The Future of Engineering Education
2024 Annual Conference & Exposition
Oregon Convention Center ASEE
Portland, OR . June 23 - 26, 2024
Paper ID #42581

A Collaborative Effort to Convert MATLAB-based Curriculum to Python in Undergraduate Biomedical Engineering Education

Dr. Elizabeth Kathleen Bucholz, Duke University

Dr. Bucholz is an Associate Professor of the Practice for the Department of Biomedical Engineering at Duke University and serves as the Director of Undergraduate Studies for the Department of Biomedical Engineering in the Pratt School of Eng

David Ward, Duke University

Title: Bridging the Gap: A Collaborative Effort to Convert MATLAB-based Curriculum to Python in Undergraduate Biomedical Engineering Education

Abstract:
In response to the evolving landscape of programming languages in the field of biomedical engineering education, this abstract presents the outcomes of an innovative initiative aimed at transforming MATLAB-based classroom exercises, labs, and homework assignments into Python exercises. Spearheaded by a team of enthusiastic undergraduates and coordinated by a dedicated faculty member over the summer, this conversion project was undertaken to ensure alignment with contemporary industry demands, curricular uniformity that will allow for knowledge to build semester-to-semester, and enhance the educational experience for biomedical engineering students and provides a framework for others looking to perform similar conversions.

The scope of this endeavor encompassed all 11 required undergraduate biomedical engineering classes, across 24 different faculty members assisted by 12 undergraduate students. Courses that were part of the conversion effort included Signals and System, Modeling Cellular Systems, Instrumentation, Biomaterials, and more. Additionally, the initiative extended to cover a spectrum of junior-level track courses, such as Imaging, Biomaterials and Biomechanics, Cellular Engineering, Molecular Engineering, and Fluid Transport. By employing Python, a versatile and widely used programming language, the curriculum was not only modernized but also made accessible to a broader range of students, as our department worked to make the programming content more uniform across the curriculum.

This paper delves into the extensive collaborative process used, working across faculty and classes, highlighting the integral role played by undergraduates in the conversion efforts. Through the combined expertise of the faculty member and the students, a systematic approach was employed to meticulously transform MATLAB assignments into Python, ensuring the retention of educational integrity and pedagogical objectives. The challenges faced during this transition, ranging from technical intricacies to pedagogical considerations, are discussed, along with the innovative solutions devised to overcome these hurdles.

The successful conversion of these diverse engineering courses signifies a significant milestone in the evolution of Duke's BME engineering education, empowering students with a foundational understanding of Python programming while engaging them in real-world applications within their respective fields. This abstract serves as a testament to the collaborative spirit driving educational innovation, illustrating how the synergy between dedicated faculty and enthusiastic students can bridge the gap between traditional classroom practices and contemporary industry demands.

Introduction:

Biomedical Engineering stands at the nexus of innovation, where cutting-edge technology intersects with human biology to advance healthcare and improve quality of life. As this interdisciplinary field evolves, so too must the educational approaches that prepare students to tackle its challenges. A fundamental aspect of this preparation lies in programming proficiency, which serves as a vital tool for analyzing data, simulating systems, and developing solutions across various biomedical engineering domains [1]. Much discussion has been had around which programming language best prepares students for success in the biomedical engineering workforce.

In recent years, the programming landscape within BME education has witnessed a shift, mirroring broader trends in the medical technology and biotechnology industry [2].. MATLAB, long regarded as a staple in biomedical engineering classrooms for its robust numerical computation capabilities and user-friendly interface, has begun to share the spotlight with Python, a versatile and increasingly ubiquitous programming language particularly with the recent interest and investment in data science and machine learning. Python's popularity stems from its readability, extensive integrated libraries, low cost, and widespread adoption in industry, making it an attractive choice for educators seeking to align curricula with contemporary demands [3]. While in the past languages such as C, Java, and MATLAB were considered foundational, in the past 10 years Python has shown significant growth across many engineering disciplines, being labeled the top programming language in 2018 [4] as well as the most popular language of the year [5]. This year, Python is ranked #1 by the TIOBE index and has shown remarkable growth starting at 4.31% in 2010 to 9.35% in 2020, up to 14.82% in 2023 as compared to MATLAB's rank of 15 with an index score of 0.92% [2]. While there are many who argue that all programs should be language agnostic or varied and that computational fundamentals are universal and therefore adoption of a particular language is unimportant, we see a benefit in consistency of language throughout the program as this reduces the time students spend at the beginning of a semester re-learning the fundamentals of a new tool and gives them more time to actually program. The author has observed students with experience in the same language used in a course save 3-4 weeks ramping up on the language, tools, and libraries as compared with a student who has previously used the course language in a preceding course (this has held consistent for courses taught in Python and MATLAB). Additionally, our students have complained they are disadvantaged because their instruction used different languages and was missing deep study in a single language relevant to industry. Our introductory programming course switched to Python five years ago, and thus our students have even less preparation and familiarity with MATLAB than they had previously, this being a major impetus towards migrating the BME curriculum to Python. We recently reached out to a MedTech industry executive who hires biomedical engineering graduates who estimated that it takes 6 months for students to become proficient in a new language and having students already trained in Python would give them a boost in comparison to students with only MATLAB skills.

Recognizing the importance of staying abreast of these changes, our institution embarked on an ambitious endeavor to convert significant programming exercises and projects across all required undergraduate biomedical engineering courses from MATLAB to Python. This effort required both retraining of faculty, as well as effort in making the conversions as seamless as possible. This paper presents the outcomes of this collaborative effort, which involved a network of students, faculty, and staff.

Methods:

The Python conversion efforts of the BME curriculum began the summer of 2022, as a team of dedicated instructors met in cohorts to experiment with Python adoption and try different platforms as possible contenders to integrate into our curriculum. As part of this effort, the freshmen programming instructor who taught Python was brought in to consult on the best ways to scaffold Python instruction throughout our curriculum. As a test case, a signals and systems class was selected to be entirely converted in the summer of 2022 from MATLAB-based exercises to Python exercises. Through discussions with faculty, Jupyter Notebooks with embedded markdown text cells and code cells were selected as a uniform way to incorporate Python instruction across the department. Jupyter Notebooks have been selected in a wide variety of engineering classrooms as they combine the ability to give explanations like textbooks with the interactivity of a software applications [6] [7]. This allows laboratories to contain embedded text, with images, and tables to be integrated with coding boxes allowing students to step through programming assignments. An example of such a notebook is included in the Appendix of this paper. This first conversion exercise was performed entirely by the instructor of record with no Python background to judge the difficulty involved in making the planned departmental wide conversion the subsequent year. This course was selected to be a good test case as the course included 10 significant programming exercises and one comprehensive programming project and is largely seen as the foundational computational class for BME students following their initial introduction to programming class taken by all freshmen regardless of major. For this course, three of the four learning objectives contained a significant computational component that is satisfied by computational exercises. While the students would perform the math by hand, often the computer programming tool was brought in to demonstrate the applications that were possible. Specifically, the three learning objectives targeting computational exercises are reported below:

I. Formulate and solve linear differential equations representing engineering and biological systems.

II. Master time domain and frequency domain analysis of analog and discrete-time linear systems (convolution, Fourier transform, FFT, Laplace transform).

III. Demonstrate ability to design signal processing systems using principles of linear system theory.

The first version of the converted course was taught in a pilot to a smaller cohort of 9 students in the summer of 2022 to iron out any issues that might result from the conversion efforts. After some revision, including importing important libraries a week before they were needed in the laboratory exercises, the modules were then rolled out to a much larger cohort of around 50 students in the fall of 2022.

Following the first major class conversion, in the fall of 2022 two undergraduate students were hired to convert exercises from MATLAB to Python for the downstream course to be taught in spring 2023. This allowed us to test hiring students to make the conversions for faculty, the goal of which was to reduce overall faculty effort involved in the conversion and therefore increase faculty buy in and willingness to accept the language change, which had been the most significant hurdle in the past. Unlike the previous implementation, after the students performed the conversion to Python, the course was rolled out to students at scale without having the luxury of the smaller cohort class.

During the spring of 2023, discussions with the curriculum committee in our department resulted in naming Python as the language adopted by our department and the policy was set to require all required undergraduate biomedical engineering courses to use Python by the summer of 2024. To ease the conversion efforts, a strategic plan was created for the summer of 2023 to create a comprehensive picture of how much MATLAB was embedded in the curriculum and to connect faculty with students who would work over the summer to make the conversions happen. A Python consultant was brought in for the summer and a team of 8 undergraduate students were hired to span the diverse classes that required conversions. While adoption of Python was required for undergraduate courses, elective undergraduate and graduate level courses are still free to use the language of their choice but all faculty were given the option of support to make the Python conversion if they so chose.

During end of year course meetings conducted in May of 2023, the MATLAB content was acquired from all BME required courses and uploaded to a box site so the scope of the work could be determined. Students who had previously taken the required classes successfully, and with recommendations from the faculty members teaching the course, were then hired for the express purpose of converting all the collated MATLAB assignments, exercises, and labs into Python assignments for each faculty member. This study was approved as IRB exempt by Duke University Campus Institutional Review Board at a private university in the southeastern United States.

Results:

The investment by the original faculty member to convert course material over summer 2022 by themselves was substantive and based on that experience the recommendation was made to provide more support to faculty making the change to make it more palatable as such a conversion would require some faculty retraining. It was the opinion of that faculty member that had the department recommended all faculty convert their own materials it would be unlikely to yield a successful adoption of a new programming language across the curriculum. Since faculty buy in was the biggest obstacle to adoption, it was decided that support be provided to ease the transition in the form of these student hires to make the conversion efforts possible. The faculty member indicated that the conversion effort was not insurmountable, but at the same time significant, requiring iteration and practice for mastery.

Monthly Python club meetings during this time with other faculty members adopting Python were helpful during the conversion process.

The first iteration of the course to a small cohort was also successful at allowing one iteration to occur before the changes affected a large group of students. The most common issues were students not following instructions on Python installations and having package import errors. This was addressed in subsequent classes by completing Python library installations and importing important packages the week before the libraries were needed, allowing students additional time to fix the issue and allow them more hands on coding during class time. In addition, the smaller cohort helped the faculty member become more comfortable with debugging Python code and interpreting errors, which again, requires skill and practice.

For the subsequent work over the summer of 2023, the department hired a team of undergraduate seniors who were enthusiastic in assisting the transition to Python. The selected students were all students who had previously taken the courses they were planning on converting. Table 1 includes the course subject, the number of students hired, along with the number of assignments that were converted for each class. Since the required courses are taught by more than one faculty and assignments vary by instructor, this required conversion for every instructor who planned to teach the required course over the following two years as computational exercises vary by faculty.

Required Course Topic	Number of faculty teaching course	Number of students hired	Total Number of hours worked over the summer	Number of Exercises converted
Physiology	2	1	62	4 homework assignments
Neural Engineering	1	2	150	8 lab assignments
Biomechanics	1	3	31.5	3 labs including data files, 1 homework assignment
Imaging	2	2	31.5	5 homework assignments, 5 simulations
Signals and Systems	2	2	148	16 simulations

In addition to the students, the python consultant was available to meet as needed with the students and had regular check ins with the students to oversee the conversion process. Students met with faculty to demo the work they put together.

As detailed in table 1, students were responsible for converting every aspect of the course materials from MATLAB to Python that involved different exercises depending on the class. Conversions included laboratory exercises where data had to be imported and therefore stored in a Python compatible format, to homework assignments, to faculty simulations that are performed in front of students, often with students providing inputs or various parameters. As part of the conversion efforts, students created the document, homework, or exercise that would be circulated to students as well as the key for faculty and TAs to use throughout the semester. The effort needed to convert classes was variable and depended mostly on the number and types of exercises that needed to be converted as well as the student skill level with Python. Faculty involvement largely depended on the individual faculty member teaching the course. Some had weekly meetings with the students and frequent discussions, and other faculty had their TAs for the fall and spring interact with the hired students instead.

The students who participated in the conversion effort felt proud of their efforts. They bemoaned the low hourly wage they received for their skilled labor, but also appreciated being part of what they perceived as a change of "outdated" computational exercises. Students who participated in this effort used this opportunity to leapfrog to others such as TAing and getting more involved in teaching and pedagogy. One student indicated that "I think it was a unique opportunity to take a class and then immediately be able to fix the things that I didn't like about it."

Discussion:

The involvement of students in the conversion process played a crucial role in ensuring the effectiveness and relevance of the converted exercises. Crucial to the success was selecting students who successfully took the courses they were involved in converting and were recommended by the instructor of record. In this way students provided valuable insights beyond mere technical execution as by participating they improved course materials, making labs and exercises more coherent and easier to follow. By soliciting feedback from these students, we gained valuable perspectives on the usability and efficacy of the converted assignments, allowing for iterative improvements beyond basic conversion tasks. This approach not only enhanced the quality of the converted exercises but also fostered a sense of ownership and investment among the student community and they were proud to be part of the effort. In addition, the conversion effort was valuable to the students in that they gained real debugging and python coding skills. The amount of time needed to convert exercises depended on difficulty as well as student skill implementing Python. As a recommendation, it would be helpful to have students apply and demonstrate Python proficiency before being engaged in the conversion efforts as some students performed the conversion more quickly and with less overall effort than others as can be seen from the table indicating hours worked. In addition, the students typically performed the conversion in isolation and another recommendation would be for the students to collaborate on the conversion more as that too could reduce the overall effort and improve the output result.

Some limited number of the exercises that needed to be converted used a graphical user interface (GUI). The students taking these core BME classes do not do any GUI development themselves, but need the ability to run this code with a GUI in Python. The conversion of these GUI-based exercises posed a significant learning curve for both students and faculty involved in the conversion process. Given the limited experience with GUI development in both MATLAB and Python, substantial time and effort were dedicated to acquiring the necessary skills. Luckily, one member of the conversion team had significant experience in Python GUI development and could guide others and provide advice. Identifying such a person for any future conversion teams is suggested. Overall, the learning experience inherent in tackling GUI development contributed to student growth and skill diversification. Additionally, resources provided for Python GUI creation facilitated the adaptation of MATLAB-based projects to Python, albeit with some adjustments and compromises.

Standardization emerged as a key consideration in the conversion process, particularly regarding the format and presentation of converted code. Discussions centered around the use of Jupyter Notebooks for their interactive capabilities and the importance of maintaining consistency across biomedical engineering classes. While decisions regarding plotting libraries, such as matplotlib or Seaborn, were deliberated, additional standardization aspects, such as data handling and function usage, may warrant further consideration to ensure seamless integration of converted exercises into the curriculum.

Assistance from the Python consultant primarily focused on troubleshooting and providing guidance on technical discrepancies between MATLAB and Python functions. Notably, differences in data handling, such as those observed in sound file importation, highlighted the importance of thorough examination and validation of converted code. Faculty intervention was instrumental in identifying and resolving such discrepancies, ensuring the fidelity and functionality of the converted exercises. Furthermore, faculty support extended to reviewing Jupyter Notebooks for copyediting and formatting, enhancing the clarity and coherence of instructional materials.

The biggest hurdle to overcome with this transition was overcoming faculty resistance. Faculty are well versed in MATLAB, for some it is their only known language, and are resistance to invest the time and energy to improve their skills in Python. While many will say that computational skills are foundational and agnostic to languages, differences in notations, function calls, and ways of storing data can be challenging to overcome without significant effort. One faculty member indicated the conversion effort was neither an insurmountable challenge nor a trivial one.

Student reception to the changes has been largely positive, and assessment is planned over the coming years as the first cohort of students to experience the full conversion effort will graduate in 2026. Since the new cohort of students took their primary introductory course in Python, the incoming class of students see the Python embedded in the curriculum as a continuation of their learning and have not noticed a significant change. Faculty feedback so far has been positive, indicating there have been few problems with the adoption and the transition has gone smoothly. Students involved in the conversion effort over the summer have stayed in contact with the faculty teaching the courses and indicated their assignments have been utilized successfully. Some faculty indicated that some of the exercises are slow and some of the features in Python are not to their liking. That is to be expected with any change as faculty have to learn new ways to implement their exercises and gain proficiency in Python themselves. One surprising benefit of making the transition is that for faculty unfamiliar with Python, it has brought them closer to their student's perspective as both students and faculty alike have had to embark on learning a new tool.

In summary, the collaborative effort to convert MATLAB-based exercises into Python within the biomedical engineering curriculum benefited greatly from student involvement, faculty support, and a standardized and methodical approach of utilizing Jupyter Notebooks. By leveraging student feedback, addressing technical challenges, and ensuring consistency in presentation, the conversion process yielded a more cohesive and accessible learning experience. Moving forward, continued collaboration, refinement, and assessment will be essential to further enhance the effectiveness and sustainability of Python integration in biomedical engineering education.

As the conversions have taken place and many of the classes affected have been taught once already, faculty feedback from affected conversion efforts has been strongly positive. Some instructors notice no difference and say students and their supportive TAs are ready, willing, and able to utilize the provided Python notebooks to perform class exercises. Some faculty indicate some students are reluctant to use Python when they are more comfortable with MATLAB. As the university taught computational course has recently converted to Python, it is expected that rising upperclassmen will be more familiar with Python and that will be a short-lived problem.

Conclusion:

This paper details a successful process for converting biomedical engineering curriculum from MATLAB to Python that could be employed at other institutions by leveraging student resources on campus. Python has grown in popularity and is the industry's choice language. It is to our students benefit to develop their Python programming skills within technical course content and have biomedically relevant programming exercises to bring home important concepts and help students develop their programming skills, which will only continue to grow in importance. Getting faculty buy in can be challenging when the cost of faculty time to make the conversion is so high. This paper outlines a process for making the conversion as easy for faculty as possible. We began by selecting a test case course, Signals and Systems, for a comprehensive conversion effort and a test of the methods described in this paper. Through discussions with faculty, we opted for Jupyter Notebooks as a uniform platform for Python integration. Following a trial run with a smaller cohort, we expanded our efforts by hiring undergraduate students to assist with the conversion process for downstream courses. This model proved effective in reducing faculty workload and ensuring a smoother transition to Python with minimal disruption to students. The hiring of a Python consultant and a team of students over the summer facilitated the systematic conversion of MATLAB exercises to Python across seven courses, spanning over 14 faculty, ensuring consistency and alignment with program educational objectives. Overall, our methodical approach, coupled with interdisciplinary collaboration, has enabled us to successfully bridge the gap between traditional classroom practices and contemporary industry demands in biomedical engineering education.

Works Cited

[1] G. C. Fleming, "What engineering employers want: An analysis of technical and professional skills in engineering job advertisements," *Journal of Engineering Education,* 2024.

[2] TIOBE, "The TIOBE index," 2024. [Online]. Available: https://www.tiobe.com/tiobe-index/.

[3] A. Gujar, "C vs Python: A Cursory Look with Industry Opinion," *Internationl Journal for Research in Applied Science & Engineering Technology,* vol. 11, no. 11, 2023.

[4] S. Cass, "The 2018 top programming languages.," *IEEE Spectrum,* 2018.

[5] D. Ramel, "Popularity Index: Python is 2018 "Language of the Year"," [Online]. Available: https://adtmag.com/articles/2019/01/08/ti obe-jan-2019.aspx..

[6] G.-J. J. Samuel-Felipe Baltanas, "Jupyter Notebooks in Undergraduate Mobile Robotics Courses: Educational Tool and Case Study," *Applied Sciences,* pp. Vol 11, Iss. 3, 2021.

[7] C. Tang, "Computer-aided Linear Algebra Course on Jupyter-Python Notebook for Engineering Undergraduates," *Journal of Physics: Conference Series,* 2021.

Appendix:

BME 303: Modern Diagnostic Imaging Systems
Spring 2022

Homework Assignment #3 Part B, 100 points
Assigned Friday, February 18th 2022
Part 2 Due: by 12:00am on Gradescope on Friday, March 4th, 2022

1. Let's have some fun in Matlab!

   (a) Build a circle in MATLAB with a diameter of 18cm (like you did in the last problem of Homework 3A) and a μ=0.3 cm^{-1} using whatever method you like. For consistency sake make your total image 20cm wide with a step size of .01, with an 18cm diameter circle in the middle. Using the 'sum' command in MATLAB, create 1 projection of your simulated circle. Please note the 'sum' command just adds up all rows or columns so it replaces the integral with a summation instead.

   (b) In MATLAB, using the analytical solution you found in HW 3 Part A, 5b), explicitly determine one projection of the circle using the same angle you chose previously for the same l you used in your simulation (20 cm wide with a stepsize of .01cm). Plot your analytical solution from 5b) on top of your simulated solution from above. Are the answers the same? Why or why not? What would you need to do to make them the same? Since you know they *should* be the same make any adjustments necessary to your simulated circle projection to make it match the analytical solution perfectly. Please don't 'hard code' your solution - i.e. if you were to change paramters when you went to replot the analytical solution vs the simulated solution would they still match?

   (c) If you were to change your object from 20 cm wide with step size of .01 to 40 cm wide with a step size of .001, how would you fix your simulated solution?

(d) Display the sinogram for a circle, $g(l, \theta)$ for 12 angles. Please make this sinogram yourself (i.e. no using radon/iradon - I'll let you use those in a bit) You should be able to create this sinogram using matrix math. Create a 3x1 subplot with the original circle on the first subplot and the plot of the analytical solution on top of the simulated solution(with appropriate legend) in the second subplot and the sinogram for 10 angles of the circle in the third. Attach the figure here.

(e) How many projections do you need to use to make sure your dataset is Nyquist sampled if you wanted to reconstruct a 128x128 image?

(f) How many projections would you need to make sure your dataset is Nyquist sampled if you wanted to reconstruct a 512x512 image?

(g) Explain why the number of projections is different from the two examples you just calculated.

2. Simple Backprojection (Yes, you can do it!)

   (a) Using Matlab, create a 128x128 pixel image consisting of an 20x20 pixel white square centered on a black field. This represents your object. Display this image in grayscale.

   (b) Consider that you have an array of 128 detectors. One exposure of x-rays from the source strikes a row of 128 detectors, that are each acquiring a line integral for your image. Using the imrotate function, with 'nearest' interpolation, calculate the 128 line integrals (using the 'sum' command like you did before earlier in this homework, fixed like you fixed it previously) for your phantom and acquire 10 projections of your phantom using 8 equal increments between 0 and 180 degrees (but don't include 180 that would be double counting the same projection). Note that imrotate doesn't return an image of equal size as its input, so you must crop the edges of each rotated image so that you are only including the central 128x128 portion of the rotated image. The size and ceil functions will be useful here in doing this. After cropping, use the sum function to obtain a vector of 128 line integrals, repeat for each value of θ. Create a 1x2 subplot and display the original object in the first plot and all 10 projections in a sinogram for the second. Upload this to Sakai when asked.

   (c) Now reconstruct your image from the sinogram. 'Smear' each vector of projections over 128 rows (each row should be identical using matrix math), and populate a 3D array (128x128x10) with your 'smeared' vectors. Then use imrotate and cropping again rotate in the opposite direction to re-orient each smeared vector with the original coordinate system, and populate a new 3D array. (Hint: Use a 'for' loop to step through your # of angles, and matrix algebra to smear each projection) Finally, use sum(:,:, 3) to add all the projections together into a 128 x 128 matrix that represents the backprojection result. Normalize your result, and display it using the mesh and colormap gray commands. Upload this figure to Sakai where asked. Can you see your phantom in the backprojection result?

3. Download Corona.mat from the Sakai site, a 650x650 matrix and complete the following.

(a) Use MATLAB's 'radon' function to calculate the sinogram for the image provided in the .mat file assuming you take 179 projections from 0 to 179 degrees. Note the radon transform uses degrees, not radians (ironically). Plot the sinogram of your image. What does the x and y axis of the sinogram represent?

(b) Using the iradon function (which uses filtered backprojection to calculate the inverse Radon transform), reconstruct an image from the sinogram. Use 'nearest' for interpolation and use 'Hamming' for the type of filter.

(c) Repeat using 'Ram-Lak' and 'Cosine' filters and view all 4 images (Original, Hamming, Ram-Lak, and Cosine) on one figure using subplot. Display and label all images. Display the image here.

(d) Extract a line profile through the original and the 3 reconstructed images at the exact same position in the image and, using 'hold on', plot all line profiles on top of one another. Use 'legend' to specify which line profiles belong to which filter (and MATLAB has plot features such as '*' and '-' that allow lines to be plotted differently even if not printed in color). Describe the effect of the different filters on the reconstructed images. Which filter did the best job of reconstructing your image and why do you feel that way?

(e) Recreate the sinogram at 10 degree increments (instead of 1 degree) and take the inverse Radon transform using the filter of your choice. Display your image below.

(f) How does reducing the number of projections affect image quality? What is the trade-off associated with choosing a small number of projections versus a large number of projections when doing a CT scan?

(g) Suppose that the number of detectors (i.e. along l axis) is halved. Make a new sinogram by averaging the signal coming from every pair of adjacent detectors in the original sinogram from Problem 6. Reconstruct the image using this new sinogram, 'nearest' interpolation, and a Hamming filter. Display the reconstructed image. Comment on the difference between this reconstructed image verses the reconstructed image in (b). What is this equivalent to in the real world?

(h) Display the image below.

(i) Simulate what might happen in your reconstructed image if one detector's scintillation crystal fell off and reports no signal for all theta. Display your reconstructed image with one detector crystal missing and upload it below.

(j) How might this be different for different generations of CT scanners?

(k) In class, we've talked about how the 0^o and 180^o radon transforms will have the same data. Is this always true? What factors might make the attenutation more or less depending on the way it approaches the detector?

4. (8 points) Extra Credit! Let's try a new problem if you are so inclined, combining Poission Distributions from Xray and CT to see how it affects image quality in CT.

   (a) As mentioned in class, CT data takes Beer's Law and readjusts it, such that the radon transform you created from the previous Corona problem was actually the intregral of $\mu(x)$ dx. We ended up with the following equation: $ln(\frac{N_0}{N}) = \mu dx$. The Poisson affects the N and N_0.Create a 923x180(the size I found the radon transform produces for an object that is 650x650 like our Corona) array of poisson distributed numbers with a mean of 100 that will serve as N_0 and another 923x180 array of poisson distributed numbers with a mean of 100 that will serve as N for each projection. Take the second distribution (your N) and multiply by $e^{-\mu dx}$ and round the result to the N that makes it to the detector. Be sure to round() the result as you cannot have 1/2 an Xray making it to the detector. For this part I got some Infinite results that were due to the fact that at certain portions of my object no Xrays made it through the material. To fix this I divided my Radon transform result by 100. Then take $ln(\frac{N_0}{N})$ for each projection and iradon the result. Was a mean of 100 enough? If not increase the fluence until you are happy with the result. Display your filtered back-projection result below. Display the result for at least 3 different fluences using subplot to demonstrate what happens to your CT as your fluence increases.

(b) How did the fact that Xrays obey a Poisson distribution affect the result? Were there any new artifacts you saw?

BME 303L: Modern Diagnostic Imaging Systems

Laboratory 3 Part B

Spring 2023

Due: Friday March 10th at 8:00pm

Problem 1:

Let's have some fun in Python!

(a) Build a circle with a diameter of 20cm (like you did in problem 7b of Homework 3A) and a μ=0.6 $cm^{−1}$ using whatever method you like. For consistency sake make your total image 22cm wide with a step size of .01, with an 20cm diameter circle in the middle. Using the ′sum′ command, create 1 projection of your simulated circle. Please note the ′sum′ command just adds up all rows or columns so it replaces the integral with a summation instead.

In [ ]:
```
import numpy as np
import matplotlib.pyplot as plt
import math
```

In [ ]:
```
#Write equation for a circle here, once equation for a circle is created
#Take the integral of the circle over one dimension (x or y will work) using th
```

(b) Using the analytical solution you found in HW 3 Part A, 7b), explicitly determine one projection of the circle using the same angle you chose previously for the same $l$ (distance from origin) you used in your simulation (22 cm wide with a stepsize of .01cm). Plot your analytical solution from 7b) on top of your simulated solution from above. Are the answers the same? Why or why not? What would you need to do to make them the same? Since you know they *should* be the same make any adjustments necessary to your simulated circle projection to make it match the analytical solution perfectly. Please don′t ′hard code′ your solution - i.e. if you were to change paramters when you went to replot the analytical solution vs the simulated solution would they still match? Your goal is for your Python simulation to create $\int \mu(x)dx$, as we did in class together.

In [1]:
```
# d =
# mu =
# imsize =
# dx =
#plt.imshow()
```

In [2]:
```
## Plotting Simulated and Analytical Solutions

#proj_simulated =



#proj_analytical =




#fig, ax = plt.subplots()
#ax.plot(proj_simulated)
#ax.plot(proj_analytical, linestyle = "dashed")
```

(c) If you were to change your object from 22 cm wide with step size of .01 to 44 cm wide with a step size of .001, how would you fix your simulated solution?

Write your response here

(d) Display the sinogram for a circle, $g(l, θ)$ for 12 angles. Please make this sinogram yourself using matrix math.

In [3]:
```
## EXAMPLE CODE:
import numpy.matlib

#sinogram =
#theta =

#fig, ax = plt.subplots()
#plt.imshow()
```

(e) How many projections do you need to use to make sure your dataset is Nyquist sampled if you wanted to reconstruct a 128x128 image?

Write your response here

(f) How many projections would you need to make sure your dataset is Nyquist sampled if you wanted to reconstruct a 512x512 image?

Write your response here

(g) Explain why the number of projections is different from the two examples you just calculated.

Write your response here

Problem 2

Simple Backprojection - yes you can do it!

(a) Create a 128x128 pixel image consisting of an 20x20 pixel white square centered on a black field. This represents your object. Display this image in grayscale.

In [4]:
```
#square

#fig, ax = plt.subplots()
#ax.imshow()
```

(b) Consider that you have an array of 128 detectors. One exposure of x-rays from the source strikes a row of 128 detectors, that are each acquiring a line integral for your image. Using the rotate function in skimage, calculate the 128 line integrals (using the 'sum' command) for your phantom and acquire 10 projections of your phantom using 8 equal increments between 0 and 180 degrees (but don't include 180 that would be double counting the same projection). Display all 10 projections in a sinogram for the second.

In [5]:
```
## EXAMPLE CODE:
import skimage
from skimage.transform import rotate


#sq_sinogram =


#for n in range():
  #new_pic =
  #sums =


#fig, ax = plt.subplots()
#ax.imshow()
```

(c) Now reconstruct your image from the sinogram. 'Smear' each vector of projections over 128 rows (each row should be identical using matrix math), and populate a 3D array (128x128x10) with your 'smeared' vectors. Then use the rotate function again, but this time rotate in the opposite direction to re-orient each smeared vector with the original coordinate system, and populate a new 3D array. (Hint: Use a 'for' loop to step through your # of angles, and matrix algebra to smear each projection) Finally, use `sum(axis=2)` to add all the projections together into a 128 x 128 matrix that represents the backprojection result. Normalize your result, and display it as a 3D projection. Upload this figure to Sakai where asked. Can you see your phantom in the backprojection result?

In [6]:
```
#backproj =
#stack_count =


#for n in range():


#final_sum =
#fig = plt.figure()
```

Problem 3

Download Corona.mat from the Sakai site, a 650x650 matrix and complete the following:

(a) Use skimage's 'radon' function to calculate the sinogram for the image provided in the .mat file assuming you take 179 projections from 0 to 179 degrees. Note the radon transform uses degrees, not radians (ironically). Plot the sinogram of your image. What does the x and y axis of the sinogram represent?

Write your response here

In [7]:
```
# EXAMPLE CODE
# For file processing (IO = input/output):
import scipy.io
# For signal processing:
from scipy import signal
# For radon transform:
from skimage.transform import radon

# Importing corona image data:
#data = scipy.io.loadmat('Corona.mat')


#fig = plt.figure()

##Plot Corona
#ax1 = plt.subplot(121)


##Plot Randon
#ax2 = plt.subplot(122)
#theta =
#cor_sinogram =


#fig.tight_layout()
#plt.show()
```

(b) Using the iradon function (which uses filtered backprojection to calculate the inverse Radon transform), reconstruct an image from the sinogram. Use 'Hamming' for the type of filter.

In [ ]:
```
#Write code here
```

(c) Repeat using 'Cosine' filter and view all 3 images (Original, Hamming, and Cosine) on one figure using subplot. Display and label all images. Display the image here.

In [8]:
```
#reconstruction_ham =

#reconstruction_cos =

#fig = plt.figure()
```

(d) Extract a line profile through the original and the 3 reconstructed images at the exact same position in the image and plot all line profiles on top of one another. Use 'legend' to specify which line profiles belong to which filter. Describe the effect of the different filters on the reconstructed images. Which filter did the best job of reconstructing your image and why do you feel that way?

Write your response here

In [9]:
```
#LineProfile_Original =
#LineProfile_Hamming =
#LineProfile_Cosine =

#fig, ax = plt.subplots()
```

(e) Recreate the sinogram at 10 degree increments (instead of 1 degree) and take the inverse Radon transform using the filter of your choice. Display your image below.

In [10]:
```
#fig = plt.figure()

#ax1 = plt.subplot(121)
#theta =
#cor_sinogram_2 =
#dx, dy =

#ax1.imshow()

#ax2 = plt.subplot(122)
#reconstruction_cos_2 =

#ax2.imshow()

#plt.show()
```

(f) How does reducing the number of projections affect image quality? What is the trade-off associated with choosing a small number of projections versus a large number of projections when doing a CT scan?

Write your response here

(g) Suppose that the number of detectors (i.e. along l axis) is halved. Make a new sinogram by averaging the signal coming from every pair of adjacent detectors in the original sinogram from Problem 6. Reconstruct the image using this new sinogram, 'nearest' interpolation, and a Hamming filter. Display the reconstructed image. Comment on the difference between this reconstructed image verses the reconstructed image in (b). What is this equivalent to in the real world?

In [ ]:
```
#Write code here to simulate two detectors being averaged into 1
```

Write your response here

(h) Display the image below.

In [ ]:
```

```

(i) Simulate what might happen in your reconstructed image if one detector's scintillation crystal fell off and reports no signal for all theta. Display your reconstructed image with one detector crystal missing and upload it below.

In [11]:
```
#damaged_sinogram =

#theta =
#reconstruction_damaged =

#fig = plt.figure()

#ax2.imshow()


#plt.show()
```

(j) How might this be different for different generations of CT scanners?

Write your response here

(k) In class, we've talked about how the 0o and 180o radon transforms will have the same data. Is this always true? What factors might make the attenutation more or less depending on the way it approaches the detector?

Write your response here

Problem 4

1. Import an image of an interesting object (black/dark background and signal intensity at center) and find the radon transform of it and display it from 0^o to either 180^o or 360^o degrees, depending on what you prefer. You are welcome to find something off the internet, take a picture of your phantom you made on a dark background, something you have in your room, whatever you like. The only rule is it can only have significant signal intensity in the middle of the image (like a CT scanner where the patient is placed in the middle of the scanner). We'll display our favorites to class as extra credit!

In [1]:
```
#Import Image
#Take Radon Transform as you did before
#Display it for all
```

1. Using whatever method you like, perform the filtered backprojection and display both the sinogram and the reconstructed image in a 1x2 subplot below.

In [ ]:
```
#Your code here
```