

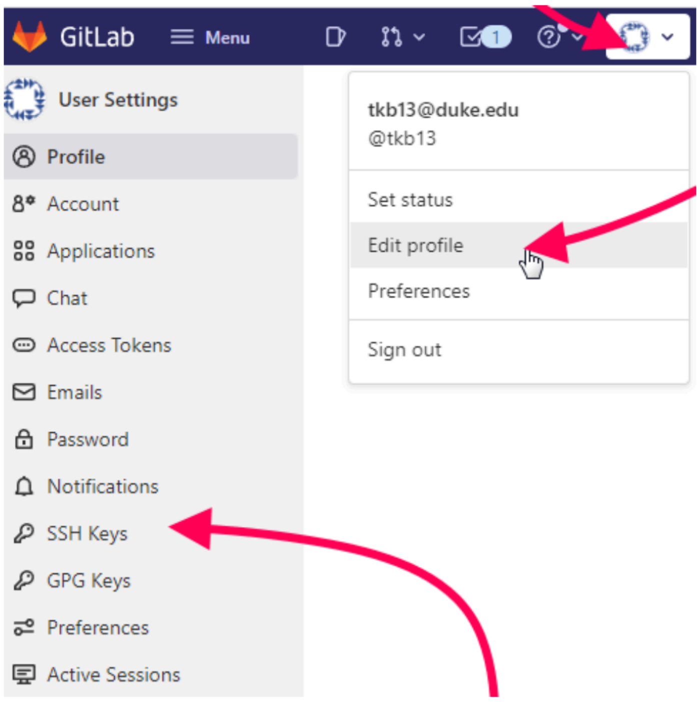
[3] You can learn all about this sort of thing by taking an intro computer security class, such as CS 351 or ECE 560.

```
ssh-keygen -t rsa -b 4096 -C NetID@duke.edu
```

Press enter when prompted to enter a path to save the SSH key and also bypass adding a passphrase by pressing enter. Now run:

```
cat ~/.ssh/id_rsa.pub
```

and copy the output. This is your public key. **Navigate to https://gitlab.oit.duke.edu/ and sign in using the "Duke Shibboleth Login" option. Click on the profile icon in the upper right corner and select "Edit profile". Now choose SSH Keys on the left sidebar. Paste your SSH public key and give it a descriptive title such as "CS 521" and click "Add key".**



NOTE: If you are familiar with SSH and know that you have already created an SSH key pair on your computer, you do not have to make a new one. Feel free to copy your public key (*.pub) using the cat command and paste that into GitLab. Ask your TA if you are unsure of whether you have an existing SSH key pair.

NOTE #2: If you are wondering whether your GitLab is set up with SSH keys correctly, you can run the following test command (case-sensitive, the -T must be capital)

```
ssh -T git@gitlab.oit.duke.edu
```

The first time you run this command, you may encounter a prompt like the following:

```
The authenticity of host ****** can't be established.
ECDSA key fingerprint is SHA256:********.
Are you sure you want to continue connecting (yes/no/[fingerprint])?
```

You will need to type "yes" at this prompt and press ENTER. This prompt will appear **regardless of whether your SSH is set up correctly or not**, but you have to type "yes" in order to continue.

If SSH is set up correctly, then the test command should produce the following output:

```
Welcome to GitLab, @netid!
```

where netid is your NetID. If instead, you get a prompt for "Password: ", then SSH is not set up correctly. You can use CTRL-C to exit this prompt. Then, try to upload the correct SSH key to GitLab and run the test command again.

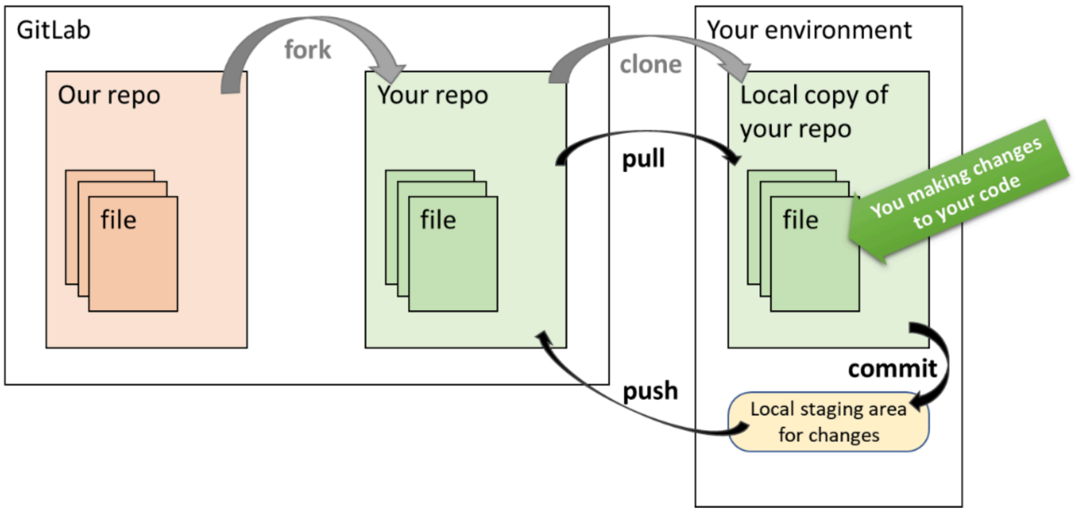
When SSH is set up correctly, you should not need a password. Ask your TA for help if you aren't sure why you didn't get the "Welcome to GitLab" message.

Helpful Git Commands

- `git add` *`filename`*: **add file** *`filename`* **to stage for commit**
  You'll do this for all your source code files, e.g. *`byseven.c`* ***in Homework 1!***
- `git status`: **display the state of the working directory and the staging area**
- `git commit -m "`*`MESSAGE`*`"`: **commit stages changes with the commit message** *`MESSAGE`*
- `git push`: **push changes upstream (to GitLab)**
- `git pull`: **pull changes from upstream (from GitLab)**

There are many online resources on how to use Git. Git is very powerful and has many cool features. For this class the simple commit and push workflow should be sufficient but if you want to dive deeper into Git try using branches to work on features and merging those features into the master branch! The Git documentation is a good place to start.

The diagram shows the major steps involved in git. One-time steps are shown in grey, whereas steps you do repeatedly during development are in black.



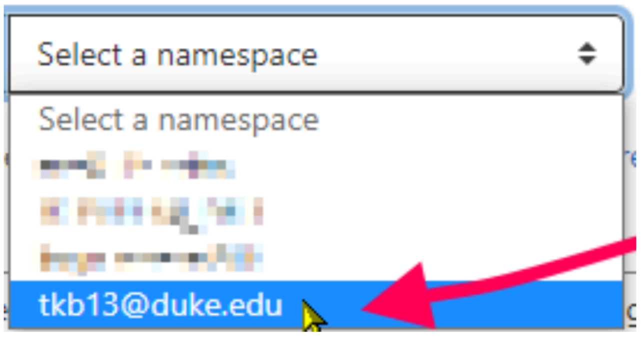
Your First GitLab Assignment

Git step 1: Fork

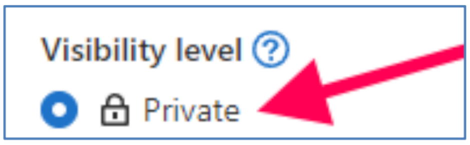
Now let's fork a project (aka repo) from the "CS521_24FA" GitLab group, clone the forked project, make changes, commit those changes and push the changes to GitLab. This is the exact same workflow you will use to fork recitation examples and templates, make changes, and backup these changes on GitLab.

Navigate to https://gitlab.oit.duke.edu/cs521_24fa and log in and request access if needed. Here you should click on a project titled "recitation1-test". If you have trouble finding the project, this link will take you right there.

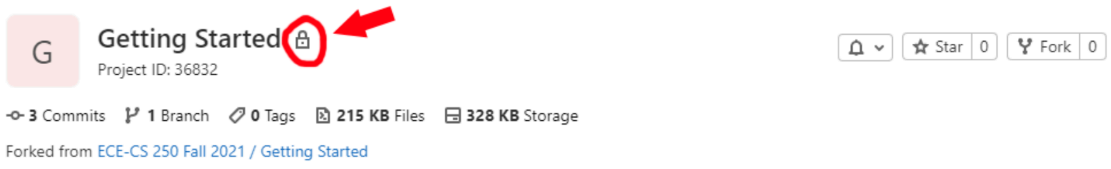
On the upper right click "Fork" to fork the project and, under "select a namespace", pick your NetID:



Make it private:



You should see the 🔒 icon next the project name if it is private:



If the project is not private, navigate to Settings > General on the left side bar. Then expand the "Visibility, project features, permissions" section and change the "Project visibility" to "Private".

Make sure to fork a project before making changes. DO NOT clone and make changes before forking a project since you will not be able to push changes.

ALWAYS MAKE SURE THE FORKED PROJECT IS PRIVATE! Make sure to make it private if it is not already.
Not doing so is considered a violation of the Duke Community Standard.

Git step 2: Clone to your local environment

Now click on "code" in the upper right and copy the link for "Clone with SSH" to clone the project. Open the terminal, navigate to the Desktop (run cd ~/Desktop) or where ever you want to store this code, then run:

```
git clone <PASTE_LINK_HERE>
```

This command will clone the project to your environment. Navigate to your local copy of the project. Use *ls* to see what's there.

Git step 3: Mess with some code!

Let's compile *homework-1-analysis.tex*

```
pdflatex homework-1-analysis.tex

xelatex homework-1-analysis.tex

lualatex homework-1-analysis.tex
```

Git step 4: Commit and push changes

Lastly, and **importantly**, let’s commit this change and push it to GitLab so there’s a backup online. **From the repo directory, run:**

```
git add welcome.c    #Adds the changed file to staging

git commit -m "now it’s very cool"   #Commits change locally

git push   #Pushes change to remote gitlab repo
```

Go to the repo on GitLab. You should see the changes reflected there!

4. Get hands dirty with MatLab and GPTs

https://www.mathworks.comGo to the link and try set it up yourself. It should be smooth.

DKU student shall register with non-DKU nor Duke email address and then change into @duke.edu email to receive the license.

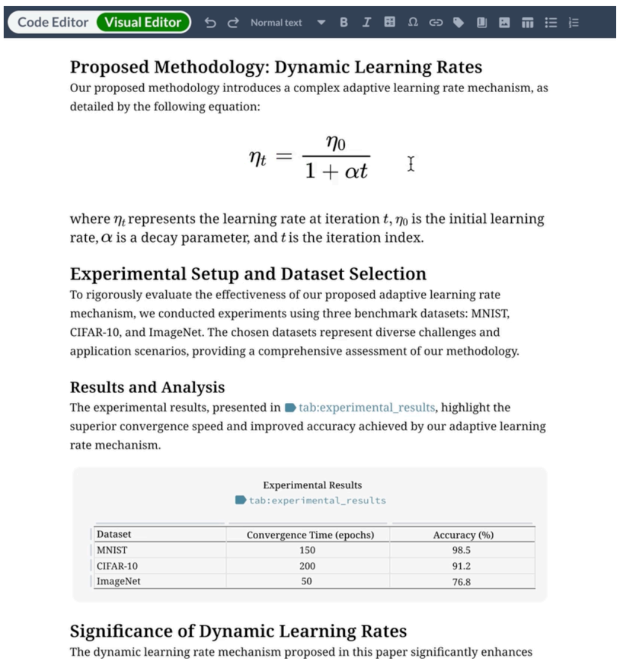
https://chatgpt.comGo to the link and try set it up yourself. 4-o is free with limited number of responses (should be enough). Some other language model you could look into: Copilot, Ollama, Github Copilot...

Appendix. Using Overleaf for LaTeX content creating

https://www.overleaf.com

LaTeX, simplified

- **Visual Editor and Code Editor**
  Switch to our Visual Editor to edit without writing code. Switch back to Code Editor anytime to see the code behind your writing.
- **Tables and figures with the click of a button**
  Whether you're working in Visual Editor or Code Editor, our toolbar lets you add figures and tables, and easily format text.
- **Templates for everything**
  Our range of free templates and example projects mean you don't have to start from scratch.
- **No setup, installation, or package management**
  Our online editor is setup and ready to go the minute you log in.



Proposed Methodology: Dynamic Learning Rates

Our proposed methodology introduces a complex adaptive learning rate mechanism, as detailed by the following equation:

$$\eta_t = \frac{\eta_0}{1+\alpha t}$$

where η_t represents the learning rate at iteration t, η_0 is the initial learning rate, α is a decay parameter, and t is the iteration index.

Experimental Setup and Dataset Selection

To rigorously evaluate the effectiveness of our proposed adaptive learning rate mechanism, we conducted experiments using three benchmark datasets: MNIST, CIFAR-10, and ImageNet. The chosen datasets represent diverse challenges and application scenarios, providing a comprehensive assessment of our methodology.

Results and Analysis

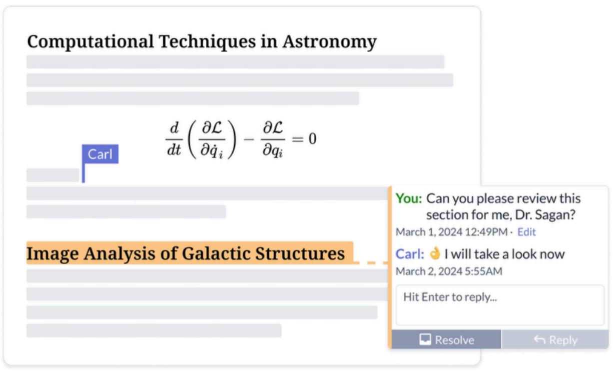
The experimental results, presented in tab:experimental_results, highlight the superior convergence speed and improved accuracy achieved by our adaptive learning rate mechanism.

Experimental Results
tab:experimental_results

Dataset	Convergence Time (epochs)	Accuracy (%)
MNIST	150	98.5
CIFAR-10	200	91.2
ImageNet	50	76.8

Significance of Dynamic Learning Rates

The dynamic learning rate mechanism proposed in this paper significantly enhances



Computational Techniques in Astronomy

$$\frac{d}{dt}\left(\frac{\partial \mathcal{L}}{\partial \dot{q}_i}\right) - \frac{\partial \mathcal{L}}{\partial q_i} = 0$$

Image Analysis of Galactic Structures

You: Can you please review this section for me, Dr. Sagan?
March 1, 2024 12:49PM

Carl: 👌 I will take a look now
March 2, 2024 5:55AM

Seamless collaboration

- **Project sharing**
  Share your project via a link or add collaborators directly by email.
- **Simultaneous editing, commenting, and chat**
  Multiple people can edit and comment on shared LaTeX documents simultaneously and take advantage of in-document chat to discuss changes as they go.
- **Real-time track changes** ★ Premium
  See your collaborators' edits in real time and easily identify what they've changed.
- **Project history and version control** ★ Premium
  Easily review changes to your project over time. Compare, label, and download different versions.

Have fun! 🎉