(This document will be updated from time to time. Bookmark and check again later. Also feel free to contribute to the document by suggesting changes!)

General advice

- https://yyiki.org/wiki/Research%20advice/
- U. Alon, How to choose a good scientific problem
- The Beginner's Creed by Peter J. Denning:
- http://managingbias.fb.com - Managing unconscious bias. Valuable resource from Facebook about our unconscious biases.
    - https://implicit.harvard.edu/implicit/ - Take tests and recognize your biases!

- Merton's CUDOS:
    - **Communalism:** All scientists should have equal access to scientific goods (intellectual property) and there should be a sense of common ownership in order to promote collective collaboration, secrecy is the opposite of this norm.
    - **Universalism**: All scientists can contribute to science regardless of race, nationality, culture, or gender.
    - **Disinterestedness**: according to which scientists are supposed to act for the benefit of a common scientific enterprise, rather than for personal gain.
    - **Originality**: requires that scientific claims contribute something new, whether a new problem, a new approach, new data, a new theory or a new explanation.
    - **Skepticism (Organized Skepticism)**: Skepticism means that scientific claims must be exposed to critical scrutiny before being accepted.

- Don't be afraid to discuss your career, ask for the reference letter, and other stuff outside research. I'm happy to chat!
- If you aim to stay in academia it is a good exercise to maintain a research statement. It helps you to plan your career direction and to think about the relevance of your research in broader contexts. If you plan to go to industry, it is a good idea to have your product (software, visualization, …) portfolio on http://github.com.
- Michael Faraday said: "The secret is comprised in three words — **Work, finish, publish**." For me, "**finish**" is the most important word.
- Pay attention to the Social aspects of science
- How to follow current research?
- Participate in the writing group!
- Don't: Professor, Professor Ahn, Dr. Ahn, Professor YY / Do: YY

Meetings

Group meeting (2021 Spring): Wed 11am-12pm (https://iu.zoom.us/my/yyahn) - schedule

CX Reading group (): ?
Writing group (2021 Spring): Thursday 9:30am-12pm (https://iu.zoom.us/my/yyahn) - https://docs.google.com/document/d/1dFDc4wC8S5tANOx6aN1MjwUYytNkgV9Kuna4L-u3w7g/edit

http://yongyeol.com/group/calendar.html - You can add this calendar (+ button at the right bottom). Volunteer to present!

Other meetings you may want to attend (feel free to add):

- NaN meeting: http://cnets.indiana.edu/groups/nan/meetings
- Sporns lab / Betzel lab meetings (email them)
- CogLunch (cognoscente mailing list)

Lunch, tea, coffee, and beer. They are great opportunities to relax and talk about everything (and practice your english if you're not proficient enough). Great ideas often come with tea or beer.

You can check YY's travel schedule from here: http://yongyeol.com/travel/.

Communication

Mailing list: yy-l@indiana.edu

To subscribe, send an empty message to yy-l-subscribe@indiana.edu. It is an open mailing list. Share events, job postings, and other interesting things.

Group slack: https://yy.slack.com

Slack.com is a channel-based messaging + Information sharing service. You can think of it as a modern IRC. You can also download desktop & mobile apps. Feel free to peek into channels of your interest.

Computing

Computing servers

VSCode + Remote SSH provides an almost seamless environment (including jupyter). Try it out! Alternatively, you can use SSH tunneling: https://youtu.be/IUHmL1Ze2Ss (btw, if another process — likely another Jupyter process — already is using port 8888, a new Jupyter process will use 8889, 8890, and so on. When you run the Jupyter, it tells you the port number that it uses, or you can specify it.)

If your computation needs GPU/TPU (but not too much), the most convenient option may be Google Colab (https://colab.research.google.com/). You can use a decent GPU/TPU for free.

Group

There are mandatory usage guidelines before using any computing resources. You need to first read this: http://carl.cs.indiana.edu/cnets-howto/

- General purpose server: **snowball.cs.indiana.edu** (192GB RAM + Several TB Storage): to use it, send an email to sicehelp@indiana.edu (cc me) saying that you have read the cnets server usage guidelines and would like to use the snowball server.

The large storage is mounted at `/l/nx/`. If you want to have a home directory here send Rob an email. Put data into `/l/nx/data/`. If you have any special requests (e.g. db server, packages, etc.), send an email to Rob and me.

- GPU servers (Ember & Cinder): A large memory machine with 2x V100 and a small machine with 2x 2080Ti.

This was purchased for the Science Genome project. Because GPU resources cannot be shared easily, the access to this server will be prioritized case by case. In general the SG team members will have the priority, but feel free to request access.

The server backup schedule:
https://help.sice.indiana.edu/hwdb/search.php?searchstring=yyahn&search_all_fields=no&search_contacts=yes&backupreport=yes

School

1) gh.luddy.indiana.edu (named after Grace Hopper)

This is a general purpose login server that will replace the current tank (aka sharks). This is a good system to use for general login and less cpu/memory intensive jobs. The system has 36 CPU cores (72 virtual processors with hyperthreading) and 512GB of RAM.

2) kj.luddy.indiana.edu (named after Katherine Johnson)

This is a large memory compute server that will replace the current hulk. This is a good system to use for large, cpu/memory intensive jobs. The system has 48 CPU cores (96 virtual processors with hyperthreading) and 1.5TB of RAM.

3) al.luddy.indiana.edu (named after Ada Lovelace)

This is a GPU server that adds new functionality the school did not previously have generally available. This system should be used only for jobs requiring GPUs. The system has 4 x NVIDIA Quadro RTX 6000 GPUs as well as 20 CPU cores and 256GB of memory.

Campus

https://kb.iu.edu/d/anrf

Carbonate: https://kb.iu.edu/d/aolp
Big Red 200: https://kb.iu.edu/d/brcc

Security

It's becoming alarmingly easy to brute forcing your short passwords and there are always security holes you should be careful about. Some suggestions:

- Use long passwords. see http://xkcd.com/936/
- Use password managers such as **1password, lastpass,** and **dashlane** to manage long, site specific passwords.
- Use two-step verification, especially for services like Google.
- Always backup. ~~Today~~ Everyday is the International Backup Awareness Day.

Workflow

Version control and organizing a project

You are expected to publish data and code that is replicable. Please use https://github.com (or https://github.iu.edu but it tends to lag with features) for your projects. It is also possible to use dropbox or other tools, but I strongly encourage to use GitHub to store code, paper (overleaf is also fine), and small datasets (Data can be stored in Dropbox or in a server). Use Github's wonderful functionalities: pull requests, issue tracker, and project tracker. To learn git, see https://yyiki.org/wiki/Git/

When the project ends, **you should publish your code and data so that others can replicate your results**. Organize it carefully so that anyone can easily access it and reproduce our results.

Here's a sample git repo template for research projects: https://github.com/yy/project-template I'd be happy to explain the structure. Here is an example repository with notebooks and a full workflow: https://github.com/yy/sex-reporting

Text editor

Learn powerful text editors. In doing so, your productivity can keep increasing for years to come. If you don't have any strong preference yet, my default recommendation would be VSCode because of its awesome remote SSH plugin and the whole plugin ecosystem. If you think you're

a nerd, I recommend Vim. I use both VSCode and Vim ~~Request a demo~~, but there are other good options (Emacs, sublime text, etc.). Vim is particularly versatile since practically every machine has them. VSCode is super nice because of the Remote SSH plug-in that allows a seamless experience across local and servers.

Coding style

Maintaining a good style makes code easier to read and fix. For Python, read PEP8, install Black (and other linting tools) and integrate it into your editor so that it automatically formats your code whenever you save the file. You can check out this video: Linters and fixers: never worry about code formatting again (Vim + Ale + Flake8 & Black for Python).

Workflow

It is a good idea to learn build tools (e.g., GNU make[1], Snakemake[2]). We usually use Snakemake. Also see "An efficient workflow for reproducible science"

I highly recommend using Jupyter notebook (now JupyterLab) for experiments, where you can maintain documentation, code, and results in one place. At the same time, try to package your code so that we can test and reuse.

You want to spend more time maintaining & cleaning your existing code than what you may think. Everyone thinks "oh I'll just use this once so I'll just quickly code it up and not waste my time". But soon, you'll find yourself in a big mess of spaghetti code. Spending time to create clean code pays off in most cases.

Writing text

Participate in the writing group!

Plain text formats are preferred when writing documents (e.g. Markdown or LaTeX). If you use LaTeX, Latexmk is a pretty convenient tool for automatically monitoring and compiling tex files. VSCode has a powerful LaTeX plugin. Plain text works better with version control systems and is highly accessible with all kinds of tools. Collaborative platforms such as Google Docs or overleaf are good alternatives.

Maintain a summary document that contains method & data descriptions and results (I call it "mini papers"), to keep track of your research progress. It's important to write down for many reasons. First, it will help you correctly remember later how you did your work. Second, it will be crucial for putting the whole team on the same page. Without a written document, your team will not really get the details of what and how you have done the work.Third, it makes it easier to communicate with others and recruit collaborators.

---

[1] A nice use of GNU make in research: http://www.narrykim.org/s/park-dicer-2011/
[2] An example from YY: https://github.com/yy/sex-reporting

Some tips about research workflow

- **Know what you're doing and why**: Always ask yourself: what is the main question that you are trying to address? What is the main result of the paper? Can you summarize it in a single sentence? Why is your problem important in the broader context?
- **Be obsessive about robustness**: Always ask yourself: can this be an artifact or a result of systematic biases? What could produce such artifacts? Think of a situation of retracting your paper because someone discovers such systematic biases or an error.
- **Start small**: Start with small examples (or small samples of data) that you can exactly understand every detail and can iterate quickly. Make sure everything is correct and then scale up.
- **Test your code**: Create tests for your code (e.g., use https://docs.pytest.org/en/latest/). Whenever I write a test the following happens: I first feel resistant because it feels like a waste of time. Then I realize that it catches so many errors right away and remember that this is why I want to write tests.
- **Document, document, and document**: always write down details about data and methods. Where did the data come from? How did you process the data? What are the parameters for the simulation? What exactly did you do? Make sure to make everything reproducible (even the manual curations, by creating a data file of curation operations and script). Documentation is automation.
- **PEPKAC**

External brain

I strongly encourage you to cultivate a system to collect and organize information and thoughts. I recommend Obsidian, which just uses markdown files as its database. I use a personal wiki (https://yyiki.org) that is essentially a git repository with a bunch of markdown files. I access them locally with Obsidian.

Homepage

You need a homepage.

Put your CV on your homepage *even if you don't have any paper yet*. Periodically update your CV and homepage.

A good option: https://pages.github.com

Graduation

Proposal

See this guideline. Aim for 10-20 pages, but you can put already finished papers as chapters and treat it as a draft for the dissertation. The proposal document should clarify:

1. The *thesis* of the dissertation: what is the most central statement of your dissertation?
2. Why is it important? What is the background?
3. What has been done so far?
4. What will be done by the defense?
5. What's the timeline?

Reference Letter

1. You should first discuss with me before putting my name as a reference.
2. You should allow me about **one month** to write it. In other words, don't expect an awesome letter within a few days.
3. It is usually much easier for me if you provide me with some material, such as your CV, description of your projects, your roles in the projects, the points (qualities) that you want to emphasize, and other useful information.
4. You want to waive the right to view the letter.

There are some helpful articles about it.
http://yyahnwiki.appspot.com/Research_advices#h_949f0dc37da5a88f07d9f89096ab2320

Plagiarism and Misconduct

http://yyahnwiki.appspot.com/Plagiarism
https://www.indiana.edu/~academy/firstPrinciples/index.html

Plagiarism or misconduct may end your career (possibly mine too). Write from scratch. It's dangerous to write while looking at another paper. Making the project reproducible by ensuring data provenance and code publication is helpful to avoid any slippery slope.

It is always better to find errors than publishing them. It is always better to put erratum or retract a paper than not doing anything about it. If you find something's wrong it is never late to report it even after the paper is published.

Reproducibility and code sharing is not only for others but also for ourselves. Publish replicable code for every project.

Classes

As Matt Might wrote, an easy way to fail a Ph.D. is focusing on grades or coursework too much because the *grades become irrelevant* after graduation. What matter after graduation are what you have produced (papers for academia and papers/software for industry), what kinds of skills you mastered, and how others think about you. Courses should serve your research and career. Good grades should not be your objective. Use courses to initiate collaborations or learning & applying new methodologies.

yy is teaching mainly:

- I606: Network Science
- I422/I590: Data Visualization
- I709: Complex Systems Seminar II

Please share other useful courses (feel free to add):

- Machine Learning for Signal Processing (ENGR 511 by Minje Kim)
- Machine learning & Applied machine learning
- Network Neuroscience by Olaf Sporns

Teaching

Doing an AI (TA) requires significant time commitment so you will have less time for your research. However, I'd recommend doing some AIs even when funds are available for RAship, especially if you are interested in academic positions. Teaching experience is helpful to assess whether, and how much, you like teaching (although doing an AI may be different from being an instructor). Because most academic career paths involve teaching as a substantial part of the job, having some teaching experience is very helpful. Then, teaching a topic is a really good way to learn the topic because you really need to understand the topic to teach it. So if there is a course about a topic that you want to master, it is a good idea to serve as an AI. Finally, if you are doing an AI, try to excel. Having some teaching experience can be helpful for your job prospects, *especially if you can substantiate* your excellence in teaching in the form of awards and reviews, such as an AI award, a strong course evaluation, etc.

Recommended Books and Resources

Programming (Python)

http://yyahnwiki.appspot.com/Python

1. https://www.python.org/doc/ (Python official tutorials)

Networks

https://yyiki.org/wiki/Network%20science/

Information theory

1. MacKay's information theory book: https://www.amazon.com/Information-Theory-Inference-Learning-Algorithms/dp/0521642981 and lectures: https://www.youtube.com/playlist?list=PLruBu5BI5n4aFpG32iMbdWoRVAA-Vcso6

Statistics

https://yyiki.org/wiki/Statistics/

1. Statistics for Hackers - https://speakerdeck.com/jakevdp/statistics-for-hackers - a super nice talk that can change how you think approach statistics problems.
2. An Introduction to Statistics: http://work.thaslwanter.at/Stats/html/index.html - doing statistics with Python.
3. Think Stats - https://greenteapress.com/wp/think-stats-2e/
4. Think Bayes - https://greenteapress.com/wp/think-bayes/
5. Data Analysis: A Model Comparison Approach - https://www.amazon.com/Data-Analysis-Comparison-Approach-Second/dp/0805833889 - "Starts with a very simple "first principle", (data = model + error) then incrementally builds upon it, layering more complexity in each chapter, but always returning to this simple high level premise." (Quote by Benjamin Sugar)

Machine learning

https://yyiki.org/wiki/Machine%20learning/

1. http://mlreference.com
2. https://chrisalbon.com
3. An Introduction to Statistical Learning with Applications in R: http://www-bcf.usc.edu/~gareth/ISL/ - A very good & easy introduction
4. Machine Learning: A Probabilistic Perspective http://www.amazon.com/Machine-Learning-Probabilistic-Perspective-Computation/dp/0262018020/ - Grad level textbook
5. https://www.youtube.com/playlist?list=PLD0F06AA0D2E8FFBA Youtube video series by mathematicalmonk
6. https://remicnrd.github.io/the-machine-learning-cheatsheet/