2016

MATH 433: Nonlinear Optimization—A Peer Review of Teaching Project Benchmark Portfolio

Adam Larios
University of Nebraska-Lincoln, alarios@unl.edu

Course Portfolio for Math 433 - Nonlinear Optimization
Introducing Programming in a Senior-Level Mathematics Course

Adam Larios

May 31, 2016

Course: Math 433 – Nonlinear Optimization
Spring 2016, University of Nebraska–Lincoln

Abstract

My intention in this portfolio is to highlight various approaches to teaching higher-level mathematics with a programming component that I tried in a course on nonlinear optimization. There is a particular focus on students with little or no previous programming experience. Several case studies are done using "pre- and post-course" surveys which examined items such as student confidence in programming, and particular programming skills. Sample exams and sample homeworks are presented and discussed. Also presented are several materials I designed to lead students into programming and shed light on certain problems. These materials assume some basic mathematical reasoning and knowledge, and therefore may be best suited to students in STEM fields. Moreover, learning outcomes are described and assessed with quantitative data. A coding contest for motivating students is also discussed.

Keywords: Undergraduate teaching, teaching with technology, Matlab in the classroom, mathematics programming, nonlinear optimization.

Contents

1 Introduction and Objectives of Course Portfolio **3**

2 Summary and Overall Assessment of Peer Review Process **4**

3 Benchmark Memo 1: Reflections on the Course Syllabus **6**
3.1 Description of the Course . . . 6
3.1.1 Purpose of the course . . . 6
3.1.2 Students . . . 6
3.1.3 Curricula . . . 6
3.1.4 The Course and the Broader Curriculum . . . 6
3.2 Goals and Objectives of the Course . . . 7
3.2.1 Knowledge . . . 7
3.2.2 Abilities . . . 7
3.2.3 Understanding . . . 7
3.2.4 Retention . . . 7
3.2.5 Perspectives and Attitudes . . . 7
3.2.6 The Field, and The Larger Society . . . 8
3.2.7 Justification for Goals . . . 8

3.3 Goals Reflected in Daily Course Structure . . . 8
3.4 Goals of the Portfolio . . . 8
3.4.1 Specific Goals of the Portfolio . . . 9

4 Benchmark Memo 2: Description of Course Activities 10
4.1 Teaching Methods and Classroom Time . . . 10
4.1.1 How Are The Methods Used Over The Course of The Semester? . . . 10
4.1.2 How Do The Methods Facilitate Course Goals? . . . 10
4.2 Course Activities Outside of Class . . . 10
4.2.1 Reasoning Behind The Structure . . . 10
4.2.2 Learning Goals of Each Activity . . . 10
4.2.3 Assessing Student Performance . . . 11
4.3 Course Materials . . . 11
4.3.1 Rational for the Use of the Course Materials . . . 11
4.3.2 How Should Students Use The Materials? . . . 11
4.4 Rational for Choice of Teaching Methods . . . 12
4.4.1 Influences of Wider Curriculum . . . 12
4.4.2 Impact of Materials on Student Learning . . . 12
4.5 Course Choices and the Broader Curriculum . . . 12
4.5.1 Building Upon Material From Previous Courses . . . 12
4.5.2 Preparing Students for Broader University and Department curriculum . . . 12
4.5.3 Effect On Students' Future Courses And Career . . . 12

5 Benchmark Memo 3: Document and Analyze Student Learning 13
5.1 The Nature of Student Understanding (in each focused activity analyzed) . . . 13
5.1.1 Evidence of Students Meeting Learning Goals . . . 13
5.1.2 Differences in Student Understanding Shown In Work Samples . . . 13
5.1.3 Analysis of Student Understanding . . . 13
5.1.4 What Students Are Actually Learning: Outcomes and Misunderstandings . . . 13
5.2 Distribution of Student Performance . . . 13
5.3 Student Performance And The Broader Curriculum (For Overall Course) . . . 13
5.3.1 Did Distribution of Student Work Meet Expectations For Intellectual Goals? . . . 13
5.3.2 Preparation For Other Courses: Evidence Via Documented Student Performance . . . 14
5.3.3 What Does Student Work Say About Their Prior Preparation? . . . 14
5.3.4 Possible Changes, Redesigns, And Future Plans . . . 14

6 Syllabus 15

7 Coding Contest 21
7.1 Description of the contest . . . 21
7.2 Results of the speed trial . . . 22
7.3 Results of the readability trial . . . 24

8 Comparative Case Studies 27
8.1 Pre-Course Survey . . . 28
8.2 Post-Course Survey . . . 30
8.3 Results and Analysis . . . 32
8.3.1 Measures of Confidence and Broader Scope . . . 32
8.3.2 Measures of Basic Programming Knowledge . . . 34

9 Introduction to Matlab Worksheet **35**
9.1 Matlab As A Calculator . . . 35
9.1.1 Comments . . . 35
9.2 Variables . . . 35
9.3 Matrices and Vectors . . . 36
9.3.1 Matrix and Vector Operations . . . 36
9.3.2 Building Matrices and Vectors . . . 37
9.3.3 Accessing Matrices and Vectors . . . 37
9.4 Getting Help . . . 38
9.5 Plotting . . . 38
9.5.1 Multiple plots . . . 39
9.5.2 Labeling and Legends . . . 39
9.5.3 Subplots . . . 39
9.6 Saving work in m-files . . . 40
9.7 Loops . . . 40
9.7.1 Nested loops . . . 40
9.7.2 Initialization . . . 41
9.7.3 Tracking loop iterations . . . 41
9.8 First basic program: Fibonacci Numbers . . . 42
9.9 Conditionals (if/then statements) . . . 43
9.9.1 Breaks and While Loops . . . 44
9.10 Functions . . . 44
9.10.1 Programming the Factorial Function . . . 44
9.10.2 Two-variable functions: Programming the choose function . . . 45
9.10.3 Multiple Outputs . . . 46
9.10.4 Symbolic functions . . . 46
9.11 Miscellaneous Tips . . . 46

10 Exam Samples **48**
10.1 Exams . . . 48
10.2 Sample student exams . . . 56
10.2.1 Exam 1 . . . 56
10.2.2 Exam 2 . . . 68

11 Homework Samples **78**
11.1 Sample student homeworks . . . 78
11.2 Homework Solutions . . . 86
11.2.1 Homework 0 . . . 86
11.2.2 Homework 1 . . . 88
11.2.3 Homework 2 . . . 91
11.2.4 Homework 3 . . . 95
11.2.5 Homework 4 . . . 98
11.2.6 Homework 5 . . . 104
11.2.7 Homework 6 . . . 108
11.2.8 Homework 7 . . . 112

1 Introduction and Objectives of Course Portfolio

This portfolio is aimed at serving several purposes.

The first purpose is to to examine how my course goals connect to my course activities, looking for possible improvements. In particular, understanding certain methods used in teaching programming in a higher-level

mathematics class, and to try to discover ways to improve these methods. This is addressed in the memos, in the comparative case studies, and also in the coding contest.

The second purpose is to document the major resources and materials I have created for the course, and observe how they impact the learning goals of the course. This is also to hopefully serve as a resource for teaching the course in the future, and hence, many of the course materials are included, as are samples of student work with comparative comments. In particular, a worksheet for introducing students to Matlab that I created is included. This worksheet received very positive comments from students. There is also a coding contest which the students found very fun, and I believe was a great success in terms of student learning. It gave students a fun and challenging way to write the best code they could, and gave them an opportunity to critically examine each other's codes. The details of how this contest was run (including some steps to make it more efficient and anonymous) are recorded in detail, hopefully to aide future instructors who want to try this or a similar technique.

The third purpose is to make a first step towards a larger-scale study of teaching mathematics with programming goals in mind. For me personally, this portfolio will serve as a preliminary step towards a larger-scale, more involved study about teaching methods for mathematics courses with a programming component.

The goals associated with these purposes are met below, via several means. A major component is the data analysis in before-and-after tests of student confidence in and understanding of programming. These data show clear improvements in student confidence and knowledge. Moreover, by examining homework and exam samples, we can see specific areas that students struggle with, which may allow one to plan for future courses by making adjustments to addresses these areas.

2 Summary and Overall Assessment of Peer Review Process

As part of the review process, I implemented the following changes to the course:

1. I re-thought the course, and ended up cutting out some material that did not seem to address the key learning goals. I replaced it with an increase in time for in-class programming activities. I also designed before-and-after tests in hopes of measuring student progress. I felt that these changes helped to stream-line the course, and make my teaching more effective, in addition to giving me more information about my teaching.

2. I examined the before-and-after tests. The "before" portion showed me that more students than I was expecting had a fair amount of programming experience, and that almost all students had some programming experience. However, few of the students had experience in scientific programming, and it was focused mostly on things like web-based programming. This allowed me to spend less time focused on the syntax in programming, and gave me more time to focus on the understanding and implementation of algorithms. Without the peer review process, I wouldn't have had this insight.

3. After serious consideration, I decided to stick with lecture-based learning for the majority of the course, but made it a point to have the lecture include many more opportunities for question and answer, and I also implemented the think-pair-share method (discussed below), which was successful in terms of both student learning and attention.

4. I determined the value of linear algebra as a prerequisite for the course. It is clear that those with a weaker background in linear algebra had a more difficult time with the course, and therefore linear algebra is clearly a very necessary prerequisite.

5. I was open with the students about the methods I chose for the course, and why I chose them. IN particular, I started writing the key points we were going to cover each day on the board at the start of class. This lead to a huge improvement in student evaluations for me. I used to get many comments along the lines of "the lectures seem disconnected", etc. These comments are essentially gone now, and it appears that the student have a clear picture of what we are doing and why we are doing it. Of

course, they might not agree with everything, but knowing that there is a plan seems to have made a huge improvement in the students ability to follow the lectures. This improvement is a direct result of the peer review program, as I started implementing it immediately after it was mentioned in one of the first PRTP sessions.

3 Benchmark Memo 1: Reflections on the Course Syllabus

3.1 Description of the Course

UNL Course Bulletin Description
Mathematical theory of unconstrained and constrained optimization for nonlinear multivariate functions, particularly iterative methods, such as quasi-Newton methods, least squares optimization, and convex programming. Computer implementation of these methods.

Advertisement blurb to students:

This course studies nonlinear optimization, both the mathematical theory and its effective application to solve practical problems. We will begin by reviewing the optimization methods of calculus, and then consider numerical iterative methods, including Newton's method and least squares. We then proceed to the major topic of the semester: Convex optimization. This topic is highly important in economics and finance, where the functions are often not differentiable but are convex. Fortunately, many Calculus-based methods can be extended to this setting. A major highlight here are the Karush-Kuhn-Tucker conditions for optimality of a convex program.

Many of the techniques in the course are not just of interest in mathematics, but also in science and industry. We will also do a fair amount of programming (in a language of your choice, but Matlab or Python are natural choices). No prior knowledge of programming is assumed, so this may be a good place to learn programming, or broaden your programming experience.

3.1.1 Purpose of the course

This course serves three major purposes. First: get students to be able to recognize optimization problems in real-world situations to the extent that they can write down a relevant mathematical model for an applied optimization problem. Second: to teach the students a selection of mathematical techniques to solve the problems inherent in these models. Third: to teach students the basics of computer programming to the extent that they can solve large-scale optimization problems computationally.

3.1.2 Students

The students will be mostly junior and senior level mathematics and engineering majors, with possibly some computer science majors. They will have taken a course in linear algebra, which will give them the necessary mathematical background, as well as a "proofs-based" course, which will give them the intellectual framework to handle the abstract concepts in the course. Engineering students will likely know at least some applications of this material, which adds an intellectual richness to the course that should not be overlooked.

3.1.3 Curricula

This course is an elective course, and is not required for any major or minor. It can be used to satisfy part of the the 400-level course requirement for mathematics and physics majors.

The course can also be taken at the 800-level as Math 833. For this, an extra project or assignment should be designed.

3.1.4 The Course and the Broader Curriculum

The course is not a prerequisite course for any other course. Nevertheless, it can be a course which informs future business, economic, and scientific questions, and may be desirable to industry or academic programs.

It would be interesting to see if this course could be worked into a course which satisfies a programming requirement of certain majors.

3.2 Goals and Objectives of the Course

3.2.1 Knowledge

Students should come away from the course knowing how to solve optimization problems with classical and modern mathematical techniques. They should also know how to write basic programs in MATLAB.

Concretely, some of the ideas they should be familiar with include:

- The simplex method (i.e., convex optimization)
- The Duality Principle in convex optimization problems
- The conjugate gradient method
- Quasi-Newton methods
- Least squares optimization
- The KKT conditions and the associated methods for optimizing nonlinear problems

3.2.2 Abilities

Students should be able to think of a real-world problem and write down a mathematical model for it. They should also be able to take a mathematical optimization problem and write a MATLAB program to solve it using the algorithms we learn in class.

3.2.3 Understanding

Student should understand how these algorithms work well enough to be able to program them explicitly. They should also understand when it is appropriate to choose one method over another, and what the rade-offs are of various methods (e.g., speed vs. accuracy, robustness vs. efficiency).

They should also understand how to quantify real-world variables and implement them into a sound mathematical model (this includes understanding how to check if a model is sound/feasible).

3.2.4 Retention

It is unlikely that a student will use *all* of the techniques discussed in the course in future applications. However, predicting which particular technique will be of use is likely impossible. Therefore, one goal is to build up a "test-bank" of technique and methods that the students can draw upon for when they meet future challenging problem in science, business, industry, etc. Sometimes just knowing a certain technique exists is a major advantage in dealing with a difficult problem.

More important than particular techniques is the ability to frame real-world problems in terms of mathematical optimization problems. This is a major skill we want to develop and hone in the students.

Computer programming is another point upon which we want to build some retention. Being able to wield the basics of programming is a skill that can be used in a virtually unlimited number of fields; it is akin to being able to read and write. Familiarity with these basics, as well a demystification of the world of computer programming, are high priority for passing on to the students for long-term retention.

3.2.5 Perspectives and Attitudes

It is common for many students to have a negative attitude toward computer programming (they often view it as boring, or as tedious number-crunching). However, after having experiences which demonstrate the power and simplicity of programming, many students become drawn to it, and often begin to explore programming on their own.

In terms of attitudes toward mathematics, since these are junior and senior-level students in STEM fields, we likely do not have to do much to improve attitudes towards mathematics. However, it would be wise to focus on developing a wider-world perspective about the applications of mathematics.

3.2.6 The Field, and The Larger Society

As the subject of this course is relatively new, there is little history to be taught, although the fact that this is very modern mathematics will be emphasized. It would be interesting to discuss open mathematical problems in the subject.

A major opportunity exists in the course to tie the subject into other fields. We will focus on showing many, many examples of how this subject has influenced the development of technology, business, and society. We will also discuss areas where application of the subject could have a strong impact.

3.2.7 Justification for Goals

We want the student to come away from the course with an ability to make in impact in the lives of others, for example though applying the ideas of optimization to improve technology and design in the real world. These skills will hopefully make them not only more appealing to potential employers, but make them more productive members of society.

We also want to give them skill that will make them valuable to the work force. Computer programming, especially in MATLAB, is a skill that can go directly on a resumé. It can also open doors to new scientific explorations.

It is necessary for students to achieve these goals if they want to understand more deeply and systematically how to optimize complex systems. Without formal mathematical training in modern optimization techniques, it can be difficult to make good progress in efficiency.

3.3 Goals Reflected in Daily Course Structure

At the beginning of each new topic, we will spend a significant amount of class time on applications. We will look at many examples of real-world applications, and also discuss possible future applications.

Students will be given projects where they design models for real-world situations. They will get practice at solving these problems mathematically, and also at implementing them computationally.

We will also begin MATLAB instruction within the few few days of class, focusing first on writing basic programs, then on developing larger programs. We will have in-class MATLAB sessions, where I will first give a brief lesson on new techniques, then have hands-on, in-class programming time, where I will come around and help the students as they work. Note: It would be useful to have a TA or grad student help with this activity!

3.4 Goals of the Portfolio

The course is challenging on several levels. First, the students will likely have a wide variety of educational backgrounds. Some may be very skilled programmers, while others may be seeing programming for the first time. Some may have strong mathematical backgrounds, while others might not. There may even be some graduate students in the course. Moreover, the course is very open-ended. I could teach the entire course straight from the book, and have it be a very dry course, but this is not a good use of anyone's time in my opinion. Instead, designing a course portfolio will give me a chance to make the course dynamic and interesting for students (and also for myself!). It will allow me time to plan activities and lessons which will be of use to the students, and also for me to learn more about the possible applications of the subject.

In addition, this is my first time teaching the course. I never even took a course on this subject, so I am really learning it for the first time. This means that I have my work cut out for me, not just in terms of learning the subject matter, but in terms of learning how it is applied, and how it relates to other subjects. The course design, other than the rough outline in the course bulletin, is largely up to me. I intend to do justice to the subject, and therefore this portfolio will be a major step in designing the course.

Moreover, the course being open-ended makes it difficult to assess. I want to develop good methods for assessing student progress, and also for understanding if my methods are helping them, or could be improved.

3.4.1 Specific Goals of the Portfolio

I want to design several projects around the main subjects in the course. I want these projects to be clearly relevant to real-world applications, or even to come directly from real-world data. This will take time to develop.

4 Benchmark Memo 2: Description of Course Activities

4.1 Teaching Methods and Classroom Time

Since classroom time is limited (2.5 hours per week), it is important that the course be structured efficiently to maximize student learning. A variety of methods will be employed, including lecture, question and answer, the "think-pair-share" method. In addition to these methods, there will also be live-coding instructional sessions, and free-coding sessions,

4.1.1 How Are The Methods Used Over The Course of The Semester?

In lecture, it will often be appropriate to assess how the students are keeping up. This is the point of the question and answer and the "think-pair-share" method. In think-pair-share, at various points during the lecture (say 2-5 in the course of an hour), the students are asked (verbally) a question about the lecture, and given a moment to think about it. They are often asked to write something down at this point, to solidify their ideas. They are next told to share their ideas with a neighbor. The students are then informally asked to report their thoughts.

In coding sessions, the students code on their laptops, sometimes with a friend, and I observe their progress and resolve individual difficulties.

4.1.2 How Do The Methods Facilitate Course Goals?

Think-pair share is a quick activity that allows the student to engage more deeply with the material. It also gives me a chance to assess how they are doing, and judge whether the lecture should be slowed down or sped up.

Since much of the material is theoretical, it is natural that much of the course time is devoted to lecture. However, the in-class coding sessions are very important. In particular, when students are learning to code, their progress can be completely halted by minor bugs that they don't know how to find or get rid of. Therefore, it is important to have an instructor present as the students are first learning, so that they don't get discouraged by minor difficulties. This facilitates the students in learning on their own, as they see different methods for finding and resolving bugs.

4.2 Course Activities Outside of Class

Outside of class, student will have longer, more involved projects to work on. These projects will involve combining several smaller pieces learned about in class or on previous projects. Students will also be assigned homework problems.

4.2.1 Reasoning Behind The Structure

The reason to have this kind of a structure is twofold. Firstly, students need expose to quick, easy problems when they are learning new concepts. This makes the concepts easier to digest, and also gives encouragement to students. These learning opportunities correspond to example seen in lecture and small coding problems worked on in class. Secondly, students also need exposure to deep and interesting problems to see how ideas can be combined. This promotes creativity and helps student to master the topics. These learning opportunities correspond to the homework problems and programming projects.

4.2.2 Learning Goals of Each Activity

The goal of the homework problems is to get students to dig deeply into problems so that they can learn the underlying ideas, and not just the surface-learning that they may gather from lecture. In particular, they will have a chance to work though the details of problems. They will likely take many wrong steps, which is something that is rarely done intentionally in lecture (although sometimes it can be worth pointing out the

wrong steps to the students). In correcting their mistakes, they will learn to be more careful and find new and creative ways to handle the material. These are steps toward mastery.

The learning goals of the in-class coding sessions are to get the students to be familiar with basic programming structures, such as loops, conditionals, variable assignment, and so on. The goals also include one of the most important parts of learning programming, which is debugging code. Debugging is nearly an artform, and requires both creativity and skill. However, it can be difficult to teach debugging directly, since the students do not at first have a personal connection with it. It is much better to wait until student arrive at a bug themselves. Debugging then becomes much more relevant, and students are eager to learn their way out of pitfalls.

4.2.3 Assessing Student Performance

Assessing student performance in lecture has already been discussed in the "think-pair-share" discussion above.

Assessing student performance in homework is relatively straight-forward, and is done by grading the homework for both quality and correctness. Samples of student homework are given below in Section 11.

Student performance in classroom coding is done in class by assuring that every student gets the basic code running before they leave (student who need to leave early can be worked with on an individual basis later on).

4.3 Course Materials

The textbook is *Linear and Nonlinear Optimization*, 2nd Edition. Society for Industrial Mathematics (SIAM). ISBN-13: 978-0898716610, by Igor Griva, Stephen G. Nash, and Ariela Sofer.

The students will also use the Matlab software suite.

Course notes are provided online to the students. To see copies of these notes, see the homework solutions in subsection 11.2.

4.3.1 Rational for the Use of the Course Materials

The course book is a standard, well-respected text. More importantly, it contains great explanations and exercises, and covers nearly all of the material the course is designed to cover.

As for the choice of Matlab, there are many reasons. The most important is that Matlab is a very easy language/system to learn to program in. Unlike languages like C++ or Java, students do not need to learn about different number systems and variable declarations before they start coding. Instead, one can just begin coding, almost in the same way one might write on a blackboard. Moreover, Matlab is a very forgiving language, and will warn but not error out for a wide array of mistakes. In addition, the fact that student have a readily available command-line and don't have to learn about compiling their code makes learning coding in Matlab even more accessible. Next, the built in IDE (integrated design environment) in Matlab makes learning debugging extremely easy, since students can "pause" their code live, and see what values are held by the variables. This can be done in other languages as well, using tools such as `gdb`, but this is a somewhat advanced technique, while in Matlab, it is at the user fingertips.

Finally, one may want to use a language such as Python instead of Matlab. While there is a strong argument that can be made for Python, Matlab really wins out in two areas. The most obvious is in graphing. No additional modules or libraries needed to be imported. Simply `plot(x,y)`, and you can see a plot. Next, Matlab ("Matrix Laboratory") is designed to bee very good at handling arrays, which is of prime importance in mathematics.

4.3.2 How Should Students Use The Materials?

Students will use the materials of the course both in class and at home. They will have the opportunity to work loosely in groups, and learn from each other. Their final submissions should be their own work however.

4.4 Rational for Choice of Teaching Methods

Lecture is crucial for getting across key theoretical points, and can also be useful for seeing simple examples. Homework expands these examples or uses them as building blocks towards deeper learning.

In-class coding is useful for practice, and for learning debugging techniques. This hands-on learning can give a very helpful boost toward learning the basics of programming. The programming projects then allow students to take these ideas to a deeper level.

4.4.1 Influences of Wider Curriculum

These days, it is becoming increasingly necessary for students to learn programming if they want to be competitive on the job market or academic market. Moreover, having the skill of programming can be useful in other courses, or on research projects that students may be engaged in.

Also, the mathematical subject of the course, optimization problems, appear in a wide variety of disciplines, such as in physics, business, population dynamics, logistics, and many others. Therefore, it is likely that students will find a use for this material in many other subjects.

4.4.2 Impact of Materials on Student Learning

The step-by-step approach outlined above should lead students into the subject gently, so they do not get intimidated, but allows for plenty of challenges to improve their skill as time goes on. This approach will hopefully maximize student learning of these topics.

4.5 Course Choices and the Broader Curriculum

It is important that the material ans subjects taught in the course have a connection with previous courses, with future courses, with the wider disciplines in science, and also with practical applications in science, industry, and engineering. I briefly outline some of these connections below.

4.5.1 Building Upon Material From Previous Courses

This material directly enhances material learned in calculus and linear algebra. It uses these subjects as tool, directly extends some of the theoretical tools, and shows how they can be used in a very applied setting.

4.5.2 Preparing Students for Broader University and Department curriculum

The course develops a lot of fundamental mathematics, such as the theory of symmetric positive definite matrices, the spectral theory for matrices, convergence of iteration schemes, and many other tools which are ubiquitous in mathematics, fundamental science, and engineering.

4.5.3 Effect On Students' Future Courses And Career

Another reason for choosing Matlab is that it is often desired on resumes by employers, as opposed to languages/programs like Mathematica, Maple, Octave, Sage, etc. Therefore, Matlab was very consciously chosen to benefit students' future career options.

5 Benchmark Memo 3: Document and Analyze Student Learning

5.1 The Nature of Student Understanding (in each focused activity analyzed)

Here, I describe some of the activities involved in growing and enhancing student understanding.

5.1.1 Evidence of Students Meeting Learning Goals

A major student learning goal was to get the students to learn basic programming and to become more comfortable with attacking a problem using programming. Indeed, the data shows that the students did indeed improve what they knew, as discussed in detail in section 8 below. Moreover, there is also a clear increase in confidence in programming based on the survey data presented in that section.

In terms of the theoretical material, the high scores on exams is at least somewhat indicative of the learning achievements of the students. The grading of a mathematics exam is only slightly subjective, and therefore it is nearly impossible for the students to do well on the exams without having learned a fair amount. Therefore, the homework and lectures appear to be working.

5.1.2 Differences in Student Understanding Shown In Work Samples

It is clear that some students struggled with the concepts, and some concepts were never fully grasped even by the time of the final exam for a few students. This is discussed further in the section 10 below. On the other hand, some students achieve near perfect understanding, in so far as it can be judged from exam scores.

Regarding the programming samples (see section 7.3), essentially all of the students seemed to have grasped the concepts of using loops and conditionals, which was a major goal of the course.

5.1.3 Analysis of Student Understanding

5.1.4 What Students Are Actually Learning: Outcomes and Misunderstandings

It still seems difficult to know whether this learning will be used in different contexts; however, more students indicated that they would likely use programming in other courses, as discussed in section 8.

5.2 Distribution of Student Performance

In terms of the final grades, 12 of the 21 students were in the "A" range, and 6 were in the "B" range, which is a very nice distribution for a course of this level of difficulty. One student failed the course, but this appeared to be due to some extra-curricular difficulties the student was having, and the student was warned multiple times that this would be a likely outcome due to extremely low grades on the exams, and no homework turned in.

5.3 Student Performance And The Broader Curriculum (For Overall Course)

We next discuss student performance in the course, its outcomes, and its relationship to the broader course curriculum.

5.3.1 Did Distribution of Student Work Meet Expectations For Intellectual Goals?

The exams and homework seemed to meet expectations, and I could clearly see the students learning a lot as they challenged themselves to learn the material in preparation for the exams, and in completing their assignments. In retrospect, it would have been better to have at least one or possibly two more projects. I would also have like to base these projects on real data, which is something I was planning to do, but ended up cutting out due to time constraints.

Paradoxically, we might have had more time if we had more frequent programming projects. This is because a slightly faster pace may have pushed the students to learn the material more quickly, rather than

having large gaps of time when they were not programming. These gaps also meant that for some students who were just learning programming for Matlab for the first time, they had to spend some time remembering how things worked at the begging of each project, rather than having it fresh on their minds.

5.3.2 Preparation For Other Courses: Evidence Via Documented Student Performance

It is clear from the comparative case studies in section 8 that the student clearly mastered basic programming concepts. This would be a stumbling block in any future course which requires any amount of programming. Therefore, we have documented evidence of improvement in student performance. Moreover, in the coding contest portion (section 7), some of the more advanced students were able to push themselves to new heights, which one student's code reaching far beyond what was discussed in class. Therefore, we see that students both at lower-level of ability and at higher levels were able to noticeably improve their knowledge and abilities.

Just as important to improvement in student ability was the strong improvement in student confidence regarding programming. This is documented in section 8, where 66% of the students had an increase in the confidence in writing a computer program, and no student had a decrease (an additional 19% were already at maximum confidence).

Given these very positive measures in terms of programming ability and confidence, the students should be well-prepared for future courses involving programming.

5.3.3 What Does Student Work Say About Their Prior Preparation?

We know from the surveys in section 8 that some students had little programming experience, and some had none at all. Moreover,

5.3.4 Possible Changes, Redesigns, And Future Plans

It would be interesting to give students an open-ended project which could be tackled with either programming methods or theoretical methods, and see which one they opt for. It would be especially interesting to compare this with programming confidence measures. A follow-up study would also be interesting, where we could ask if the students had found uses for programming in other courses or other contexts.

6 Syllabus

Below is the syllabus for the course.

MATH 433/833: Nonlinear Optimization
UNL, Spring 2016, Section: 001, CRN: 4345/4346
Lecture: T, R, 11:00 am-12:15 pm, Avery Hall 109

Instructor:

Dr. Adam Larios **Email:** alarios@unl.edu
Office: Avery Hall 305 **Math Dept. Phone:** (402) 472-7250
Office Hours: M,W,F, 11:00 am - 12:00 pm, or by appointment
Web: www.math.unl.edu/~alarios2/courses/2016_spring_M433/content.shtml

Prerequisites:

MATH 314/814 (Linear Algebra), and MATH 310 (Abstract Algebra) or MATH 325 (Elementary Analysis). You are also expected to know differentiation and integration techniques from calculus, as well as the material from multivariable calculus. You are also expected to be able to write mathematical proofs. This course will require basic computer skills. We will learn programming, but prior knowledge of programming is *not* a prerequisite.

Textbook:

Igor Griva, Stephen G. Nash, and Ariela Sofer. *Linear and Nonlinear Optimization*, 2nd Edition. Society for Industrial Mathematics (SIAM). ISBN-13: 978-0898716610.

ACE Outcome 3:

"Use mathematical, computational, statistical, or formal reasoning (including reasoning based on principles of logic) to solve problems, draw inferences, and determine reasonableness." Your instructor will provide examples, you will discuss them in class, and you will practice with numerous homework problems. The exams will test how well you've mastered the material. The final exam will be the primary means of assessing your achievement of ACE Outcome 3.

Contacting me:

NOTE: Because of privacy rights, **I cannot discuss grades over email or telephone. Please do not email me asking about your grade. I will not be able to give you any information.** Of course, I am happy to discuss grades in my office.

Description:

Mathematical theory of unconstrained and constrained optimization for nonlinear multivariate functions, particularly iterative methods, such as quasi-Newton methods, least squares optimization, and convex programming. Computer implementation of these methods.

Motivation:

Nonlinear optimization is a generalization of the material in the multivariable calculus course dealing with finding and analyzing critical points, solving global extremum problems, and constrained optimization using the Lagrange multiplier rule. The machinery of linear algebra makes it easier to state nonlinear optimization problems and discuss the mathematical theory. Most nonlinear problems are too complicated to solve by hand, so numerical methods for optimization are an important component of any study of the subject. Nonlinear optimization problems can be broadly classified as unconstrained and constrained optimization, with similar theory but very different methods. We will explore tools that have been developed to handle this beautiful and very useful subject.

Homework:

Homework is designed to help students understand the material, to prepare them for exams, and to give them experience writing programs based on the material. The given exercises represent a minimal assignment. Some students may have to work additional exercises or do additional reading to attain sufficient mastery of the material.

I may also assign practice exercises that will not be collected. Homework problems must be written neatly in narrative English with mathematics embedded. Raw calculations with no explanation will not be accepted. Work that is not sufficient to earn 3/4 of the points on any given problem will be returned with no score. You may resubmit such work, but resubmissions will not be awarded more than 3/4 of the points for the assignment.

Reading & Exercises:

You are expected to read the appropriate sections of the text **before** coming to the class meeting in which the topic is scheduled. You are also expected to work through the indicated exercises after the corresponding material is presented in class, and **before** the next class meeting.

Attendance & Preparation:

There is no textbook that exactly fits the course goals; hence, the lecture notes are the primary record of the course. Regular attendance and attention is therefore critical. It will be helpful for you to browse through the material before it is presented in class.

Daily attendance for class lectures is expected and is extremely important. While attendance is not recorded, missing even **one** class will put you behind. Note that there is a strong correlation between class absences and poor grades. You are responsible for all material and announcements in class regardless of whether or not you attended. **You are also responsible for making arrangements with another classmate to find out what you missed. You should not ask me to go over material you missed (due to tardiness or absences) during office hours or over email.**

If you know ahead of time that you will miss a deadline, exam date, etc. Please let me know as soon as possible in advance. Reasonable accommodations will be made for university-excused absences.

Computing:

We will be writing programs to implement numerical methods. I will prepare directions for writing in Matlab; however, you are welcome to use a different language, such as python or C++ (although I request that you do not use Java). You may use your own computing equipment, and you may also use the computers in the

Math Department computer lab in Avery 9, or in labs around campus. Matlab is free to download for UNL students, and can be accessed here:

`http://procurement.unl.edu/matlab-licenses`

Collaboration:

Collaboration is encouraged in this course. However, copying someone else's work and submitting it as your own is unacceptable. This act of academic dishonesty will be prosecuted in accordance with university policy.

Electronics:

You are not allowed to have on your person during exams or quizzes any device that can access the internet or communicate in any way. Cell phones, Apple watches, etc. should be put away in backpacks/purses. Calculators, laptops, tablets, cell phones, and other non-medical electronic devices are not permitted during exams unless otherwise stated. During class, cell phones should be set on vibrate or off. If you need to take a call, send a text message, etc., please quietly leave the classroom to do so, so that you do not distract other students. You are welcome to return to class quietly when you are finished. If you wish to take notes using an electronic device, you must first demonstrate to me that you can type or write fast enough to do so properly, and that you can do it without distracting others, before the privilege to use such devices may be granted. If you are found to be abusing this privilege, you risk forfeiting it.

Grading:

Your course grade will be based on a weighted average computed as follows.

Homework:	30%
Midterms:	$2 \times 20\% = 40\%$
Final Exam:	30%
Total:	100%

Grading:

Your minimal course grade will be computed as follows. All work in the course will be graded according to the following scale:

A: 90, A-: 87, B+: 84, B: 80, B-: 77, C+: 74, C: 70, C-: 67, D+: 64, D: 60, D-: 57

If deemed necessary, minor adjustments to this scale will be made in favor of the students (commonly known as "applying a curve"). A grade of "A+" may be assigned in the case truly exceptional work.

Make-up exams:

Make-up exams will only be given with written evidence of an official university excused absence.

Incompletes:

A grade of "incomplete" may be considered if all but a small portion of the class has been successfully completed , but the student in question is prevented from completing the course by a severe, unexpected, and documented event. Students who are simply behind in their work should consider dropping the course.

Programming:

This course contains a gentle introduction to scientific computing with Matlab. Matlab is one of the most widely-used programming languages in science, mathematics, and engineering, and can be a very strong asset to future scientific work. **No previous programming experience is assumed.** Student are assumed to be able to have basic computer skills, such as using a mouse, keyboard, etc., and be able to download and install programs and navigate websites. Basic programming in Matlab will be taught in class on designated days. Programming assignments and/or projects will be announced in class.

ADA Statement:

Students with disabilities are encouraged to contact the instructor for a confidential discussion of their individual needs for academic accommodation. It is the policy of the University of Nebraska-Lincoln to provide flexible and individualized accommodation to students with documented disabilities that may affect their ability to fully participate in course activities or to meet course requirements. To receive accommodation services, students must be registered with the **Services for Students with Disabilities (SSD) office**, 132 Canfield Administration, 472-3787 voice or TTY.

Grade Questions:

Any questions regarding grading/scoring of homework, exams, or projects must be made within two class days from when they were handed back, or no change in grade will be made.
NOTE: Because of privacy rights, **I cannot discuss grades over email or telephone. Please do not email me asking about your grade. I will not be able to give you any information.** Of course, I am happy to discuss grades in my office.

Important Dates:

Jan. 22, 2016 (Fri):	Last day to withdraw from this course and not have it appear on your transcript.
Mar. 4, 2016 (Fri):	Last day to change your grade option to or from Pass/No Pass.
Mar. 20-27, 2016:	Spring break, no class.
Apr. 8, 2016 (Fri):	Last day to drop this course and receive a grade of W. (No permission required.) After this date, you cannot drop.

Departmental Grading Appeals Policy:

Students who believe their academic evaluation has been prejudiced or capricious have recourse for appeals to (in order) the instructor, the departmental chair, the departmental appeals committee, and the college appeals committee.

Final Exam Policy:

Students are expected to arrange their personal and work schedules to allow them to take the final exam at the scheduled time. The final exam for this course is:
Tuesday, May 3, 2016, 3:30 pm-5:30 pm (same classroom).

Disclaimer:

While this syllabus was prepared carefully and according to information available at the beginning of the semester, changes may be necessary in the interest of good teaching. Changes to any of the information above will be announced in class and posted on the class web site. This includes in particular possible updates or corrections to the syllabus, and changes of exam dates. Care has been taken to avoid any conflict between this syllabus and official university policy. Any such conflict, if it exists, is purely accidental, and appropriate measures will be taken to rectify any such mistake.

Tentative List Of Topics:

The following tentive list of topics is a rough guide to the material covered in the course, but is subject to change. Updates and changes to the content will be announced in class, over email, on blackboard, or on the course website.

- Introductory Material: **Sections 1.5-7, 2.2-7**.
  (approximately 5 classes)
- Theory of Unconstrained Optimization: **Sections 11.2-5**.
  (approximately 6 classes)
- Methods for Unconstrained Optimization: **Sections 12.2-3**.
  (approximately 4 classes)
- Theory of Constrained Optimization: **Sections 3.1-3, 14.2-5**.
  (approximately 7 classes)
- Methods for Constrained Optimization: Selected topics from **Chapters 15-16**.
  (approximately 4 classes)

Most likely, Exam 1 will be on Chapters 1, 2, and 11 and Exam 2 will be on Chapters 12, 3, and 14, but again, this is subject to change.

#	Week	Day	Date	Notes
1	1	T	1/12	First day
2		R	1/14	
3	2	T	1/19	
4		R	1/21	Jan.22: Last day to withdraw without a W
5	3	T	1/26	
6		R	1/28	
7	4	T	2/3	
8		R	2/5	
9	5	T	2/9	
10		R	2/11	
11	6	T	2/16	
12		R	2/18	
13	7	T	2/23	
14		R	2/25	**Exam 1** (in class)
15	8	T	3/3	Mar. 4: Deadline to change Pass/No Pass Grade
16		R	3/5	
17	9	T	3/8	
18		R	3/10	
19	10	T	3/15	
20		R	3/17	
–		T	3/22	Spring break
–		R	3/24	Spring break
21	11	T	3/29	
22		R	3/31	
23	12	T	4/5	**Exam 2** (in class)
24		R	4/7	April 8: Last day to drop
25	13	T	4/12	
26		R	4/14	
27	14	T	4/19	
28		R	4/21	
29	15	T	4/26	
30		R	4/28	
–		T	5/3	Final Exam 3:30pm-5:30 pm (same classroom)

7 Coding Contest

To motivate the students, we had a coding contest to see who could write the fastest code. Since it is also important for code to be readable, we also had a prize for the easiest to read code. The prize for fastest code was objective and easy to evaluate: whoever had the fastest running code, that computed the answer to within a reasonable error, was the winner.

To decide who had the "easiest to read" code, we examined the codes on the projector, and students voted for their favorite one. To keep the voting easy, we compared two codes, voted on the winner, and that winner went on to be compared with the next code. This means that if N codes were submitted, only $N-1$ comparisons needed to be made, so the voting process went fairly quickly.

A major benefit to the "easiest to read" part of the contest was that students had the chance to see code written by their fellow students. They saw mistakes other students made, improvements, different styles, and different ideas. I think this was one of the best parts of the contest, and it was completely unexpected. I would certainly do this again in any future programming class.

7.1 Description of the contest

To formally announce the contest, I sent the following email to the students.

Dear all,

Now that you (hopefully) have a working conjugate gradient code, let's see who can write the fastest code!

This is an extra credit assignment. If you have the fastest code (as measured by tic and toc), or the code which is voted "easiest to read", for a possible total of 10 extra points on your homework! Thus, it is possible to score more that 100% on you total homework grade, which is worth 30% of your final grad in the course.

Here are the parameters:

Run your conjugate gradient code with 1e-6 on a matrix of size 10,000 (that is, size 1e4). Don't run it on just any matrix! Use a sparse matrix. To randomly generate a sparse matrix which is SPD, you can do this:

```
rng(433);
n = 10000;
ev = rand(1,n);
A = sprandsym(n,2/n,ev);
```

Try to make your code as efficient as possible. You might try re-writing the code slightly, using a different initial guess, or maybe some other idea. To submit your code, email it to me before 9:00 am on Tuesday. PLEASE stick to the following format (i.e., have the following line as the first line of your code):

```
function [x iter] = conjugateGradient(A,b,x0,maxIter,tol)
```

This way, it will be easy to test. I will take your name off the code (so nobody will be singled out), and seed the RNG with a different value, then we will test them in class! The fastest code (or codes) will receive an extra 5 points. The class will then vote on the most readable code, and the winner will receive an extra 5 points.

Have a great weekend, and good luck!

Best regards,
Adam

Note: All codes were run and displayed anonymously, with the students being given a strip of paper with their number on it that I distributed in class so that each student could only see their own number. However, the results are presented anonymously here.

7.2 Results of the speed trial

Here, we see the data collected from the speed test, organized by running time. Note the variation in errors, with some students sacrificing accuracy for the sake of speed. This was interesting, so I allowed it. Entry #13 was the clear winner, with only 2 iterations, the fastest time by far, and near machine-precision accuracy. However, this student used an advanced preconditioning step which he learned about due to his own reading. I thought this was great, and therefore I allowed it, but still gave points to some of the top winners. The last code was written incorrectly, and did not even converge.

Entry #	Iterations	time (seconds)	error
13	2	0.010669	1.09153772740018e-15
12	145	0.075274	0.000470572884477261
4	161	0.076064	0.00024677174066561
7	250	0.111684	3.86621845572846e-06
8	251	0.111891	3.86621845572846e-06
10	250	0.112647	3.73794382906209e-06
5	251	0.117549	3.86621845572846e-06
6	250	0.117859	3.86621845572846e-06
9	320	0.120432	1.76652534413823e-07
1	250	0.122078	3.86621845572846e-06
2	250	0.122921	3.86621845572846e-06
11	250	0.126107	3.86621845572846e-06
14	250	0.211774	3.8662184555921e-06
3	∞	∞	∞

The code that won the fastest code award is shown on the next page. Additional codes are shown in the readability tests on the proceeding pages.

Contest entry #13. Winner: Fastest Code.

This student went beyond the textbook and classroom examples to investigate more sophisticated methods. The student used the Cholesky factorization, which greatly speeds up the code. No other student did this. It is possible the student simply found an example of this online, but this at least means they are doing some kind of research.

```
function [x,iter]=iccg(A,b,x0,maxiter,tol)
%   Function: solving Ax=b for a large sparse SPD matrix A with ICCG method
%   Parameters:
% input:
%   A: the sparse SPD matrix
%   b: the right-hand side vector
%   x0: the initial guess
%   tol: the maximum tolerance
%   maxiter: the number of max iterations
% output:
%   x: the final iteration
%   iter: the actual iteration times

x=x0;
r=b-A*x;                % r:residual(r0)
%test if x0 meets the requirement by a stroke of fortune
if abs(r)<=tol
    iter=0;
    return
end
%incomplete cholesky factorization
L= ichol(A, struct('type','ict','droptol',1e-8));
r=L\r;                  % r:preconditioned residual(r0)
p=L'\r;                 % p:preconditioned search direction(p0)

% go to the normal iteration
for iter=1:maxiter
old_rr=r'*r;            % store r_i*r_i
z=A*p;                  % z_i
alpha=old_rr/(p'*z);    % alpha:step length(alpha_i)
old_x=x;                % store x_i
x=x+alpha*p;            % x_i+1
% test if x_i+1 meets the requirement
if abs(x-old_x)<=tol
    return
end
r=r-alpha*(L\z);        % r_i+1
new_rr=r'*r;            % store r_i+1*r_i+1
beta=new_rr/old_rr;     % beta_i
p=L'\r+beta*p;          % p_i+1
end
warning ('Maximum iterations reached . May not be converged . ');
end
```

7.3 Results of the readability trial

In this section samples of several codes submitted by the students are included, presented here essentially unedited. Note that the winning code (code #7 as voted on by the students, and I happen to agree with their choice) first describes the code in detail in the comments at the beginning. It then has small comments in the code itself, but not enough to distract the user from the algorithm. First, we take a look at a code that the students rejected immediately.

Contest entry #2. Example of poorly written code. No explanation, and the few existing comments mostly just restate what the algorithm is doing, which adds no real information or clarity. It is also poorly indented. (I warned the students many times to use auto-indent in Matlab.)

```
function[x,iter]=conjugateGradient(A,b,x0,maxIter,tol)
%%
x=x0;
r=b;
p=0;
%%% i=0 the 1st iteration
% belta=0;
R=r'*r;
% p=r+belta*p;
p=r;
K=A*p;
alfa=R/(p'*K);
% x=x+alfa*p;
x=alfa*p;
% r_minus=r;
R_minus=R;
r=r-alfa*K;
%%% From the 2rd iteration to begin
for i=1:maxIter
R=r'*r;
belta=R/R_minus;
p=r+belta*p;
K=A*p;
alfa=R/(p'*K);
x=x+alfa*p;
R_minus=R;
r=r-alfa*K;
    if norm(r)-tol< 0
        break;
    end
end
iter=i+1;
```

Contest entry #7. Winner: Most Readable Code.

```
%       March 15 2016

%   conjugateGradient : Solve Ax=b by conjugate gradients
%   [x, iter] = conjugateGradient(A,b,x0,maxIter, toler)
%
%   Given symmetric positive definite sparse matrix A and vector b,
%   this runs conjugate gradient to solve for x in A*x=b.
%   It iterates until the residual norm is reduced by 10^-6,
%   or for at most the maximum number of iterations.
%
%   Inputs:
%   matrix A
%   vector b
%   vector x0
%   maxIter - maximum number of iterations
%   tol - tolerance
%
%   Outputs:
%   vector x
%   number of iterations


function [x, iter] = conjugateGradient(A,b,x0,maxIter, toler)

%Initialization values
iter =0;
x = x0;
w = A*x;
r = b-w;
p = r;
d = p'*p;

%Defining when the algorithm must stop
while norm(r) > toler && iter < maxIter
    iter = iter+1;
    Ap = A*p;
    %Defining the scalar alpga
    alpha = d / (p'*Ap);
    %Updating x based on the last x, alpha and p
    x = x + alpha * p;
    %Updating the Residual
    r = r - alpha * Ap;
    %Storing the last value for d in the variable dlast
    dlast = d;
    %We define Ap = r, so:
    d=r'*r;
    %Calculing p based on the previous p, d and dlast.
    p= r + (d/dlast)*p;
   end
```

Contest entry #4. Example of overly commented code.

```
function [x,iter] = conjugateGradient(A,b,x0,maxIter,tol)
% Use the Conjugate Gradient method to solve Ax = b.
% Outputs the approximate solution and number of iterations.
    % Convert the supposedly "sparse" matrix into sparse format
    A = sparse(A);
    % Set the first "solution" as the given initial guess.
    x = x0;
    % Calculate the residual for the initial guess.
    r = b;
    % Calculate the square of the residual and save the result
    % since we will use it multiple times.
    rTr = r'*r;
    % Initialize the search direction.
    p = r;
    % Start the iteration count.
    iter = 0;
    % Start iterating until the given number of max iterations.
    % Stop iterating when the error (i.e. the norm of the residual)
    % is below the given tolerance.
    while sqrt(rTr)/norm(x) > tol
        % If the max number of iterations has been
        % reached, stop iterating.
        if (iter > maxIter);  break; end
        % Increment the iteration count
        iter = iter + 1;
        % Multiply A by the search direction and save the result
        % since we will use it multiple times.
        Ap = A*p;
        % Calculate the next step size
        alpha = rTr/(p'*Ap);
        % Calculate a (hopefully) better guess for the solution
        % Using the new step size and search direction.
        x = x + alpha*p;
        % Store the previous square residual.
        rTr0 = rTr;
        % Compute the next residual.
        r = r - alpha*Ap;
        % Store the new square residual.
        rTr = r'*r;
        % Calculate the scalar Beta used in determing the search direction.
        B = rTr/rTr0;
        % Calculate the next search direction.
        p = r + B*p;
    end % End of iterative scheme
    % Warn the user if the max iterations were reached.
    if iter == maxIter
        warning('Max iterations reached. Method may fail to converge.');
    end
end % End of function conjugateGradient
```

8 Comparative Case Studies

The following pre-course and post-course surveys, included below were given to the students. The questions vary slightly on the first page, since the pre-course survey asks for information that only needs to be obtained once (e.g., have you taken a programming course before?), and the post-course survey asks for information that couldn't be obtained at the beginning of the course (e.g., reflections on the course).

The next set of questions (questions 4-6 on the pre-course survey) are essentially aimed at testing programming confidence.

One question is aimed directly at algorithmic thinking. This is the equations about factorials. The factorial $6! = 1 \times 2 \times 3 \times 4 \times 5 \times 6$ can be computed by finding "easy" combinations of numbers (such as using $2 \times 5 = 10$, etc.), which is not a very algorithmic way to compute it. It can also be computed by simply multiply one number after the other. I was curious to know if after taking a programming course, students would start computing it more algorithmically. Only 4 students computed it "algorithmically" in the "pre" test, but strangely, only 3 students did it in the "post" test, with only one student using an algorithmic approach in both surveys. Moreover, it did not seem to be a good predictor of final grades, with the students who used the algorithmic approach on at least one survey receiving grades A, B, D, and F. Therefore, this seemed to be a failed test. This small-scale test made me reconsider testing this metric in future studies.

The next questions are taken from a test devised by Professors Saeed Dehnadi and Richard Bornat [S. Dehnadi, R. Bornat, *The camel has two humps*, published online at http://www.eis.mdx.ac.uk/research/PhDArea/saeed/ (2006)]. The interpretation of the results given by the authors has become controversial, as they themselves discuss on the website previously cited (they imply that the results say something about the innate programming abilities of people). I make no comment on those matters. However, the survey itself appears to be one of the few existing surveys that tests very basic programming concepts. Therefore, I decided to use it to see how the results might vary as students took the course and became more familiar with programming.

These surveys were not anonymous, because they needed to be linked to the results on the previous tests. However, the results are displayed anonymously here.

In this section, we first present the surveys in their raw forms, essentially as the students saw them. Then we discuss the results. The first survey begins on the next page.

8.1 Pre-Course Survey

1. (a) Have you ever written a computer program in any language?
   Circle one: `Yes` `No`
   (b) If so, in which language(s)? (If multiple, name only your top 3.)

2. Have you ever taken a course that involved at least some programming?
   Circle one: `Yes` `No`

3. Have you ever taken a programming course in a computer science department?
   Circle one: `Yes` `No`

4. On a scale of 1 (lowest) to 10 (highest), how would you rank your current level of confidence in writing a "simple" computer program? (Circle a number.)

   `1 2 3 4 5 6 7 8 9 10`
   `(low) (high)`

5. On a scale of 1 (lowest) to 10 (highest), how useful do you think programming ability will be to you after graduating from college? (Circle a number.)

   `1 2 3 4 5 6 7 8 9 10`
   `(low) (high)`

6. On a scale of 1 (lowest) to 10 (highest), how likely do you think you will be to use programming to help solve problems from other classes, even if you are not required to? (Circle a number.)

   `1 2 3 4 5 6 7 8 9 10`
   `(low) (high)`

7. Compute 6! (6 factorial) by hand, showing all your steps in detail.

(Please see the questions on the next page too.)

Please try to answer these equations to the best of your ability. It is OK if you are unsure; just write something for each answer. Doing your best will help me understand where the class is at. **Don't change your answers or erase!** Just put down the answer you are thinking out at the time.

8. Read the following code and write the correct answers in the provided spaces.

```
1 a = 10;
2 b = 20;
3 a = b;
```

`a =` _____, `b =` _____.

9. Read the following code and write the correct answers in the provided spaces.

```
1 a = 10;
2 b = 20;
3 b = a;
```

`a =` _____, `b =` _____.

10. Read the following code and write the correct answers in the provided spaces.

```
1 big = 10;
2 small = 20;
3 big = small;
```

`big =` _____, `smallg =` _____.

11. Read the following code and write the correct answers in the provided spaces.

```
1 a = 10;
2 b = 20;
3 a = b;
4 b = a;
```

`a =` _____, `b =` _____.

12. Read the following code and write the correct answers in the provided spaces.

```
1 a = 10;
2 b = 20;
3 b = a;
4 a = b;
```

`a =` _____, `b =` _____.

13. In line 1 in each of the above codes, what does the "`=`" sign mean?

8.2 Post-Course Survey

1. Is there anything you learned in the course that might be useful in your future academic or professional career?

2. Is there anything you learned that you weren't expecting to learn?

3. Is there anything you didn't learn that you wanted to learn?

4. How did the programming projects help your understanding of the material?

5. Is there anything you would change about the programming projects?

6. On a scale of 1 (lowest) to 10 (highest), how would you rank your current level of confidence in writing a "simple" computer program? (Circle a number.)

   1 2 3 4 5 6 7 8 9 10
   (low) (high)

7. On a scale of 1 (lowest) to 10 (highest), how useful do you think programming ability will be to you after graduating from college? (Circle a number.)

   1 2 3 4 5 6 7 8 9 10
   (low) (high)

8. On a scale of 1 (lowest) to 10 (highest), how likely do you think you will be to use programming to help solve problems from other classes, even if you are not required to? (Circle a number.)

   1 2 3 4 5 6 7 8 9 10
   (low) (high)

9. Compute 6! (6 factorial) by hand, showing all your steps in detail.

(Please see the questions on the next page too.)

Please try to answer these equations to the best of your ability. It is OK if you are unsure; just write something for each answer. Doing your best will help me understand where the class is at. **Don't change your answers or erase!** Just put down the answer you are thinking out at the time.

10. Read the following code and write the correct answers in the provided spaces.

```
1 a = 10;
2 b = 20;
3 a = b;
```

`a =` _____, `b =` _____.

11. Read the following code and write the correct answers in the provided spaces.

```
1 a = 10;
2 b = 20;
3 b = a;
```

`a =` _____, `b =` _____.

12. Read the following code and write the correct answers in the provided spaces.

```
1 big = 10;
2 small = 20;
3 big = small;
```

`big =` _____, `smallg =` _____.

13. Read the following code and write the correct answers in the provided spaces.

```
1 a = 10;
2 b = 20;
3 a = b;
4 b = a;
```

`a =` _____, `b =` _____.

14. Read the following code and write the correct answers in the provided spaces.

```
1 a = 10;
2 b = 20;
3 b = a;
4 a = b;
```

`a =` _____, `b =` _____.

15. In line 1 in each of the above codes, what does the "`=`" sign mean?

8.3 Results and Analysis

Some fact from the first few questions:

- 19 out of 21 students answer "yes" to "Have you ever written a computer program in any language?"
- Of these, 11 had used Matlab, 8 had used Java, 9 had used some form of C/C++/C#, 1 had used Fortran, 1 had used Python, and there were an assortment of in-house languages people used in labs.
- 19 out of 21 students answer "yes" to "Have you ever taken a course that involved at least some programming?" (one student had programmed before, but had not taken a course; I don't know how one student apparently took a course that involved programming, but did not write a program).
- 13 out of 21 students answer "yes" to "Have you ever taken a programming course in a computer science department?'

8.3.1 Measures of Confidence and Broader Scope

We next look at measures of confidence in programming, and how the students perceive their programming skills will be used in other context.

- On a scale of 1 (lowest) to 10 (highest), how would you rank your current level of confidence in writing a "simple" computer program?

Student #	Pre	Post	Change
1	10	10	0
2	8	10	+2
3	8	8	0
4	10	10	0
5	5	9	+4
6	2		
7	9	10	+1
8	6	9	+3
9	3	6	+3
10	6	9	+3
11	8	9	+1
12	5	8	+3
13	1	5	+4
14	10	10	0
15	8		
16	10	10	0
17	3	6	+3
18	9	10	+1
19	7	10	+3
20	6	8	+2
21	8	10	+2

- On a scale of 1 (lowest) to 10 (highest), how useful do you think programming ability will be to you after graduating from college?

Student #	Pre	Post	Change
1	10	9	-1
2	10	9	-1
3	8	8	0
4	10	10	0
5	8	9	+1
6	7		
7	10	10	0
8	9	9	0
9	7	4	-3
10	10	10	0
11	9	8	-1
12	9	8	-1
13	6	6	0
14	10	10	0
15	10		
16	9	8	-1
17	5	8	+3
18	9	10	+1
19	10	10	0
20	8	7	-1
21	10	10	0

- On a scale of 1 (lowest) to 10 (highest), how likely do you think you will be to use programming to help solve problems from other classes, even if you are not required to?

Student #	Pre	Post	Change
1	8	9	+1
2	10	9	-1
3	3	1	-2
4	10	10	0
5	8	9	+1
6	5		
7	10	9	-1
8	7	5	-2
9	8	3	-5
10	8	10	+2
11	8	10	+2
12	9	8	-1
13	4	6	+2
14	10	10	0
15	10		
16	9	10	+1
17	10	10	0
18	10	8	-2
19	9	10	+1
20	8	8	0
21	9	10	+1

Although there was a slight overall decrease in students thinking they would use programming in other courses or in their future career, we see the confidence in programming has increased significantly, with not even a single student having a decrease.

A variable that maybe have had an effect on the questions about using programming in other contexts may be that many of the students were graduating, and may have known much more about their future career at the point in time (some of them even had a job lined up by the end of the semester), while they may not have had as clear a picture at the beginning.

8.3.2 Measures of Basic Programming Knowledge

Here we report the results of the second portion of the survey. The questions are labeled by the numbers they were given on the Pre-Course Survey. The "Pre-" and "Post-" are abbreviated to "B" (before) and "A" (After). If the answer was incorrect, a mark of "X" is placed. Otherwise, a blank space is placed.

Student #	8A	8B	9A	9B	10A	10B	11A	11B	12A	12B	13A	13B
1												
2												
3											X	
4												
5												
6					X		X				X	
7	X		X									
8												
9												
10												
11							X					
12												
13							X		X		X	
14												
15	X		X									
16							X					
17												
18									X			
19												
20							X					
21												

In the above table, we see that several students had difficulties with one or more basic programming concepts before the course. Note that students #6 and #13 were the student who stated that they had never written a computer program in any language. However, after the course, every student received a perfect score. Therefore, we can see that the course had a measurable impact on the programming knowledge of the students. In particular, over 20% of the students missed question #11 before the course, but all were able to master the concept by the end of the course.

While this test examined only basic programming questions, the questions are based in logic and reasoning, rather than, e.g., memorization of facts or terms. This small study shows that these concepts can indeed be taught, and that it can be done over the course of the semester. In fact, I believe the students actually mastered these concepts much earlier, and it would be interesting to hold such a survey earlier in the semester.

9 Introduction to Matlab Worksheet

The following document was given to the students as a worksheet for them to learn the basic syntax and operations of Matlab. It also walks the student through writing very basic programs that teach core concepts in programming, such as loops and conditional statements, which act as building blocks for future codes.

Introduction to Matlab
by Dr. Adam Larios

It is important to type all of the code as you go along. You will learn it better this way: your brain will make connections based on what your hands are physically typing.

9.1 Matlab As A Calculator

Let's first learn the basic operations in Matlab. We will type them into the command window at the prompt which looks like "`>>`". For example, to compute `3+5`, we would do:

```
>> 3+5
```

(don't type the "`>>`") and hit `[Enter]`. Try these as well:

```
>> sqrt(2)
>> 3^2
>> 2*5
>> sin(pi)
```

Notice the last one gives you a result of `1.2246e-16`. This is computer short-hand for scientific notation, with the "`e`" meaning "times 10 to the power of". So,

$$\texttt{1.2246e-16} = 1.2246 \times 10^{-16} = 0.0000000000000012246$$

But, shouldn't $\sin(\pi) = 0$? What is going on here is that computers typically only have about **16 digits of accuracy**, so to a computer, `0` is the same thing as `0.0000000000000012246`, and all smaller positive numbers. This usually doesn't cause trouble, but it is important to be aware of because it can sometimes cause unexpected bugs in your code.

9.1.1 Comments

"Comments" are parts of a program that the computer doesn't read. They are intended for human use only, and can make a program much easier to read. **Good code is well-commented code.** In Matlab, comments are made by the "`%`" sign like this:

```
>> 2 + 3 % The text after the '%' sign does nothing.
>> % This line does nothing at all.
```

9.2 Variables

In mathematics, an equals sign "`=`" means "is equal to", but in Matlab (and most computer languages), it is used in a different way; namely, it is used for assignment. Try:

```
>> x = 5;
>> x
```

The first line assigns `x` the value of 5. The semi-colon "`;`" suppresses output (try it without the semi-colon too). The second line tells Matlab to print the value of `x`. This should be fairly straight-forward. However, consider the following code:

```
>> x = 5;
>> x = x + 2;
>> x
```

Here, the second line does not make much sense in mathematics, because in an equation like "$x = x + 2$", we could cancel the x to get $0 = 2$, which is false. However, remember that the "`=`" sign to a computer means assignment, so it computes the right-hand side `x + 2` first. Since `x` is equal to `5`, the result is `5 + 2` or `7`. The computer then assigns this value to the left-hand side, which is `x`, so now is equal to `7`.

Try these as well:

```
>> x = 10;
>> myVariable = 2.5 + 1e1; % Remember: 1e1 means 10^1 or 10
>> myVariable + x^2
```

9.3 Matrices and Vectors

Matlab is very good at handling matrices and vectors. In fact, Matlab stands for "**Mat**rix **Lab**oratory." In Matlab, everything is a matrix (also called an "array"). For example, numbers (i.e., scalars) are really 1×1 matrices. There are a few different ways to input matrices and vectors. Try the following examples:

> **Time-Saving Hint:** Use the up-arrow ↑ on your keyboard to cycle through old commands.

```
>> u = [4 5 6]
>> v = [4,5,6]
>> x = [4;5;6]
>> A = [1 2 3; 4 5 6; 7 8 9]
>> M = [1,2,3; 4,5,6; 7,8,9]
>> A*x  % Matrix time vector.
>> u*x  % Row matrix times column vector.
>> A'   % The transpose of A. It switches rows and columns.
```

9.3.1 Matrix and Vector Operations

As you can see form the last two examples, matrix multiplication works like it usually does in linear algebra. However, sometimes we will want to treat matrices and vectors as just tables of numbers, and we want to do element-wise operations on them. This is done my putting a dot before the operation. Try this:

```
>> [4 5 6].^2
>> [1 2 3; 4 5 6; 7 8 9].^2
>> u = [1 2 3];
>> v = [4 5 6];
>> u.*v
>> 2.^u
```

Matlab is smart enough to apply many common operations element-wise. For example:

```
>> t = [-pi, -pi/2, 0, pi/2, pi];
>> cos(t)
>> sin(t)
>> exp([1 2 3])
>> log([1 exp(1) 10 exp(2)])
```

Note: In Matlab "`log`" is base $e \approx 2.71828 = \texttt{exp}(1)$.

Matlab also makes many common operations easy. For example:

```
>> x = [-3.7 0 5.5 -9];
>> max(x)
>> min(x)
>> abs(x)
>> max(abs(x))
>> min(abs(x))
>> sum(x)
>> norm(x) % The squart-root of the sum of the squares of x.
>> sqrt(sum(x.^2)) % Another way to compute the norm of x.
```

9.3.2 Building Matrices and Vectors

Matlab has many built-in features to generate vectors easily. Let's make a list of numbers from 3 to 10. We can do it like this:

```
>> x = 3:10
```

If we want to "step" or "skip" by a certain amount, we put the skip-size in the middle:

```
>> 3:2:10
>> 3:0.5:10
```

It is very common to want a certain number of points between two fixed numbers. To get a vector with exactly 16 points between (and including) 3 and 7, use `linspace` like this:

```
>> x = linspace(3,8,16)
```

This saves time, since you don't have to figure out what the step-size will be, but you could also do it like this:

```
>> delta_t = (8-3)/16;
>> x = 3:delta_t:16
```

(Check out `logspace` as well.) Here are some shortcuts to generate special, useful matrices:

```
>> rand(2,3)  % A random 2 by 3 matrix.
>> eye(3,3)   % The 3x3 identity matrix.
>> zeros(2,3) % A 2 by 3 matrix of all zeros.
>> ones(2,3)  % A 2 by 3 matrix of all ones.
```

9.3.3 Accessing Matrices and Vectors

Often, we want to read or edit the entries of a matrix or vector. We can do this by passing the "index" of the component we want. Matlab indices start at 1. (Some languages, like C and C++, start at 0. Fortran starts at 1, like Matlab.) Try these examples:

```
>> v = [4 5 6];
>> v(2)
>> v(2) = 3*v(2);
>> v

>> A = [1 2 3; 4 5 6; 7 8 9];
>> A(2,3)
>> A(2,:)      % This ':' tells Matlab to use all columns
>> A(:,2)      % This ':' tells Matlab to use all rows
>> A(2:3,2:3) % This gives a 'submatrix' of A
>> A(:,3) = [0; 0; 0]
>> A(:,2) = v' % Remember: v' turns the row vector v into
               % a column vector
```

9.4 Getting Help

Matlab has a built-in help feature for almost every function. You may have to scroll up to read everything it outputs. For a more graphical (but sometimes slower) version `help`, type `doc` instead of `help`. Try these:

```
>> help max
>> help logspace
>> doc sin
```

Note: Professional programmers use Google all the time! Google is your friend. Men and women who gain skill at Googling their programming issues are men and women who get better at programming. One tip is to Google the exact or nearly exact error output from an error message, preferably using copy/paste.

9.5 Plotting

In Matlab, plotting works differently than in the symbolic-based math programs you may be used to. Here is the key point to keep in mind: **In Matlab, you have to tell it every point in the plot!** First, build a vector of the independent variables, and call it `x` (for example). Then, we build another vectors of `y`-values. These points are then plotted one-by-one. Below is an example. We type "`close all`" first to get rid of any other possibly open plot windows.

```
>> close all;
>> x = linspace(-2,2,10);
>> y = x.^2;
>> plot(x,y)
```

You can see that Matlab is just connecting the dots. To get better resolution, try `x = linspace(-2,2,100);` or `x = linspace(-2,2,1000);`. Next, you can plot in different colors and different symbols. Try these commands, after inputting an `x` and `y` as above. **Remember to type `close all` each time!**

```
>> close all;
>> plot(x,y,'g'); % plot in green
>> close all;
>> plot(x,y,'*'); % plot with stars
>> close all;
>> plot(x,y,'-*'); % plot with stars connected by lines
>> close all;
>> plot(x,y,'b-*'); % plot with stars connected by lines in blue
>> help plot % find out about other plotting options
```

9.5.1 Multiple plots

Matlab will wipe out all previous plots each time it plots something new. To avoid this, we use "`hold on`" like this:

```
>> close all;
>> x = linspace(-2,2,100);
>> plot(x,x.^2,'r');
>> hold on;
>> plot(x,x.^3,'b');
>> axis([-1, 1, -2, 4]);  % [x_min, x_max, y_min, y_max]
```

9.5.2 Labeling and Legends

Labeling your plots and plot axes is very good practice as a scientist, mathematician, engineer, etc. Let's see how to do this in Matlab.

```
>> close all;
>> t = linspace(-pi,pi,100);
>> plot(t,cos(t),'r');
>> hold on;
>> plot(t,sin(t),'b');
>> title('Input and Response');
>> xlabel('Time');
>> ylabel('Amplitude');
>> legend('Input signal','Response Signal');
```

The plot may have gone behind the main Matlab window, so try minimizing some windows to find it.

9.5.3 Subplots

Matlab has a wonderful ability to make different plot windows in the same figure. Try the following.

```
>> close all;
>> x = linspace(0,2*pi):
>> subplot(1,2,1);
>> plot(x, sin(x));
>> subplot(1,2,2);
>> plot(x, cos(x));
```

The first two arguments of `subplot` tell the number of rows and columns for the different plots. The last argument tells Maplab which plot num ber is the current plot. For more information, type `help subplot`.

9.6 Saving work in m-files

As can be seen from the example above, we often have many lines of code, and we don't want to have to enter them into the command line each time. Let's put these in a file. Matlab files end with ".m". Open the editor, and save a new file as "myTest.m". Type this in it (the numbers on the left are just line numbers; don't type them).

```
1  x = 5;
2  x = x + 2;
3  x
```

You can now run this script (m-file) by pushing the green "▷"(run) button.

Tip: If you ever write some code, and then want to change it, you don't have to delete it! A common technique is to just "comment it out" by highlighting the text and pressing the "Comment" button in the menu. Uncomment it with the "Uncomment" button.

9.7 Loops

Loops are one of the most useful constructs in programming. They often come up when you have to do something repetitive many times, making only small changes each time. Let's look at a basic "`for` loop" to get started in an m-file (call it `loopTest.m` or something).

```
1  x = 0;
2  for i = 1:10
3      x = x + i
4  end
```

Let's break down the parts of this loop to get the idea. We start by setting `x = 0`, since we are going to use it later. Then, lines 2 and 4 tell Matlab that it is going to start and end a loop. Note that the Matlab keywords `for` and `end` get highlighted differently when you type them. This is called "syntax highlighting."

This loop will repeat the instructions on line 3 each time it is run. The number of times it will run is determined by the "`i = 1:10`" after the "`for`" in line 2. On the first pass through the loop, i will equal 1. On the second pass, it will equal 2, and so on, until it counts all the way up to 10. (Recall that in Matlab `1:10` means the vector `[1 2 3 4 5 6 7 8 9 10]`, so Matlab is just running over all of those values).

Let's analyze what will happen to `x` as we go through the loop. Before the loop, `x` is set to zero, so on the first pass, `x = 0` and `i = 1`, so `x+i` will equal `0+1=1`, and line 3 sets `x` to this value. On the next pass, `x = 1` and `i = 2` (remember, `i` just counts up by one each time), so `x+i` will equal `1+2=3`. Continuing in this way, the outputs will be 1, 3, 6, 10, 15, 21, 28, 36, 45, and 55. Try it!

What if we didn't start with the line `x = 0`? This can cause many problems. If `x` has not been defined yet, the right-hand side of line 3 would not make any sense (since there would be no `x` to add to `i`), and Matlab will raise an error. Even worse would be if no error arises, because `x` was somehow defined somewhere else, and now Matlab will use whatever value `x` happens to be, likely yielding a wrong result but no error. This kind of bug can be a hard one to catch!

9.7.1 Nested loops

A loop within a loop is called a nested loop. These arise very often. Here is an example which defines a certain matrix.

```
1  for i = 1:3
2      for j = 1:3
3          A(i,j) = i + j;
4      end
5  end
6  display(A);
```

The "`display(A);`" line is a more formal way to output the matrix, but we could have just written "`A`" without a semi-colon to output it as well.

9.7.2 Initialization

The above construction is inefficient, because the matrix keeps growing as the loop runs. First, it is a 1×1 matrix, then when `A(2,1)` is assigned, it becomes a 2×1 matrix, and Matlab does this by *copying* the old 1×1 matrix into a 2×1 matrix, which is an unnecessary step. If the matrix is really big, this can really slow down your program. However, if, we just tell Matlab beforehand what size we want the matrix `A` to be, it doesn't need to grow the matrix, and will run much faster. You can do this by first making `A` be a zero-matrix of the correct size, like this:

```
1  A = zeros(3,3);
2  for j = 1:3
3      for i = 1:3
4          A(i,j) = i + j;
5      end
6  end
7  display(A);
```

> **Tip:** In Matlab, the most efficient way to access a matrix is first by the rows, then by the columns (similarly in Fortran). If we switched lines 2 and 3 above, the result would be the same, but slightly slower. In C/C++, the order is reversed.

9.7.3 Tracking loop iterations

Sometimes, it is convenient to index by one set of values, and loop over another. For example, we can plot several polygons like this:

```
1  % A program to plot several polygons.
2  i = 1;  % Initialize a counter to index the plots.
3  for N = 3:10
4      theta = linspace(0,2*pi,N+1);
5      subplot(2,4,i);
6      plot(cos(theta),sin(theta));
7      axis('square');
8      title(sprintf('N = %g',N));
9      i = i + 1; % Update the counter.
10 end
```

Note that the counter `i=1,2,3,...`, but `N=3,4,5,...`. Therefore, we don't always need to use the loop iteration as an index, and sometimes is it convient to keep them separate.

The `sprintf` ("string print format") command is useful for the `%g` command tells Matlab, "put the following number here."

9.8 First basic program: Fibonacci Numbers

Let's compute the first few Fibonacci numbers. The first two are 1 and 1. After that, the next number is always the sum of the previous two, so the third number is 1+1=2, the fourth is 1+2=3, and the fifth is 2+3=5, then 8, 13, 21, 34, and so on. To compute these, we need to store the old values. We can do this in a vector (called "`fib`", and then loop over the first few numbers, like this:

```
1  fib(1) = 1;
2  fib(2) = 1;
3  for i = 3:10
4      fib(i) = fib(i-1) + fib(i-2);
5  end
6  display(fib);
```

This will do the job, but we can make it even better with these fixes:

- The loop is inefficient because we did not pre-declare the vector. We can fix this by putting "`fib = zeros(1,10);`" at the top of the program.
- We should also comment the code to tell the user what it does. We shouldn't comment every line, but just enough so that we can see what is going on.
- Another fix is that we have "hard coded" the number 10 in the program. What if we want more numbers? We would have to dig through the code and change each place where the 10 occurs (after the above fix, it now occurs in two places). If we make a mistake, the code will have a bug. It is easier to give it a name (say, "`N`"), and then use it.

> **Tip:** In general, you should try to **avoid hard-coding** as much as possible. Even if you are only going to use a number twice, give it a variable. Often, you should name something even if you only use it once. This can help make it more clear for the user of the code (which may be your future self!).

The above fixes now give us a better program:

```
1  % A program to compute the first N Fibonacci numbers.
2
3  N = 10; % The number of Fibonacci number to compute.
4
5  fib = zeros(1,N);
6
7  fib(1) = 1;
8  fib(2) = 1;
9  for i = 3:N
10     fib(i) = fib(i-1) + fib(i-2);
11 end
12 display(fib);
```

Finally, let's look at a different approach to this problem. Maybe we only want the N^{th} Fibonacci number, and we don't want to store all the previous values. A little thought let's us see that we can do this by using two auxiliary variables to hold the previous two values, call them "`fibLast1`" and "`fibLast2`".

```
1  % A program to compute the Nth Fibonacci number.
2
3  fibLast1 = 1; % initialize
4  fibLast2 = 1;
5  fib = 1;        % in case N == 1 or N == 2
6  for i = 3:N
7      fib = fibLast1 + fibLast2; % Compute next fib number
8      fibLast2 = fibLast1; % Update fibLast2 to fibLast1
9      fibLast1 = fib;      % Update fibLast2 to fib
10 end
11 display(fib);
```

Question: What would happen in we switched lines 8 and 9? Would the code be correct still? Remember, Matlab process each line in *sequence*; that is, one after the other.

> **Tip:** The overwriting done in the update steps in lines 8 and 9 are very common in programming! You will often see a mathematical statement written with indices, such as $f_i = f_{i-1} + f_{i-2}$. This does *not* mean your code should contain indices! It is often far better to overwrite previous data whenever it is not longer needed, as in the code above. This can save memory, especially when problems become very large.

9.9 Conditionals (if/then statements)

Another major programming construct involves "`if`" statements, which often have an `else` statement with them. Try this example to get the idea:

```
1  x = 3;
2  if x == 3
3     display('I love Matlab!');
4  else
5     display('I love Corn Flakes!');
6  end
```

Try running this, then change the first line to something else, like `x = 5`, and run again.

Note the double equals sign, "`==`" on line 2. This is more like the traditional mathematical equality. It does not set `x` equal to 3, it *checks* if `x` is equal to 3. For example, Matlab will return a value of "true" for `3 == 3`, and a value of "false" for `3 == 5`. (Matlab uses the number 1 for true, and the number 0 for false.) You can also use inequalities. Try these:

```
>> 3 == 3 % Is 3 equal to 3?
>> 3 == 5 % Is 3 equal to 5?
>> 3 < 5  % Is 3 less than to 5?
>> 3 < 3  % Is 3 less than to 3?
>> 3 <= 5 % Is 3 less than or equal to 5?
>> 3 >= 5 % Is 3 greater than or equal to 5?
>> 3 >= 3 % Is 3 greater than or equal to 3?
>> 3 ~= 3 % Is 3 not equal to 3?
>> 3 ~= 5 % Is 3 not equal to 5?
```

9.9.1 Breaks and While Loops

Sometimes, we don't know how long a loop should run. Suppose we want to find the biggest Factorial number less than 200. We could do it like this;

```
% Find the largest factorial less than N.
N = 200;
fac = 1;
for i = 1:N
    fac = fac*i;
    if fac > N
        break;
    end
end
display(sprintf('%g! = %g',i,fac/i));
```

This can also be achieved with a `while` loop, like this:

```
% Find the largest factorial less than N.
N = 200;
fac = 1;
i = 1;
while fac < N
    i = i + 1;
    fac = fac*i;
end
display(sprintf('%g! = %g',i-1,fac/i));
```

Note that `for` loops and `while` loops are similar, but `while` loops have a greater danger of becoming infinite loops that never stop. If this happens, press `[CTRL + C]`.

9.10 Functions

Functions are very useful in Matlab. A function is like a mini-program with an input and an output. Sometimes, you want to do a complicated operation many times. It can often be useful to put this into a function.

9.10.1 Programming the Factorial Function

Let's build the factorial function. To do this, open a new m-file, and call it "`factorial.m`" (In Matlab, the function name has to match the file name.)

```
function value = factorial(N);
% A function to compute the factorial of N.
% N! = 1*2*3*...*(N-1)*N

value = 1;
for i = 1:N
    value = value*i;
end

end % End of the function.
```

Note how the input is `N`, so `N` does not need to be defined (the user will pass it into the function). Also, the output is whatever the final value of "`value`" is.

To call this function, make sure you are in the same directory as the file. Then type in the command window:

```
>> factorial(5)
```

and you should get an answer of 120 (= 5!). This function isn't bad, but we could make it better. What if the user passes in a zero? Recall that $0! = 1$. Also, we can throw an error if the user tries to input a negative number. This can all be handled using conditionals:

```
function value = factorial(N);
% A function to compute the factorial of N.
% N!=1*2*3*...*(N-1)*N

if N < 0
    error('Input must be a postive integer!');
elseif N == 0
    value = 1;
else
    value = 1;
    for i = 1:N
    value = value*i;
    end
end

end % End of function factorial.
```

Note the use of the "`elseif`" keyword.

9.10.2 Two-variable functions: Programming the choose function

Recall the choose function from probability:

$$ {}_nC_m = \binom{n}{m} = \frac{n!}{m!(n-m)!} $$

We can use out factorial function to code this! Both programs need to be saved in the same folder and run from that folder, so that "`choose.m`" can call "`factorial.m`". The choose function also requires two inputs. We can make a choose function like this:

```
function value = choose(n,m);
% Compute "n choose m", i.e, n!/(m!(n-m)!)

if n >= m
  value = factorial(n)/(factorial(m)*factorial(n-m));
else
  value = 0;
end

end % End of function choose.
```

Test this program to compute ${}_7C_4 = 35$.

9.10.3 Multiple Outputs

Multiple outputs are also common:

```
function [v,p] = freeFall(a0,v0,p0,t);
% Compute velocity and position of an object in free fall.

v = a0*t + v0;
p = (a0/2)*t^2 + v0*t + p0;

end % End of function freeFall.
```

You can call this code from the command line or another program (in the same folder as the freeFall function), like this:

```
a0 = 9.81;
v0 = 0.0;
p0 = 0.0;
t = 0.2;
[v,p] = freeFall(a0,v0,p0,t);
display(sprintf('At t = %g, pos = %g, and vel = %g.',t,p,v));
```

9.10.4 Symbolic functions

Sometimes, it is convenient to have a simple calculus-type function defined on the fly. Matlab can do this using "function handles," provided by the @ symbol, like this:

```
>> f = @(x,y) x^2+y^2; % Define f as a function of two variables
>> f(2,3);             % Evaluate f at the point (2,3).
```

Here is an example of a vector-valued function:

```
>> f = @(t,y) [ -0.5.*y(1)+y(2)+1 ; -y(1)-0.5.*y(2)+2 ];
```

In general, using functions like this is fairly slow (function calls are expensive), so it is best to avoid them if possible, especially if it is in a loop. For example, rather than using function handles to take a symbolic derivative in Matlab, instead, compute the derivative by hand (or compute it once using Matlab or Mathematica or Maple or something), but then use the result directly.

9.11 Miscellaneous Tips

1. **Always properly indent your code! Otherwise, it can be very hard to read. To automatically indent it, push: `[CTRL+A]` then `[CTRL+I]`. Do this every few minutes, and every time before you show anybody else your code!**

2. Don't use spaces in file names. Use underscores, such as `my_cool_file.m` or "camel case," such as (`myCoolFile.m`).

3. Name files by date in `YYYYMMDD` format. They will self-organize this way. For example, for a file made on Sept 3, 2015, call it something like `myFile20150903.m`. You could also do something like `my_file_2015_09_03.m` (If you want to get really fancy, you can look up "Unix time" and use it to name files.)

4. Use short-but-descriptive names for variables. If you name everything `a, b, c, aa, bb, cc, x, xx, xy, xx2, xx3`, and so on, it can be very hard to keep track of what you are doing. Better names might be something like: `delta_t`, `y_old`, `y_new`, `max_error`, `norm_error`, and so on. You could also use camelCase here too.

5. Save a copy of your file every time you work on it! The best way to do this is with a "versioning system" like git or svn (Google "git basics" or "svn basics", etc., if you want to learn more), but you should try to keep copies of all your old versions, organized by date.

6. Matlab uses unusual keys for copy/paste. You can use the more standard `[CTRL+C]`, `[CTRL+V]`, etc. by going in the menu to:

   Home $\longrightarrow$ Preferences $\longrightarrow$ Keyboard $\longrightarrow$ Shortcuts

   and changing "Active settings" from "Emacs Default Set" to "Windows Default Set".

7. Time things with `tic` and `toc`:

   ```
   >> tic;
   >> inv(rand(20,20));
   >> toc
   ```

   This finds how long it takes Matlab to compute the inverse of a random 20×20 matrix.

8. **Matlab stuck?** Overheating your computer? Caught in an infinite loop? Just hit:

   `CTRL + C`

   while in the command-line window to kill Matlab's current operation.

10 Exam Samples

In this section, we include exams with solutions as a resource for future instructors of the course.

10.1 Exams

FULL NAME: key (please print) Exam 1

MATH 433, Nonlinear Optimization Dr. Adam Larios No calculators

Answers without full, proper justification will not receive full credit.

1. **(10 points)** Consider the system of constraints given by

$$3x_1 - x_2 + x_3 \geq 3$$
$$x_1 + x_2 + 2x_3 \geq 4$$
$$x_1^2 + x_2^2 + x_3^2 \geq 3$$

Find a value of a such that $\mathbf{x} = [1, a, 1]^T$ is a feasible point. Is the $\mathbf{x}$ that you found an active/binding point?

Need: $3 - a + 1 \geq 3$
$1 + a + 2 \geq 4$
$1 + a^2 + 1 \geq 3$

Choosing $a = 1$ gives a feasible point which is also a binding point.

2. (a) **(10 points)** Find an LU factorization of A, where

$$A = \begin{bmatrix} 2 & -4 & 2 \\ -4 & 5 & 2 \\ 6 & -9 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 3 & -1 & 1 \end{bmatrix} \begin{bmatrix} 2 & -4 & 2 \\ 0 & -3 & 6 \\ 0 & 0 & 1 \end{bmatrix} = LU$$

Start: $A = \begin{bmatrix} 1 & 0 & 0 \\ * & 1 & 0 \\ * & * & 1 \end{bmatrix} \begin{bmatrix} 2 & -4 & 2 \\ 0 & * & * \\ 0 & 0 & * \end{bmatrix}$

Next step: $\sim \begin{bmatrix} 2 & -4 & 2 \\ 0 & -3 & 6 \\ 0 & 0 & 1 \end{bmatrix} \Rightarrow l_{23} = -1$

Next, use Gaussian elim. without pivoting:

$\begin{bmatrix} 2 & -4 & 2 \\ -4 & 5 & 2 \\ 6 & -9 & 1 \end{bmatrix} \sim \begin{bmatrix} 2 & -4 & 2 \\ 0 & -3 & 6 \\ 0 & 3 & -5 \end{bmatrix} \Rightarrow \begin{cases} l_{12} = -2 \\ l_{13} = 3 \end{cases}$

U

(b) **(4 points)** Does the above matrix have a Cholesky factorization? Briefly state why or why not.

A matrix has a Cholesky factorization if and only if it is SPD. Since it is not even symmetric, it does not have a cholesky decomposition

3. **(8 points)** Let A be a SPD (symmetric positive definite) matrix. Show that any eigenvalue λ of A must be positive. (Hint: Do not use determinants.)

If A is SPD, this means $\vec{z}^T A \vec{z} > 0$ for all $\vec{z} \neq \vec{0}$.

Let λ be an eigenvalue of A with eigenvector $\vec{x}$. Since $\vec{x}$ is an eigenvector, $\vec{x} \neq \vec{0}$.

The eigenvalue relationship is $A\vec{x} = \lambda\vec{x}$. Apply $\vec{x}^T$ to both sides:

$\vec{x}^T A \vec{x} = \lambda \vec{x}^T \vec{x} = \lambda \|\vec{x}\|^2$. Thus, $0 < \vec{x}^T A \vec{x} = \lambda \|\vec{x}\|^2$

Since $\|\vec{x}\| > 0$, we must have $\lambda > 0$.

4. **(12 points)** Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex function. Prove that the set $S = \{\mathbf{x} : f(\mathbf{x}) \leq 5\}$ is a convex set.

Let $\vec{x}, \vec{y} \in S$, and let $\alpha \in (0,1)$. We must show $\alpha\vec{x} + (1-\alpha)\vec{y} \in S$. Thus, we look at:

$$f(\alpha\vec{x} + (1-\alpha)\vec{y}) \leq \alpha f(\vec{x}) + (1-\alpha) f(\vec{y}) \leq \alpha(5) + (1-\alpha)(5)$$

(first $\leq$: convexity of f; second $\leq$: since $\vec{x}, \vec{y} \in S$)

$$= 5\alpha + 5 - 5\alpha = 5.$$

So $\alpha\vec{x} + (1-\alpha)\vec{y} \in S$.

5. **(10 points)** Suppose that $f : \mathbb{R} \to \mathbb{R}$ is a smooth function such that $f'''(x) = e^{-x^2}$, and that we approximate $f(6)$ by using the first 3 terms of the Taylor series of $f(x)$ centered at 0 (i.e., we approximate using a quadratic Taylor polynomial). Estimate the error in this approximation.

By Taylor's theorem for the remainder, we know error $= p\frac{f'''(\xi)}{3!}$ with $p = 6$, and for some ξ with $0 < \xi < 6$.

Thus, $$|\text{error}| = \frac{6^3}{3!} e^{-\xi^2} \leq \frac{6^3}{3!} e^{-0^2} = 36.$$

6. **(12 points)** Let $f(x, y) = x^3 - 3xy + y^3$. One critical point is $(0, 0)$. Find another critical point, and determine if it is a maximum, minimum, or neither.

$$\nabla f = \begin{pmatrix} 3x^2 - 3y \\ -3x + 3y^2 \end{pmatrix}. \text{ Set } \nabla f = \vec{0}. \text{ Then } \begin{cases} 3x^2 - 3y = 0 \\ -3x + 3y^2 = 0 \end{cases} \Rightarrow \begin{cases} x^2 = y \\ x = y^2 \end{cases}$$

$\Rightarrow x = x^4 \Rightarrow x = 0$ or $x = 1$. If $x = 0$, then $y = 0^2 = 0$, so this is the $(0,0)$ point.

If $x = 1$, then $y = 1$, so $(1,1)$ is a critical point.

$$\nabla^2 f = \nabla\nabla^T f = H = \begin{pmatrix} 6x & -3 \\ -3 & 6y \end{pmatrix}. \text{ Thus } H(1,1) = \begin{pmatrix} 6 & -3 \\ -3 & 6 \end{pmatrix} \sim \begin{pmatrix} 6 & -3 \\ 0 & (6 - \frac{3}{2}) \end{pmatrix}$$

($\sim$: reduce; second matrix: positive)

So $H(1,1)$ is SPD, so $(1,1)$ is a local min.

7. **(8 points)** Consider the sequence $x_k = e^{-e^k}$. Clearly, $x_k \to 0$. Find the convergence rate.

$$\frac{\|e_{k+1}\|}{\|e_k\|^r} = \frac{e^{-e^{k+1}}}{(e^{-e^k})^r} = e^{-e^{k+1} + re^k} = e^{-e^k(-e+r)}$$

Thus,

$$\lim_{k\to\infty} \frac{\|e_{k+1}\|}{\|e_k\|^r} = \begin{cases} 0 & \text{if } r < e \\ 1 & \text{if } r = e \\ \infty & \text{if } r > e \end{cases}$$

So rate $= e$ with constant 1.

8. **(12 points)** Find the **least-squares solution** of $A\mathbf{x} = \mathbf{b}$ for

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}.$$

Use least-squares method: $A^TAx = A^Tb$

Compute

$$A^TA = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$$

$$A^T\vec{b} = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \end{bmatrix}$$

So

$$\begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \end{bmatrix}$$

or $2x_1 = 4$, $x_2 = 2$

So $\vec{x} = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$

9. **(14 points)** Consider the problem of minimizing the function $f(x, y) = \frac{1}{4}x^4 + y^2$ over $\mathbb{R}^2$. Compute one iteration of Newton's method to find this minimum, with the initial guess $\mathbf{x}_0 = [1, 0]^T$.

Newton's method for minimization:

$$\vec{x}_{k+1} = \vec{x}_k - \underbrace{(H(x_k))^{-1}\nabla f(x_k)}_{p_k}$$

Compute:

$$\nabla f = \begin{pmatrix} x^3 \\ 2y \end{pmatrix} \Rightarrow \nabla f(1,0) = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$H = \nabla\nabla^T f = \begin{pmatrix} 3x^2 & 0 \\ 0 & 2 \end{pmatrix} \Rightarrow H(1,0) = \begin{pmatrix} 3 & 0 \\ 0 & 2 \end{pmatrix}$$

Solve $H\vec{p}_k = -\nabla f$:

$$\begin{pmatrix} 3 & 0 \\ 0 & 2 \end{pmatrix}\begin{pmatrix} p_1 \\ p_2 \end{pmatrix} = -\begin{pmatrix} 1 \\ 0 \end{pmatrix} \Rightarrow \begin{cases} p_1 = -1/3 \\ p_2 = 0 \end{cases} \Rightarrow \vec{p} = \begin{pmatrix} -1/3 \\ 0 \end{pmatrix}$$

Thus,

$$\vec{x}_{k+1} = \vec{x}_k + \vec{p}_k$$

$$= \begin{pmatrix} 1 \\ 0 \end{pmatrix} + \begin{pmatrix} -1/3 \\ 0 \end{pmatrix}$$

$$= \begin{pmatrix} 2/3 \\ 0 \end{pmatrix}.$$

FULL NAME: Key (please print) Exam 2

MATH 433/833, Nonlinear Optimization Dr. Adam Larios No calculators

Answers without full, proper justification will not receive full credit. This means show your work!

1. **(10 points)** Find a null-space matrix for A, where

$$A = \begin{bmatrix} 1 & 2 & 4 & 0 \\ 0 & 1 & 3 & -2 \end{bmatrix} \sim \begin{bmatrix} 1 & 0 & -2 & 4 \\ 0 & 1 & 3 & -2 \end{bmatrix}$$

$$A\vec{x} = \vec{0} \Rightarrow \begin{cases} x_1 = 2x_3 - 4x_4 \\ x_2 = -3x_3 + 2x_4 \\ x_3 = 1x_3 + 0x_4 \\ x_4 = 0x_3 + 1x_4 \end{cases} \text{free variables}$$

$$\Rightarrow \quad \mathcal{Z} = \begin{bmatrix} 2 & -4 \\ -3 & 2 \\ 1 & 0 \\ 0 & 1 \end{bmatrix} \text{ or } \begin{bmatrix} -2 & -4 \\ 1 & 0 \\ 0 & 1 \\ 1/2 & 3/2 \end{bmatrix}$$

2. **(10 points)** Let A be a matrix. Suppose $\mathbf{x} \in \text{null}(A)$, and $\mathbf{y} \in \text{range}(A^T)$. Show that $\mathbf{x}$ is orthogonal (perpendicular) to $\mathbf{y}$ (that is, show $\mathbf{x} \perp \mathbf{y}$).

Since $x \in \text{null}(A)$, $Ax = 0$. Also, since $y \in \text{range}(A^T)$, there exists a v such that $A^T v = y$.

Thus,

$$x^T y = x^T(A^T v) = (x^T A^T v) = (Ax)^T v = 0^T v = 0.$$

3. **(10 points)** Show that vectors below, $\mathbf{u}$ and $\mathbf{v}$, are A-orthogonal, where A is the following SPD (symmetric positive definite) matrix:

$$A = \begin{bmatrix} 2 & -1 \\ -1 & 3 \end{bmatrix}, \quad \mathbf{u} = \begin{bmatrix} -3 \\ 4 \end{bmatrix}, \quad \mathbf{v} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}.$$

$A = A^T$

$$(u,v)_A = (Au, v) = (Au)^T v = u^T A^T v = u^T A v$$

$$= (-3,4)\begin{pmatrix} 2 & -1 \\ -1 & 3 \end{pmatrix}\begin{pmatrix} 3 \\ 2 \end{pmatrix} = (-3,4)\begin{pmatrix} 4 \\ 5 \end{pmatrix}$$

$$= (-3)(4) + (4)(3)$$

$$= 12 - 12$$

$$= 0$$

so they are A-orthogonal

4. **(12 points)** Find $\mathbf{p}$ and $\mathbf{q}$ such that $\mathbf{x} = \mathbf{p} + \mathbf{q}$, and $\mathbf{p} \in \text{null}(A)$, and $\mathbf{q} \in \text{range}(A^T)$, where

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & -1 & 1 \end{bmatrix}, \quad \text{and} \quad \mathbf{x} = \begin{bmatrix} 4 \\ 3 \\ 4 \\ 0 \end{bmatrix}.$$

$$\sim \begin{bmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{bmatrix}$$

$$\Rightarrow Z = \begin{bmatrix} -1 & 0 \\ 0 & -1 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}$$

Columns are already orthogonal, so no need for Gram-Schmidt.

Let $z_1 = \begin{bmatrix} -1 \\ 0 \\ 0 \\ 1 \end{bmatrix}$, $z_2 = \begin{bmatrix} 0 \\ -1 \\ 1 \\ 0 \end{bmatrix}$.

Project:

$$p = \frac{x \cdot z_1}{\|z_1\|^2} z_1 + \frac{x \cdot z_2}{\|z_2\|^2} z_2 = \frac{-4}{2} \begin{bmatrix} -1 \\ 0 \\ 0 \\ 1 \end{bmatrix} + \frac{1}{2} \begin{bmatrix} 0 \\ -1 \\ 1 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ -1/2 \\ 1/2 \\ -2 \end{bmatrix}$$

Then:

$$q = x - p = \begin{bmatrix} 4 \\ 3 \\ 4 \\ 0 \end{bmatrix} - \begin{bmatrix} 2 \\ -1/2 \\ 1/2 \\ -2 \end{bmatrix} = \begin{bmatrix} 2 \\ 7/2 \\ 7/2 \\ 2 \end{bmatrix}$$

5. Recall the rank-one symmetric Quasi-Newton minimization method given by the updates

$$\begin{aligned} \mathbf{s}_k &= \mathbf{x}_{k+1} - \mathbf{x}_k, \text{(update for } \mathbf{x}_{k+1} \text{ not written here)} \\ \mathbf{y}_k &= \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k), \\ B_{k+1} &= B_k + \frac{(\mathbf{y}_k - B_k \mathbf{s}_k)(\mathbf{y}_k - B_k s_k)^T}{(\mathbf{y}_k - B_k \mathbf{s}_k)^T \mathbf{s}_k}. \end{aligned}$$

(a) **(4 points)** What are the two main advantages of this method over Newton's method?

- No need to compute second derivatives (Hessian) (Also allows for updates of Cholesky factorization directly.)
- Rank-one matrix updates allow for fast multiplication and fast inversion (solution of linear problems).

(b) **(4 points)** What are the two main drawbacks of this method compared to Newton's method?

- Convergence is no longer quadratic.
- The B_k matrices are not positive definite.

(c) **(4 points)** Show that the secant condition $B_{k+1} s_k = y_k$ holds for this update.

$$B_{k+1} s_k = B_k s_k + \frac{(y_k - B_k s_k)(y_k - B_k s_k)^T s_k}{(y_k - B_k s_k)^T s_k}$$

$$= B_k s_k + y_k - B_k s_k$$

$$= y_k$$

6. **(10 points)** Recall that the secant method for minimizing $f(x)$ comes from modifying Newton's method for minimization using the approximation

$$f''(x_k) \approx \frac{f'(x_k) - f'(x_{k-1})}{x_k - x_{k-1}}.$$

Secant method: $x_{k+1} = x_k - \left(\frac{x_k - x_{k-1}}{f'(x_k) - f'(x_{k-1})}\right) f'(x_k)$

Let $f(x) = x^3 - x + 1$, and choose $x_0 = 1$, $x_1 = 3$. Compute x_2 using the secant version of Newton's method.

$f'(x) = 3x^2 - 1$

By secant method,

$$x_2 = x_1 - \frac{x_1 - x_0}{f'(x_1) - f'(x_0)} f'(x_1)$$

$$= 3 - \frac{3-1}{(3\cdot 3^2 - 1) - (3\cdot 1^2 - 1)}(3\cdot 3^2 - 1)$$

$$= 3 - \frac{2}{26-2}(26) = 3 - \frac{26}{12} = 3 - \frac{13}{6}$$

$$= \frac{5}{6}$$

Problems 7 and 8 use the same set-up, given below.

Let A be an SPD (symmetric positive definite) matrix and let $\mathbf{b}$ be a given vector. Consider the problem of minimizing $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{b}^T\mathbf{x}$, which is equivalent to solving $A\mathbf{x} = \mathbf{b}$. Consider an iterative scheme for solving this problem, given by

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \alpha_i \mathbf{p}_i$$

for some search direction $\mathbf{p}_i$, and some search length α_i. Let $\mathbf{x}$ be the exact solution, i.e., $\mathbf{x}$ satisfies $A\mathbf{x} = \mathbf{b}$. We define:

$\mathbf{e}_i = \mathbf{x}_i - \mathbf{x}$ = the error

$\mathbf{r}_i = \mathbf{b} - A\mathbf{x}_i$ = the residual (i.e., the error in the output)

7. **(10 points)** (See the box above.) By considering the function $\varphi(\alpha) = f(\mathbf{x}_i + \alpha\mathbf{p}_i)$, find the optimal value of α for a given $\mathbf{x}_i$ and $\mathbf{p}_i$. As usual, *make sure to show your work.*

Since A is SPD, f is convex. Thus, φ is convex.

Moreover, $f(\vec{x}) \to \infty$ as $\|\vec{x}\| \to \infty$. Thus, φ has a unique local min, which is also a global min.

Thus, we can set $0 = \varphi'(\alpha) = \mathbf{p}_i^T \nabla f(\mathbf{x}_i + \alpha\mathbf{p}_i)$

Since $\nabla f(\mathbf{x}) = A\mathbf{x} - \mathbf{b}$, we find

$$0 = \mathbf{p}_i^T(A(\mathbf{x}_i + \alpha\mathbf{p}_i) - \mathbf{b}) = \mathbf{p}_i^T(A\mathbf{x}_i - \mathbf{b} + \alpha A\mathbf{p}_i) = \mathbf{p}_i^T(-\mathbf{r}_i + \alpha A\mathbf{p}_i)$$

$$\Rightarrow \alpha = \frac{\mathbf{p}_i^T \mathbf{r}_i}{\mathbf{p}_i^T A\mathbf{p}_i}$$

8. **(10 points)** (See the box above.) Find an update formula for the residual $\mathbf{r}_{i+1}$ that does not involve any of the vectors $\mathbf{b}$, $\mathbf{x}_i$, or $\mathbf{x}_{i+1}$.

$$\mathbf{r}_{i+1} = \mathbf{b} - A\mathbf{x}_{i+1} \quad \leftarrow \text{by definition}$$

$$= \mathbf{b} - A(\mathbf{x}_i + \alpha_i \mathbf{p}_i)$$

$$= \mathbf{b} - A\mathbf{x}_i - \alpha_i A\mathbf{p}_i$$

$$= \mathbf{r}_i - \alpha_i A\mathbf{p}_i$$

9. Frank wants to use Conjugate Gradient (CG) algorithm to solve $A\mathbf{x} = \mathbf{b}$, where A is an SPD (symmetric positive definite) matrix with all its eigenvalues lying in the interval $[1, 2]$. He implements CG like this (assume `A`, `b`, and `x0` are defined earlier in the code):

```
x = x0;
r = b - A*x; % residual
p = r;
for iter = 1:10000
    if ( norm(r) == 0 )
        break
    end
    alpha = r'*r/(p'*A*p);
    x = x + alpha*p;
    r_new = r - alpha*A*p;
    beta = (r_new'*r_new)/(r'*r);
    p = r_new + beta*p;
    r = r_new;
end
```

Note: Try both versions with the 10×10 Hilbert matrix

A = hilb(10)

to see a major difference in the iteration count.

(a) **(5 points)** The update formulas are all correct, but there is a major bug in the code. What is the problem? Why is it a problem?

Checking if norm(r) = 0 is not good, since in a computer, round-off error can make a number that "should" be zero into a small number, such as $\approx 10^{-16}$. It is better to use: if (norm(r) < tol) where tol is a small number.

(b) **(5 points)** Suppose Frank fixes the problem in part (a). Can you give Frank a suggestion that will speed up his code significantly?

Precompute A*p and store it so you don't have to compute it twice.
(r'*r can also be precomputed)

10. **(6 points)** (Short-answer questions.) **Please keep answers to one short sentence.**

(a) Given an SPD matrix $A \in \mathbb{R}^{n\times n}$ and a vector $\mathbf{r} \in \mathbb{R}^n$, write down a spanning set for the Krylov subspace $\mathcal{K}_4(A, \mathbf{r})$.

$$\mathcal{K}_4(A, r) = \text{span}\{r, Ar, A^2r, A^3r\}$$

(b) Why did Krylov subspaces arise naturally in our investigations of steepest descent/conjugate gradient methods?

The residuals r_i all lie in $\mathcal{K}_i(A, r_0)$. (Expand out formula in problem #8.)

(c) Why do we use Gram-Schmidt (or a similar method) when dealing with Krylov subspaces?

We want to project into $\mathcal{K}_i(A, r_0)$, so we need an orthogonal (or A-orthogonal) basis.

10.2 Sample student exams

Below we include a few sample student exams and discuss them.

10.2.1 Exam 1

The next exam is essentially perfect.

FULL NAME: Rachel Horzewski (please print) Exam 1

MATH 433, Nonlinear Optimization Dr. Adam Larios No calculators

Answers without full, proper justification will not receive full credit.

1. **(10 points)** Consider the system of constraints given by

$$3x_1 - x_2 + x_3 \geq 3$$
$$x_1 + x_2 + 2x_3 \geq 4$$
$$x_1^2 + x_2^2 + x_3^2 \geq 3$$

Find a value of a such that $\mathbf{x} = [1, a, 1]^T$ is a feasible point. Is the $\mathbf{x}$ that you found an active/binding point?

all constraints satisfied

$3 - a + 1 \geq 3$ $\quad 4 - a \geq 3$ $\quad 1 \geq a$ $\quad a \leq 1$

$1 + a + 2 \geq 4$ $\quad 3 + a \geq 4$ $\quad a \geq 1$

$1 + a^2 + 1 \geq 3$ $\quad a^2 + 2 \geq 3$ $\quad a^2 \geq 1$

So $a = 1$. in that case, each constraint is an equality, so $\vec{x}$ is an active/binding pt

2. (a) **(10 points)** Find an LU factorization of A, where

row reduce w/o pivoting keep track in L

$$A = \begin{bmatrix} 2 & -4 & 2 \\ -4 & 5 & 2 \\ 6 & -9 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 3 & -1 & 1 \end{bmatrix} \begin{bmatrix} 2 & -4 & 2 \\ 0 & -3 & 6 \\ 0 & 0 & 1 \end{bmatrix} = LU$$

(could DC using matrix mult. if time)

$$\begin{bmatrix} 2 & -4 & 2 \\ -4 & 5 & 2 \\ 6 & -9 & 1 \end{bmatrix} \xrightarrow[-3r_1 + r_3 = R_3]{2r_1 + r_2 = R_2} \begin{bmatrix} 2 & -4 & 2 \\ 0 & -3 & 6 \\ 0 & 3 & -5 \end{bmatrix} \xrightarrow{r_2 + r_3 = R_3} \begin{bmatrix} 2 & -4 & 2 \\ 0 & -3 & 6 \\ 0 & 0 & 1 \end{bmatrix}$$

DC: L*U

$$\begin{bmatrix} 2 & -4 & 2 \\ -4 & 5 & 2 \\ 6 & -9 & 1 \end{bmatrix}$$

✓ MATCH

(b) **(4 points)** Does the above matrix have a Cholesky factorization? Briefly state why or why not.

Also, $A^T \neq A$

The above matrix does not have a Cholesky factorization because A is not SPD (symmetric positive definite) since U in LU factorization has positive & negative values on the main diagonal.

3. **(8 points)** Let A be a SPD (symmetric positive definite) matrix. Show that any eigenvalue λ of A must be positive. (Hint: Do not use determinants.)

Let λ be an eigenvalue of A. Then $A\vec{x} = \lambda\vec{x}$.

Multiply both sides by $\vec{x}^T$: $\vec{x}^T A\vec{x} = \vec{x}^T \lambda\vec{x}$

$$\Rightarrow \vec{x}^T A\vec{x} = \lambda \|\vec{x}\|^2$$

Since A is SPD, $\vec{x}^T A\vec{x} > 0$. Thus, $\lambda \|\vec{x}\|^2 > 0$ for all $\vec{x} \neq \vec{0}$.

Since $\|\vec{x}\|^2$ is a positive quantity ($\vec{x} \neq \vec{0}$), λ must also be positive. ∎

SCRATCH WORK FOR #4

Convex fcn: $f(\alpha x + (1-\alpha)y) \le \alpha f(x) + (1-\alpha)f(y)$ $\quad x, y \in S$

Convex set: $x, y \in S \quad \alpha x + (1-\alpha)y \in S$

Sketch $x, y \in S$ convex fcn diff

$S \subset \mathbb{R}^n$

want to show $f(\alpha x + (1-\alpha)y) \le 5$ then $x, y, \ldots \in S$

4. **(12 points)** Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex function. Prove that the set $S = \{\mathbf{x} : f(\mathbf{x}) \le 5\}$ is a convex set.

Let $x, y \in S \subset \mathbb{R}^n$. Then $f(\alpha x + (1-\alpha)y) \le \alpha f(x) + (1-\alpha)f(y)$ since f is a convex function. In addition, $f(x) \le 5$ and $f(y) \le 5$ since $x, y \in S$. Thus, $f(\alpha x + (1-\alpha)y) \le \alpha 5 + (1-\alpha)5 = 5$. By definition of the set S, this means $\alpha x + (1-\alpha)y \in S$. Since for $x, y \in S$, $\alpha x + (1-\alpha)y \in S$, S is a convex set by definition of a convex set.

5. **(10 points)** Suppose that $f : \mathbb{R} \to \mathbb{R}$ is a smooth function such that $f'''(x) = e^{-x^2}$, and that we approximate $f(6)$ by using the first 3 terms of the Taylor series of $f(x)$ centered at 0 (i.e., we approximate using a quadratic Taylor polynomial). Estimate the error in this approximation.

Taylor series $f(x+p) = f(x) + p f'(x) + \frac{1}{2}p^2 f''(x) + \frac{1}{3!}p^3 f'''(\xi)$ for $x_0 < \xi < x_0 + p$

$f(0+6) = f(0) + 6f'(0) + \frac{1}{2}6^2 f''(0) + \frac{1}{3!}6^3 (e^{-\xi^2})$ $\quad 0 < \xi < 6$ in this case

error: $\left| f(0+6) - f(0) - 6f'(0) - \frac{1}{2}(36)f''(0) \right| \le \left| \frac{1}{3!}6^3 e^{-\xi^2} \right|$ $\quad \frac{1}{e^{\xi^2}}$ is biggest when ξ is smallest so let $\xi = 0$

$\le \left| \frac{6 \cdot 6^2}{6} \cdot e^{0^2} \right| = 36$

error ≤ 36

Note: Terrible error but makes sense since using $x_0 = 0$ and $p = 6$ (far away)

6. **(12 points)** Let $f(x, y) = x^3 - 3xy + y^3$. One critical point is $(0, 0)$. Find another critical point, and determine if it is a local maximum, minimum, or neither.

For a critical pt, $\vec{\nabla} f(x, y) = \vec{0}$

$\vec{\nabla} f(x, y) = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} 3x^2 - 3y \\ -3x + 3y^2 \end{bmatrix}$

$\vec{\nabla} f = \vec{0}$ ① $3x^2 - 3y = 0 \Rightarrow x^2 = y$ into ②

② $-3x + 3y^2 = 0 : -3x + 3(x^2)^2 = 0$

$-3x + 3x^4 = 0$

$\div 3 \quad x^4 - x = 0$

$x(x^3 - 1) = 0$

$x = 0 \quad x = 1$

$y = x^2$ ⇓ ⇓

$y = 0 \quad y = 1$

already found — new crit pt

Crit pt $(1, 1)$ is a local minimum

know crit pt at $(1, 1)$. need 2nd deriv to determine min, max, or neither

$H = \nabla^2 f = \begin{bmatrix} 6x & -3 \\ -3 & 6y \end{bmatrix}$

@ $(1, 1)$ $H = \begin{bmatrix} 6 & -3 \\ -3 & 6 \end{bmatrix} \xrightarrow{\frac{1}{2}r_1 + r_2 = r_2} \begin{bmatrix} 6 & -3 \\ 0 & 4.5 \end{bmatrix}$

H is SPD because diag entries > 0 so min.

7. **(8 points)** Consider the sequence $x_k = e^{-e^k}$. Clearly, $x_k \to 0$. Find the convergence rate.

$x_k \to x^*$

Convergence rate r $\quad e_k = x_k - x^* = e^{-e^k} - 0$

$\lim_{k \to \infty} \frac{\|e_{k+1}\|}{\|e_k\|^r} = C \quad 0 < C < \infty$ $\quad e_{k+1} = x_{k+1} - x^* = e^{-e^{(k+1)}} - 0$

$\lim_{k \to \infty} \frac{\|e^{-e^{k+1}}\|}{\|e^{-e^k}\|^r} = \lim_{k \to \infty} \frac{\|e^{-e^{k+1}}\|}{\|e^{-re^k}\|} = \lim_{k \to \infty} \|e^{-e^{k+1} + re^k}\| = \lim_{k \to \infty} \|e^{-e(e^k) + re^k}\|$

$= \lim_{k \to \infty} \|e^{e^k(r-e)}\|$ $\quad \therefore r = e$

convergence rate $r = e$

if $r < e \to 0$

if $r > e \to \infty$

if $r = e, C = 1$

$A\hat{x} = \hat{b}$ $\quad A^T(b-\hat{b}) = A^T(A\overset{b}{x} - A\hat{x}) = A^TA\overset{b}{x} - A^TA\hat{x} = 0$

$A^TA\overset{b}{x} = A^TA\hat{x}$

$\hat{x} = (A^TA)^{-1}(A^Tb)$

can't remember how to get to this step

$A^T(b-\hat{b}) = 0$ because $b-\hat{b}$ orthog to cols of A

8. **(12 points)** Find the **least-squares solution** of $Ax = \mathbf{b}$ for

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}.$$

$A^TA\hat{x} = A^Tb$

$\hat{x} = (A^TA)^{-1}A^Tb$

~~Check formula?~~

$(A^TA)^{-1} = \begin{bmatrix} \frac{1}{2} & 0 \\ 0 & 1 \end{bmatrix}$

$\left[\begin{array}{cc|cc} 2 & 0 & 1 & 0 \\ 0 & 1 & 0 & 1 \end{array}\right]$

$A^Tb = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}_{2\times3}\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}_{3\times1} = \begin{bmatrix} 1+0+3 \\ 0+2+0 \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \end{bmatrix}$

$A^TA = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}_{2\times3}\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}_{3\times2} = \begin{bmatrix} 1+0+1 & 0+0+0 \\ 0+0+0 & 0+1+0 \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$

$\begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} \frac{1}{2} & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ ✓

$\hat{x} = \begin{bmatrix} 1/2 & 0 \\ 0 & 1 \end{bmatrix}_{2\times2}\begin{bmatrix} 4 \\ 2 \end{bmatrix}_{2\times1} = \begin{bmatrix} 2+0 \\ 0+2 \end{bmatrix} = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$

$\hat{x} = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$

9. **(14 points)** Consider the problem of minimizing the function $f(x, y) = \frac{1}{4}x^4 + y^2$ over $\mathbb{R}^2$. Compute one iteration of Newton's method to find this minimum, with the initial guess $\mathbf{x}_0 = [1, 0]^T$.

Newton's method $\quad \vec{x}_{k+1} = \vec{x}_k + (\vec{p}_N)_k$

$H(\vec{p}_N)_k = -\nabla f$ for minimization problems

$\vec{\nabla} f = \begin{bmatrix} \frac{\partial f}{\partial x} \\ \frac{\partial f}{\partial y} \end{bmatrix} = \begin{bmatrix} x^3 \\ 2y \end{bmatrix}$ $\qquad \vec{\nabla} f(\vec{x}_0) = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$

$H = \begin{bmatrix} \frac{\partial^2 f}{\partial x^2} & \frac{\partial^2 f}{\partial x \partial y} \\ \frac{\partial^2 f}{\partial y \partial x} & \frac{\partial^2 f}{\partial y^2} \end{bmatrix} = \begin{bmatrix} 3x^2 & 0 \\ 0 & 2 \end{bmatrix}$ $\qquad H(\vec{x}_0) = \begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix}$

SPD so it is invertible and will find min.

$H^{-1} = \begin{bmatrix} 1/3 & 0 \\ 0 & 1/2 \end{bmatrix}$

$\begin{bmatrix} 3 & 0 \\ 0 & 2 \end{bmatrix}\begin{bmatrix} 1/3 & 0 \\ 0 & 1/2 \end{bmatrix} = \begin{bmatrix} 1+0 & 0+0 \\ 0+0 & 0+1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ ✓

$(\vec{p}_N)_0 = -H^{-1}\vec{\nabla} f = -\begin{bmatrix} 1/3 & 0 \\ 0 & 1/2 \end{bmatrix}_{2\times2}\begin{bmatrix} 1 \\ 0 \end{bmatrix}_{2\times1} = -\begin{bmatrix} 1/3+0 \\ 0+0 \end{bmatrix} = -\begin{bmatrix} 1/3 \\ 0 \end{bmatrix}$

$\vec{x}_1 = \vec{x}_0 + (\vec{p}_N)_0$

$= \begin{bmatrix} 1 \\ 0 \end{bmatrix} + \begin{bmatrix} -1/3 \\ 0 \end{bmatrix} = \begin{bmatrix} 2/3 \\ 0 \end{bmatrix}$

$\vec{x}_1 = \begin{bmatrix} 2/3 \\ 0 \end{bmatrix}$

The following exam was not terrible, but had several problems. The biggest is that at a few points, essentially no mathematics is used, and the student attempts to "explain" what is going on without using math. This typically is not a good approach to mathematical proofs or calculations, and students should be cautioned against it.

FULL NAME: Lucas Snethen (please print) Exam 1

MATH 433, Nonlinear Optimization Dr. Adam Larios No calculators

Answers without full, proper justification will not receive full credit.

1. **(10 points)** Consider the system of constraints given by

$$3x_1 - x_2 + x_3 \geq 3 \qquad 3-a+1 = 4-a \geq 3 \Rightarrow 1 \geq a$$
$$x_1 + x_2 + 2x_3 \geq 4 \qquad 1+a+2 = 3+a \geq 4 \Rightarrow a \geq 1$$
$$x_1^2 + x_2^2 + x_3^2 \geq 3 \qquad 1+a^2+1 = 2+a^2 \geq 3 \Rightarrow a \geq 1$$

Find a value of a such that $\mathbf{x} = [1, a, 1]^T$ is a feasible point. Is the $\mathbf{x}$ that you found an active/binding point?

$a = 1$

it is a binding point

2. (a) **(10 points)** Find an LU factorization of A, where

$$A = \begin{bmatrix} 2 & -4 & 2 \\ -4 & 5 & 2 \\ 6 & -9 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 3 & -1 & 1 \end{bmatrix} \begin{bmatrix} 2 & -4 & 2 \\ 0 & -3 & 6 \\ 0 & 0 & 1 \end{bmatrix} = LU$$

$$\sim \begin{bmatrix} 2 & -4 & 2 \\ 0 & -3 & 6 \\ 0 & 3 & -5 \end{bmatrix} \sim \begin{bmatrix} 2 & -4 & 2 \\ 0 & -3 & 6 \\ 0 & 0 & 1 \end{bmatrix}$$

(b) **(4 points)** Does the above matrix have a Cholesky factorization? Briefly state why or why not.

~~This is not~~ no since you can't take the sqrt of $D = \begin{bmatrix} 2 & 0 & 0 \\ 0 & -3 & 0 \\ 0 & 0 & 1 \end{bmatrix}$ since it would have an imaginary term $\sqrt{D} = \begin{bmatrix} \sqrt{2} & 0 & 0 \\ 0 & i\sqrt{3} & 0 \\ 0 & 0 & 1 \end{bmatrix}$

Also, $A^T \neq A$

3. **(8 points)** Let A be a SPD (symmetric positive definite) matrix. Show that any eigenvalue λ of A must be positive. (Hint: Do not use determinants.)

Note: positives on diagonal does not imply positive definite.

since A is SPD then we know the diagonal of A is going to be positive then we know that when we solve for the eigenvalues $(A-\lambda)\mathbf{z} = \mathbf{b}$ then they must be positive

4. **(12 points)** Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex function. Prove that the set $S = \{\mathbf{x} : f(\mathbf{x}) \leq 5\}$ is a convex set.

for S to be convex $\alpha x + (1-\alpha)y \in S$ for $0 \leq \alpha \leq 1$ or $f(\alpha x + (1-\alpha)y) \in S$

$\leq \alpha f(x) + (1-\alpha) f(y)$

but since f is convex we know

and since both $f(x)$ & $f(y) \in S$ and $0 \leq \alpha \leq 1$ then neither of the scalars will make the terms larger. $\therefore S$ is a convex set

Use math to prove it, not english to describe it....

5. **(10 points)** Suppose that $f : \mathbb{R} \to \mathbb{R}$ is a smooth function such that $f'''(x) = e^{-x^2}$, and that we approximate $f(6)$ by using the first 3 terms of the Taylor series of $f(x)$ centered at 0 (i.e., we approximate using a quadratic Taylor polynomial). Estimate the error in this approximation.

the error would be $\frac{p^3 e^{\wedge}(-\xi^2)}{3!}$ where

~~$6 \leq \xi \leq 6+p$~~ where p is the step size from the taylor series

Can you bound it by a number?

$0 \leq \xi \leq 6$

6. **(12 points)** Let $f(x, y) = x^3 - 3xy + y^3$. One critical point is $(0, 0)$. Find another critical point, and determine if it is a maximum, minimum, or neither.

$\nabla f(x,y) = \begin{pmatrix} 3x^2 - 3y \\ -3x + 3y^2 \end{pmatrix}$ ① ② ① $3x^2 - 3y = 0 \Rightarrow y = \frac{3x^2}{3} = x^2$

plug into ② $-3x + 3x^4 = 0$

$x(x^3 - 1) = 0$

$\Rightarrow x^3 = 1$

$x = 1 \Rightarrow 3(1)^2 - 3y = 0$

$3 = 3y$

$\Rightarrow y = 1$

$H = \nabla\nabla f(x,y) = \begin{pmatrix} 6x & -3 \\ -3 & 6y \end{pmatrix}$

plug in $(1,1) \Rightarrow \begin{pmatrix} 6 & -3 \\ -3 & 6 \end{pmatrix}$ by sylvesters criteria this is → show it.

+det so $(1,1)$ is a minimum

7. **(8 points)** Consider the sequence $x_k = e^{-e^k}$. Clearly, $x_k \to 0$. Find the convergence rate.

$e_k = x_k - x_* = e^{-e^k} - 0$

$e_{k+1} = e^{-e^{k+1}} - 0$

$\Rightarrow \lim_{k \to \infty} \frac{\| e^{\wedge}(-e^{k+1}) \|}{\| e^{\wedge}(-e^k) \|^r} = C$

$= \lim_{k \to \infty} e^{\wedge}[r(-e^{k+1} + e^k)] = C$ ($-e^{k+1} + e^k \to 0$)

$\Rightarrow e^{\wedge}(r(-\infty)) = C$

$\Rightarrow r = 0$ so ~~converges~~ sublinear

8. **(12 points)** Find the **least-squares solution** of $A\mathbf{x} = \mathbf{b}$ for

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}.$$

least squares is $A^TAx = A^Tb$

$A^T = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}$

$A^TA = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}\begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}$ $\qquad A^Tb = \begin{bmatrix} 1 & 0 & 1 \\ 0 & 1 & 0 \end{bmatrix}\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \end{bmatrix}$

$\Rightarrow \quad \begin{bmatrix} 2 & 0 \\ 0 & 1 \end{bmatrix}\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 4 \\ 2 \end{bmatrix} \Rightarrow \begin{matrix} x_1 = 2 \\ x_2 = 2 \end{matrix} \Rightarrow x = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$

9. **(14 points)** Consider the problem of minimizing the function $f(x, y) = \frac{1}{4}x^4 + y^2$ over $\mathbb{R}^2$. Compute one iteration of Newton's method to find this minimum, with the initial guess $\mathbf{x}_0 = [1, 0]^T$.

$\nabla f(x,y) = \begin{pmatrix} x^3 \\ 2y \end{pmatrix}$ $\qquad H = \nabla\nabla f(x,y) = \begin{pmatrix} 3x^2 & 0 \\ 0 & 2 \end{pmatrix}$

$H^{-1}\begin{bmatrix} 3x^2 & 0 & 1 & 0 \\ 0 & 2 & 0 & 1 \end{bmatrix} \sim \begin{bmatrix} \frac{1}{3x^2} & 0 \\ 0 & \frac{1}{2} \end{bmatrix}$

$NM = x_0 - H^{-1}\nabla f$

$\Rightarrow \begin{bmatrix} 1 \\ 0 \end{bmatrix} - \begin{bmatrix} \frac{1}{3x^2} & 0 \\ 0 & \frac{1}{2} \end{bmatrix}\begin{bmatrix} x^3 \\ 2y \end{bmatrix}$

$\Rightarrow \begin{bmatrix} 1 \\ 0 \end{bmatrix} - \begin{bmatrix} \frac{x}{3} \\ y \end{bmatrix} \Rightarrow \begin{bmatrix} 1 \\ 0 \end{bmatrix} - \begin{bmatrix} \frac{1}{3} \\ 0 \end{bmatrix} = \begin{bmatrix} \frac{2}{3} \\ 0 \end{bmatrix}$

The following exam is an example of the lower-category of exams.

FULL NAME: N. Benes (please print) Exam 1

MATH 433, Nonlinear Optimization Dr. Adam Larios No calculators

Answers without full, proper justification will not receive full credit.

1. **(10 points)** Consider the system of constraints given by

$$3x_1 - x_2 + x_3 \geq 3$$
$$x_1 + x_2 + 2x_3 \geq 4$$
$$x_1^2 + x_2^2 + x_3^2 \geq 3$$

Find a value of a such that $\mathbf{x} = [1, a, 1]^T$ is a feasible point. Is the $\mathbf{x}$ that you found an active/binding point?

$$\begin{bmatrix} 3 & -1 & 1 \\ 1 & 1 & 2 \\ 1 & 1 & 1 \end{bmatrix} \begin{bmatrix} 1 \\ a \\ 1 \end{bmatrix} = \begin{bmatrix} 3 \\ 4 \\ 3 \end{bmatrix}$$

$4 - a \geq 3$
$3 + a \geq 4$
$2 + a \geq 3$

Let $a = 1$. $\mathbf{x}$ is active.

2. (a) **(10 points)** Find an LU factorization of A, where

$$A = \begin{bmatrix} 2 & -4 & 2 \\ -4 & 5 & 2 \\ 6 & -9 & 1 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ -2 & 1 & 0 \\ 3 & -1 & 1 \end{bmatrix} \begin{bmatrix} 2 & -4 & 2 \\ 0 & -3 & 6 \\ 0 & 0 & 1 \end{bmatrix} = LU$$

$$\begin{bmatrix} 2 & -4 & 2 \\ -4 & 5 & 2 \\ 6 & -9 & 1 \end{bmatrix} + \begin{bmatrix} 2 & -4 & 2 \\ 0 & -3 & 6 \\ 0 & 3 & -5 \end{bmatrix} \to \begin{bmatrix} 2 & -4 & 2 \\ 0 & -3 & 6 \\ 0 & 0 & 1 \end{bmatrix}$$

LUL^* LL^T

(b) **(4 points)** Does the above matrix have a Cholesky factorization? Briefly state why or why not.

$\to A$

??

$$\begin{bmatrix} 2 & 0 & 0 \\ 0 & 1 & -1 \\ 0 & 0 & 1 \end{bmatrix}$$

$\det(L) = 2 + 1 + 1 - 1 = 3 > 0$.

We can find L and L^T and U s.t. $LUL^T = A$.

3. **(8 points)** Let A be a SPD (symmetric positive definite) matrix. Show that any eigenvalue λ of A must be positive. (Hint: Do not use determinants.)

$U \approx \begin{bmatrix} 1 & 3 \\ 0 & 4 \end{bmatrix} \begin{bmatrix} 1 \\ 2 \end{bmatrix}$

If A is SPD, then there exists no non-trivial $\vec{x}$ such that $\vec{x}^T A \vec{x} = 0$. In fact, $\vec{x}^T A \vec{x} > 0$ for $\vec{x} \neq \vec{0}$ as well.

Let A^{RE} be the reduced form of A such that it is upper triangular. If A spd, then all elements along the diagonal must be positive. The eigenvalues then, must also all be positive.

why? why?

4. **(12 points)** Let $f : \mathbb{R}^n \to \mathbb{R}$ be a convex function. Prove that the set $S = \{\mathbf{x} : f(\mathbf{x}) \leq 5\}$ is a convex set.

If $f: \mathbb{R}^n \to \mathbb{R}$ then $\forall a \in \mathbb{R}^n \; \exists b \in \mathbb{R}^n$ such that $f(a) = f(b)$, $a \neq b$. ???

It follows then that $\forall r \in \mathbb{R}^n$ such that $f(r) \in S$, $\exists s \in \mathbb{R}^n$ such that $f(s) = f(r)$ with $s \neq r$.

(Nonlinear transform). Need to use convexity.

5. **(10 points)** Suppose that $f : \mathbb{R} \to \mathbb{R}$ is a smooth function such that $f'''(x) = e^{-x^2}$, and that we approximate $f(6)$ by using the first 3 terms of the Taylor series of $f(x)$ centered at 0 (i.e., we approximate using a quadratic Taylor polynomial). Estimate the error in this approximation.

$2e^{1/3}$ work?

6. **(12 points)** Let $f(x, y) = x^3 - 3xy + y^3$. One critical point is $(0, 0)$. Find another critical point, and determine if it is a maximum, minimum, or neither.

critical point @ (1,1)

$= f_{yx}$ by young's thm

$\partial x \to 3x^2 - 3y$ $\quad \partial x \partial y \to -3$ $\quad \partial y \to 3y^2 - 3x$

$\partial x \partial x \to 6x$ $\quad \partial y \partial x \to -3$ $\quad \partial y \partial y \to 6y$

$$\text{Hessian} = \begin{bmatrix} \partial x \partial x & \partial x \partial y \\ \partial y \partial x & \partial y \partial y \end{bmatrix} = \begin{bmatrix} 6x & -3 \\ -3 & 6y \end{bmatrix}, \quad \text{Hessian}(1,1) = \begin{bmatrix} 6 & -3 \\ -3 & 6 \end{bmatrix}$$

$\operatorname{sgn}(\det(\text{Hessian}(1,1))) = \operatorname{sgn}(6 \cdot 6 - (-3 \cdot -3)) = \operatorname{sgn}(27) = (+)$

Minors are both positive, therefore $(1,1)$ is local min.

7. **(8 points)** Consider the sequence $x_k = e^{-e^k}$. Clearly, $x_k \to 0$. Find the convergence rate.

$x_{k+1} = e^{-e^{k+1}}$

$e^{-k+1} - e^{-k}$

8. **(12 points)** Find the **least-squares solution** of $A\mathbf{x} = \mathbf{b}$ for

$$A = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 1 & 0 \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}.$$

$x_1 = 1$
$x_2 = 2$
$x_3 = 3$

$\text{error} = \sqrt{(1-x_1)^2 + (2-x_2)^2 + (3-x_1)^2}$ ok

$\text{error} = \sqrt{1 - 2x_1 + x_1^2 + 4 - 4x_2 + x_2^2 + 9 - 6x_1 + x_2}$

$\text{error} = \sqrt{2x_1^2 + x_2^2 - 4x_2 - 8x_1 + 14}$

$d_{x_1} = 4x_1 - 8$
$d_{x_2} = 2x_2 - 4$
$d_{x_1 x_1} = 4$
$d_{x_2 x_2} = 2$
$d_{x_1 x_2} = 0$

$\vec{x} = \begin{bmatrix} 2 \\ 2 \end{bmatrix}$ ← where did this come from?

$A^T A \vec{x} = A^T \vec{b}$?

9. **(14 points)** Consider the problem of minimizing the function $f(x, y) = \frac{1}{4}x^4 + y^2$ over $\mathbb{R}^2$. Compute one iteration of Newton's method to find this minimum, with the initial guess $\mathbf{x}_0 = [1, 0]^T$.

Solving $\nabla f = \vec{0}$, not $f = 0$.

Newton-Raphson: $x_{k+1} = x_k - \frac{f(x_k)}{f'(x_k)}$

$H = \begin{bmatrix} 3x^2 & 0 \\ 0 & 2y \end{bmatrix}$ ✓ a ∇f

$\frac{df}{dx} = x^3$
$\frac{df}{dx\,dx} = 3x^2$
$\frac{df}{dy} = 2y$
$\frac{df}{dy\,dy} = 2$
$\frac{d^2 f}{dy\,dx} = \frac{df^2}{dx\,dy} = 0$

young's thm

$f(x_0) = \left[\frac{1}{4}, 0\right]^T$

$x_1 = [0, 0]$

10.2.2 Exam 2

The next exam is close to perfect.

FULL NAME: JIANAN LIU (please print)

MATH 433/833, Nonlinear Optimization — Dr. Adam Larios — Exam 2 — No calculators

Answers without full, proper justification will not receive full credit. This means show your work!

1. **(10 points)** Find a null-space matrix for A, where

$$A = \begin{bmatrix} 1 & 2 & 4 & 0 \\ 0 & 1 & 3 & -2 \end{bmatrix}.$$

$A\vec{p} = 0$ $\quad \begin{bmatrix} 1 & 2 & 4 & 0 \\ 0 & 1 & 3 & -2 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix} = 0 \Rightarrow$ $\quad \begin{matrix} p_1 + 2p_2 + 4p_3 = 0 \\ p_2 + 3p_3 - 2p_4 = 0 \end{matrix} \Rightarrow$ $\quad \begin{matrix} p_1 = -2p_2 - 4p_3 \\ p_4 = \frac{1}{2}p_2 + \frac{3}{2}p_3 \end{matrix}$

$$\Rightarrow p = \begin{bmatrix} -2p_2 - 4p_3 \\ p_2 \\ p_3 \\ \frac{1}{2}p_2 + \frac{3}{2}p_3 \end{bmatrix} = \begin{bmatrix} -2v_1 - 4v_2 \\ v_1 \\ v_2 \\ \frac{1}{2}v_1 + \frac{3}{2}v_2 \end{bmatrix}, \text{ so } \quad p = \begin{bmatrix} -2 \\ 1 \\ 0 \\ \frac{1}{2} \end{bmatrix} v_1 + \begin{bmatrix} -4 \\ 0 \\ 1 \\ \frac{3}{2} \end{bmatrix} v_2$$

$$\text{Null}(A) = Z = \begin{bmatrix} -2 & -4 \\ 1 & 0 \\ 0 & 1 \\ \frac{1}{2} & \frac{3}{2} \end{bmatrix}$$

2. **(10 points)** Let A be a matrix. Suppose $\mathbf{x} \in \text{null}(A)$, and $\mathbf{y} \in \text{range}(A^T)$. Show that $\mathbf{x}$ is orthogonal (perpendicular) to $\mathbf{y}$ (that is, show $\mathbf{x} \perp \mathbf{y}$).

$x \in \text{Null}(A) \Rightarrow A\vec{x} = 0$ $\quad y \in \text{range}(A^T) \Rightarrow y = A^T\lambda$, for some λ

$$\vec{x}^T y = \vec{x}^T A^T \lambda = (A\vec{x})^T \lambda = 0$$

Thus, $\vec{x}$ is orthogonal to $\vec{y}$.

3. **(10 points)** Show that vectors below, $\mathbf{u}$ and $\mathbf{v}$, are A-orthogonal, where A is the following SPD (symmetric positive definite) matrix:

$$A = \begin{bmatrix} 2 & -1 \\ -1 & 3 \end{bmatrix}, \qquad \mathbf{u} = \begin{bmatrix} -3 \\ 4 \end{bmatrix}, \qquad \mathbf{v} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}.$$

Test $\quad u^T A v = \begin{bmatrix} -3 & 4 \end{bmatrix} \begin{bmatrix} 2 & -1 \\ -1 & 3 \end{bmatrix} \begin{bmatrix} 3 \\ 2 \end{bmatrix}$

$$= \begin{bmatrix} -10 & 15 \end{bmatrix} \begin{bmatrix} 3 \\ 2 \end{bmatrix} = -30 + 30 = 0.$$

So $\vec{u}$ and $\vec{v}$ are A-orthogonal

4. **(12 points)** Find $\mathbf{p}$ and $\mathbf{q}$ such that $\mathbf{x} = \mathbf{p} + \mathbf{q}$, and $\mathbf{p} \in \text{null}(A)$, and $\mathbf{q} \in \text{range}(A^T)$, where

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & -1 & 1 \end{bmatrix}, \quad \text{and} \quad \mathbf{x} = \begin{bmatrix} 4 \\ 3 \\ 4 \\ 0 \end{bmatrix}.$$

$A\vec{p}=0 \Rightarrow \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & -1 & 1 \end{bmatrix}\begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix} = 0 \Rightarrow \begin{matrix} p_1+p_2+p_3+p_4=0 \\ p_1-p_2-p_3+p_4=0 \end{matrix} \Rightarrow \begin{matrix} p_1+p_4=0 \\ p_2+p_3=0 \end{matrix}$

So $\vec{p} = \begin{bmatrix} p_1 \\ p_2 \\ -p_2 \\ -p_1 \end{bmatrix} = \begin{bmatrix} v_1 \\ v_2 \\ -v_2 \\ -v_1 \end{bmatrix} \Rightarrow p = \begin{bmatrix} 1 \\ 0 \\ 0 \\ -1 \end{bmatrix} v_1 + \begin{bmatrix} 0 \\ 1 \\ -1 \\ 0 \end{bmatrix} v_2$, so $\text{null}(A) = Z = \begin{bmatrix} 1 & 0 \\ 0 & 1 \\ 0 & -1 \\ -1 & 0 \end{bmatrix}$

$p = \frac{z_1^T x}{\|z_1\|^2} z_1 + \frac{z_2 x}{\|z_2\|^2} z_2 = \frac{4}{2}\cdot\begin{bmatrix} 1 \\ 0 \\ 0 \\ -1 \end{bmatrix} + \frac{-1}{2}\begin{bmatrix} 0 \\ 1 \\ -1 \\ 0 \end{bmatrix} = \begin{bmatrix} 2 \\ -0.5 \\ 0.5 \\ -2 \end{bmatrix}$

$q = x - p = \begin{bmatrix} 4 \\ 3 \\ 4 \\ 0 \end{bmatrix} - \begin{bmatrix} 2 \\ -0.5 \\ 0.5 \\ -2 \end{bmatrix} = \begin{bmatrix} 2 \\ 3.5 \\ 3.5 \\ 2 \end{bmatrix}$

5. Recall the rank-one symmetric Quasi-Newton minimization method given by the updates

$$\begin{aligned} \mathbf{s}_k &= \mathbf{x}_{k+1} - \mathbf{x}_k, \text{(update for } \mathbf{x}_{k+1} \text{ not written here)} \\ \mathbf{y}_k &= \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k), \\ B_{k+1} &= B_k + \frac{(\mathbf{y}_k - B_k\mathbf{s}_k)(\mathbf{y}_k - B_k s_k)^T}{(\mathbf{y}_k - B_k\mathbf{s}_k)^T\mathbf{s}_k}. \end{aligned}$$

(a) **(4 points)** What are the two main advantages of this method over Newton's method?

① does not need the second derivtives.

② ~~less memory~~ ~~shorter time~~ faster each iteration step, just vectors multiplication, lower cost per iteration.

(b) **(4 points)** What are the two main drawbacks of this method compared to Newton's method?

① ~~B_{k+1} can not guarantee guarantee step~~ low rate convergence, more iterations

② ~~more iterations steps~~. can not guarantee B_{k+1} is SPD. because rank 1 vectors.

(c) **(4 points)** Show that the secant condition $B_{k+1}s_k = y_k$ holds for this update.

$$B_{k+1}s_k = \left(B_k + \frac{(y_k - B_k s_k)(y_k - B_k s_k)^T}{(y_k - B_k s_k)^T s_k}\right) s_k$$

$$= B_k s_k + \frac{(y_k - B_k s_k)\left[(y_k - B_k s_k)^T s_k\right]}{(y_k - B_k s_k)^T s_k} \quad \text{scaler.}$$

$$= B_k s_k + y_k - B_k s_k$$

$$= y_k$$

6. **(10 points)** Recall that the secant method for minimizing $f(x)$ comes from modifying Newton's method for minimization using the approximation

$$f''(x_k) \approx \frac{f'(x_k) - f'(x_{k-1})}{x_k - x_{k-1}}.$$

Let $f(x) = x^3 - x + 1$, and choose $x_0 = 1$, $x_1 = 3$. Compute x_2 using the secant version of Newton's method.

Secant NM: $X_{k+1} = X_k - \frac{X_k - X_{k-1}}{f'(X_k) - f'(X_{k-1})} \cdot f'(X_k)$, $f(X) = X^3 - X + 1 \Rightarrow f'(X) = 3X^2 - 1 \Rightarrow f'(X_0) = 2$ $f'(X_1) = 26$

$f''(X) = 6X$ $f''(X_0) = 6$, $f''(X_1) = 18$

$X_2 = X_1 - \frac{X_1 - X_0}{f'(X_1) - f'(X_0)} \cdot f'(X_1) = 3 - \frac{3-1}{26-2} \cdot 26 = 3 - \frac{26}{12} = 3 - \frac{13}{6} = \frac{5}{6}$

Problems 7 and 8 use the same set-up, given below.

Let A be an SPD (symmetric positive definite) matrix and let $\mathbf{b}$ be a given vector. Consider the problem of minimizing $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{b}^T\mathbf{x}$, which is equivalent to solving $A\mathbf{x} = \mathbf{b}$. Consider an iterative scheme for solving this problem, given by

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \alpha_i \mathbf{p}_i$$

for some search direction $\mathbf{p}_i$, and some search length α_i. Let $\mathbf{x}$ be the exact solution, i.e., $\mathbf{x}$ satisfies $A\mathbf{x} = \mathbf{b}$. We define:

$\mathbf{e}_i = \mathbf{x}_i - \mathbf{x}$ = the error

$\mathbf{r}_i = \mathbf{b} - A\mathbf{x}_i$ = the residual (i.e., the error in the output)

7. **(10 points)** (See the box above.) By considering the function $\varphi(\alpha) = f(\mathbf{x}_i + \alpha\mathbf{p}_i)$, find the optimal value of α for a given $\mathbf{x}_i$ and $\mathbf{p}_i$. As usual, *make sure to show your work.*

$\min \varphi(\alpha) = f(x_i + \alpha p_i)$, $\nabla f = Ax - b$,

$\varphi'(\alpha) = p_i^T \nabla f(x_i + \alpha p_i) = p_i^T [A(x_i + \alpha p_i) - b] = p_i^T [Ax_i - b + \alpha A p_i]$

$= p_i^T (-r_i + \alpha A p_i) = 0$.

$\alpha A p_i = p_i^T r_i \Rightarrow \alpha = \frac{p_i^T r_i}{p_i^T A p_i}$

8. **(10 points)** (See the box above.) Find an update formula for the residual $\mathbf{r}_{i+1}$ that does not involve any of the vectors $\mathbf{b}$, $\mathbf{x}_i$, or $\mathbf{x}_{i+1}$.

$x_{i+1} = x_i + \alpha_i p_i$

$r_{i+1} = b - Ax_{i+1} = b - A(x_i + \alpha_i p_i) = b - Ax_i - \alpha A p_i$ (last question answer)

$= r_i - \frac{p_i^T r_i A p_i}{p_i^T A p_i} = r_i - \frac{p_i^T r_i A p_i}{p_i^T A p_i}$

9. Frank wants to use Conjugate Gradient (CG) algorithm to solve $A\mathbf{x} = \mathbf{b}$, where A is an SPD (symmetric positive definite) matrix with all its eigenvalues lying in the interval $[1, 2]$. He implements CG like this (assume A, b, and x0 are defined earlier in the code):

```
x = x0;
r = b - A*x; % residual
p = r;
for iter = 1:10000
    if ( norm(r) == 0 )
        break
    end
    alpha = r'*r/(p'*A*p);
    x = x + alpha*p;
    r_new = r - alpha*A*p;
    beta = (r_new'*r_new)/(r'*r);
    p = r_new + beta*p;
    r = r_new;
end
```

(a) **(5 points)** The update formulas are all correct, but there is a major bug in the code. What is the problem? Why is it a problem?

if (norm(r) == 0) this code is a major bug. Since in computer norm(r) rarely equals to 0, so this code is ineffective. because the computer exist errors, Which is a calculation small number, like etc. 10^{-10}), we can set a tolerance instead.
break
end

(b) **(5 points)** Suppose Frank fixes the problem in part (a). Can you give Frank a suggestion that will speed up his code significantly?

add Ap = A*p after bold line, and sustitute A*p to Ap and.

this will speed up the program.

10. **(6 points)** (Short-answer questions.) **Please keep answers to one short sentence.**

(a) Given an SPD matrix $A \in \mathbb{R}^{n\times n}$ and a vector $\mathbf{r} \in \mathbb{R}^n$, write down a spanning set for the Krylov subspace $\mathcal{K}_4(A, \mathbf{r})$

span $\{r, Ar, A^2r, A^3r\}$

(b) Why did Krylov subspaces arise naturally in our investigations of steepest descent/conjugate gradient methods?

We are solving $Ax=b$ with iteration method $x_{k+1} = x_k + \alpha_k p_k = c_0 r_0 + c_1 Ar + \cdots c_n A^{n-1} r$ which is in $K_n(A, r)$

(c) Why do we use Gram-Schmidt (or a similar method) when dealing with Krylov subspaces?

because we want A-orthogonal basis. $p_j^T A p_i = 0$.

Yes, but we want it so we can project.

The following exam is an example of the lower-category of exams.

FULL NAME: Nirmal kc (please print) Exam 2

MATH 433/833, Nonlinear Optimization Dr. Adam Larios No calculators

Answers without full, proper justification will not receive full credit. This means show your work!

1. **(10 points)** Find a null-space matrix for A, where

$$A = \begin{bmatrix} 1 & 2 & 4 & 0 \\ 0 & 1 & 3 & -2 \end{bmatrix}.$$

since $\text{null}(A) = \{\vec{x} : A\vec{x} = 0\}$

so, let $A\vec{x} = 0$

$$\begin{bmatrix} 1 & 2 & 4 & 0 \\ 0 & 1 & 3 & -2 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ x_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \Rightarrow \begin{aligned} x_1 + 2x_2 + 4x_3 &= 0 \\ \text{and } x_2 + 3x_3 - 2x_4 &= 0 \end{aligned}$$

then,

$$\begin{aligned} x_1 &= -2x_2 - 4x_3 \\ x_2 &= -3x_3 + 2x_4 \\ x_3 &= x_3 \\ x_4 &= x_4 \end{aligned}$$

$Z = [\quad] \quad Z = ?$

2. **(10 points)** Let A be a matrix. Suppose $\mathbf{x} \in \text{null}(A)$, and $\mathbf{y} \in \text{range}(A^T)$. Show that $\mathbf{x}$ is orthogonal (perpendicular) to $\mathbf{y}$ (that is, show $\mathbf{x} \perp \mathbf{y}$).

since $x \in \text{null}(A)$ so, $A\vec{x} = 0$ ✓

and $y \in \text{Range}(A^T)$

$(Ax)(A^T y) = 0$

$\Rightarrow x \cdot y = 0$

$A^T A = A$??

since A is SPD

since $x \cdot y = 0$ so, $x \perp y$

3. **(10 points)** Show that vectors below, $\mathbf{u}$ and $\mathbf{v}$, are A-orthogonal, where A is the following SPD (symmetric positive definite) matrix:

$$A = \begin{bmatrix} 2 & -1 \\ -1 & 3 \end{bmatrix}, \quad \mathbf{u} = \begin{bmatrix} -3 \\ 4 \end{bmatrix}, \quad \mathbf{v} = \begin{bmatrix} 3 \\ 2 \end{bmatrix}.$$

vector

Defn A-orthogonal for two x_i and x_j with $i \neq j$

then $x_i^T A x_j = 0$

$$\begin{bmatrix} -3 & 4 \end{bmatrix} \begin{bmatrix} 2 & -1 \\ -1 & 3 \end{bmatrix} \begin{bmatrix} 3 \\ 2 \end{bmatrix}$$

$$= \begin{bmatrix} -3 & 4 \end{bmatrix} \begin{bmatrix} 6 & -2 \\ -3 & +6 \end{bmatrix}$$

$$= \begin{bmatrix} -3 & 4 \end{bmatrix} \begin{bmatrix} 4 \\ 3 \end{bmatrix}$$

$$= -12 + 12$$

$$= 0$$

proved

4. **(12 points)** Find $\mathbf{p}$ and $\mathbf{q}$ such that $\mathbf{x} = \mathbf{p} + \mathbf{q}$, and $\mathbf{p} \in \text{null}(A)$, and $\mathbf{q} \in \text{range}(A^T)$, where

$$A = \begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & -1 & 1 \end{bmatrix}, \quad \text{and} \quad \mathbf{x} = \begin{bmatrix} 4 \\ 3 \\ 4 \\ 0 \end{bmatrix}.$$

$AP = 0$

$\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & -1 & 1 \end{bmatrix} \begin{bmatrix} p_1 \\ p_2 \\ p_3 \\ p_4 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$

$\Rightarrow p_1 + p_2 + p_3 + p_4 = 0 \quad \cdots (1)$

and $p_1 - p_2 - p_3 + p_4 = 0 \quad \cdots (2)$ Adding (1) and (2), From (2)

$p_1 - p_2 - p_3 - p_1 = 0$

$-p_2 = p_3$

$p_1 + p_4 = 0 \Rightarrow p_1 = -p_4$

one possible null(A) $= \begin{bmatrix} 1 \\ -2 \\ 2 \\ -1 \end{bmatrix}$ ✓

$x = p + q$

$\Rightarrow \begin{bmatrix} 4 \\ 3 \\ 4 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ -2 \\ 2 \\ -1 \end{bmatrix} + q \Rightarrow q =$ project ? $\begin{bmatrix} 3 \\ 5 \\ 2 \\ 1 \end{bmatrix}$ //

5. Recall the rank-one symmetric Quasi-Newton minimization method given by the updates

$$\begin{aligned} \mathbf{s}_k &= \mathbf{x}_{k+1} - \mathbf{x}_k, \text{(update for } \mathbf{x}_{k+1} \text{ not written here)} \\ \mathbf{y}_k &= \nabla f(\mathbf{x}_{k+1}) - \nabla f(\mathbf{x}_k), \\ B_{k+1} &= B_k + \frac{(\mathbf{y}_k - B_k\mathbf{s}_k)(\mathbf{y}_k - B_k s_k)^T}{(\mathbf{y}_k - B_k\mathbf{s}_k)^T\mathbf{s}_k}. \end{aligned}$$

(a) **(4 points)** What are the two main advantages of this method over Newton's method?

(1) Let's matrix multiplication, - so cheaper??

(2) Faster, ← Maybe. ↑ why?

(b) **(4 points)** What are the two main drawbacks of this method compared to Newton's method?

So it is both faster and slower than Newton's method?

does not converge fast ← Not so more precise. ↑ why?

(c) **(4 points)** Show that the secant condition $B_{k+1}s_k = y_k$ holds for this update.

we have, $B_{k+1}s_k = B_k + \frac{(y_k - B_k s_k)(y_k - B_k s_k)^T}{(y_k - B_k s_k)^T s_k} s_k$

$B_{k+1} = \frac{B_k(y_k - B_k s_k)^T s_k + (y_k - B_k s_k)(y_k - B_k s_k)^T}{(y_k - B_k s_k)^T s_k}$

$\Rightarrow B_{k+1}s_k = \frac{(y_k - B_k s_k)^T (B_k s_k + y_k - B_k s_k)}{(y_k - B_k s_k)^T}$ ← you just canceled vectors??

$\therefore B_{k+1}s_k = y_k$

$3 \cdot 1 - 1$

$3 \cdot 9 - 1$
$= 27 - 1 = 26$

6. **(10 points)** Recall that the secant method for minimizing $f(x)$ comes from modifying Newton's method for minimization using the approximation

$$f''(x_k) \approx \frac{f'(x_k) - f'(x_{k-1})}{x_k - x_{k-1}}. \quad \Rightarrow$$

Let $f(x) = x^3 - x + 1$, and choose $x_0 = 1$, $x_1 = 3$. Compute x_2 using the secant version of Newton's method.

we need, $f'(x) = 3x^2 - 1$, $f''(x) = 6x$

using secant method, $x_{k+1} = x_k - \frac{(x_k - x_{k-1}) \cdot f'(x_k)}{f'(x_k) - f'(x_{k-1})}$

Now, $x_2 = x_1 - \frac{(x_1 - x_0) \cdot f'(x_1)}{f'(x_1) - f'(x_0)}$

$\Rightarrow x_2 = 3 - \frac{(3-1) \cdot 26}{26 - 2}$

$= 3 - \frac{2 \times 26}{24 \times 6} = 3 - \frac{13}{6} = \frac{18-13}{6} = \frac{5}{6}$ //

Problems 7 and 8 use the same set-up, given below.

Let A be an SPD (symmetric positive definite) matrix and let $\mathbf{b}$ be a given vector. Consider the problem of minimizing $f(\mathbf{x}) = \frac{1}{2}\mathbf{x}^T A\mathbf{x} - \mathbf{b}^T\mathbf{x}$, which is equivalent to solving $A\mathbf{x} = \mathbf{b}$. Consider an iterative scheme for solving this problem, given by

$$\mathbf{x}_{i+1} = \mathbf{x}_i + \alpha_i \mathbf{p}_i$$

for some search direction $\mathbf{p}_i$, and some search length α_i. Let $\mathbf{x}$ be the exact solution, i.e., $\mathbf{x}$ satisfies $A\mathbf{x} = \mathbf{b}$. We define:

$\mathbf{e}_i = \mathbf{x}_i - \mathbf{x}$ = the error

$\mathbf{r}_i = \mathbf{b} - A\mathbf{x}_i$ = the residual (i.e., the error in the output)

7. **(10 points)** (See the box above.) By considering the function $\varphi(\alpha) = f(\mathbf{x}_i + \alpha\mathbf{p}_i)$, find the optimal value of α for a given $\mathbf{x}_i$ and $\mathbf{p}_i$. As usual, *make sure to show your work.*

we know, $\nabla f(x) = Ax - b$, $H = \nabla\nabla^T f(x) = A$

set $\varphi(\alpha)' = 0$ then $0 = p_i^T \nabla f(x_i + \alpha p_i)$

$\Rightarrow 0 = p_i^T (A(x_i + \alpha p_i) - b)$

$= p_i^T (Ax_i + \alpha A p_i - b)$

$= p_i^T (-r_i + \alpha A p_i)$

$0 = -r_i^T p_i + \alpha p_i^T A p_i$

$\Rightarrow \alpha = \frac{r_i p_i^T}{A p_i^T p_i}$

$\Rightarrow \alpha = \boxed{\frac{r_i p_i^T}{p_i^T A p_i}}$

8. **(10 points)** (See the box above.) Find an update formula for the residual $\mathbf{r}_{i+1}$ that does not involve any of the vectors $\mathbf{b}$, $\mathbf{x}_i$, or $\mathbf{x}_{i+1}$.

$r_{i+1} = b - Ax_{i+1}$

$\Rightarrow = b - A(x_i + \alpha_i p_i)$

$= b - Ax_i - \alpha_i A p_i$

$= r_i - \alpha_i A p_i$

since $x_{i+1} = x_i + \alpha_i p_i$ and $r_i = b - Ax_i$

$\therefore \boxed{r_{i+1} = r_i + \alpha_i A p_i}$

9. Frank wants to use Conjugate Gradient (CG) algorithm to solve $A\mathbf{x} = \mathbf{b}$, where A is an SPD (symmetric positive definite) matrix with all its eigenvalues lying in the interval $[1, 2]$. He implements CG like this (assume `A`, `b`, and `x0` are defined earlier in the code):

```
x = x0;
r = b - A*x; % residual
p = r;
for iter = 1:10000
    if ( norm(r) == 0 )
        break
    end
    alpha = r'*r/(p'*A*p);
    x = x + alpha*p;
    r_new = r - alpha*A*p;
    beta = (r_new'*r_new)/(r'*r);
    p = r_new + beta*p;
    r = r_new;
end
```

(a) **(5 points)** The update formulas are all correct, but there is a major bug in the code. What is the problem? Why is it a problem?

(b) **(5 points)** Suppose Frank fixes the problem in part (a). Can you give Frank a suggestion that will speed up his code significantly?

10. **(6 points)** (Short-answer questions.) **Please keep answers to one short sentence.**

(a) Given an SPD matrix $A \in \mathbb{R}^{n\times n}$ and a vector $\mathbf{r} \in \mathbb{R}^n$, write down a spanning set for the Krylov subspace $\mathcal{K}_4(A, \mathbf{r})$ $\text{span}\{r, Ar, A^2r, A^3r\}$ ✓

(b) Why did Krylov subspaces arise naturally in our investigations of steepest descent/conjugate gradient methods?

we have to deal A-inner product matrix and A-norm vectors. Not necessarily . . .

(c) Why do we use Gram-Schmidt (or a similar method) when dealing with Krylov subspaces?

we need to orthonormalize the vector in an inner product. Why? We need to project.

11 Homework Samples

In this section, we look at samples of student homework from one particular assignment; namely homework #4. The purpose is to compare these homeworks, and also to look at different features that can appear in this kind of work.

11.1 Sample student homeworks

In the first example, we see a very short homework (maybe too short)

2.9: First notice that

$$\nabla f(x) = \begin{pmatrix} 4x_1 - 2x_2 + 6x_1^2 + 4x_1^3 \\ 2x_2 - 2x_1 \end{pmatrix}$$

Hence if $\nabla f(x) = 0$, then $x_1 = x_2$ and so $0 = x_1 + 3x_1^2 + 2x_1^3$. It follows that $x_2 = x_1$ may be any of $0, -\frac{1}{2}$ or -1. Next we notice that

$$H_f(x) = \begin{pmatrix} 4 + 12x_1 + 12x_1^2 & -2 \\ -2 & 2 \end{pmatrix}$$

3/3

Hence, we find that

$$H_f(0,0) = \begin{pmatrix} 4 & -2 \\ -2 & 2 \end{pmatrix} \qquad H_f(-\frac{1}{2}, -\frac{1}{2}) = \begin{pmatrix} 1 & -2 \\ -2 & 2 \end{pmatrix} \qquad H_f(-1,-1) = \begin{pmatrix} 4 & -2 \\ -2 & 2 \end{pmatrix}$$

By Sylvester's Criterion, the first and last matrices are positive definite, but the one in the middle is indefinite. Thus, we have that $(0,0)$ and $(-1,-1)$ are local minimum, and $(-\frac{1}{2}, -\frac{1}{2})$ is neither a minimum or maximum. Fixing $x_1 = 1$, and letting $x_2 \to \infty$, we see that f has no global minimum. Also, fixing $x_2 = 0$ and letting $x_1 \to \infty$, we also see it has no global maximum.

→ both global minima too.

3.2: Note that if $f(x) = 5x^5 + 2x^3 - 4x^2 - 3x + 2$, then $f'(x) = 25x^4 + 6x^2 - 8x - 3$ and $f''(x) = 100x^3 + 12x - 8$. Using f' and f'' in the Newton.m file written for homework three, we can find critical points in the designated region $[-2, 2]$. I made a mesh of $[-2, 2]$ of size .05 and tested all the points on the mesh for initial guesses. Two different results came up out fall the test points: $x \simeq 0.6084, -0.2597$. Hence, we had two critical points at these points. The value of 0.6084 corresponds to a local minimum. Indeed, $f(0) > f(0.6084)$ and $f(1) > f(.06084)$. However, it is not global on $[-2.2]$. We have that $f(-2) < f(0.6084)$.

2/2

3.6: For a quadratic function $f(x) = \frac{1}{2}\langle Qx, x\rangle - \langle x, c\rangle$ where Q is positive (and hence symmetric), we have that

$$\nabla f(x) = Qx - c \qquad \text{and} \qquad H_f(x) = Q$$

Now, if we are trying to find the minimum, then we are searching for zeros of $\nabla f(x)$. Therefore, our Newton scheme will involve $\nabla(\nabla f)(x) = H_f(x)$. Given an initial guess x_0, using the above and the fact that $Q = Q^T$, we have that the Newton iterative scheme will yield

2/2

$$x_1 := x_0 - \nabla((\nabla(f))(x_0)^T)^{-1}\nabla f(x_0) = x_0 - (Q^T)^{-1}(Qx_0 - c) = x_0 - Q^{-1}(Qx_0 - c) = Q^{-1}c$$

By same arguments, it follows that,

$$x_2 = x_1 - Q^{-1}(Qx_1 - c) = Q^{-1}c = x_1$$

Thus, the scheme converges after one step.

4

3/3

(a) By performing elementary row operations without pivoting, we find that

$$A = \begin{pmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 4 & -2 & 1 \end{pmatrix} \begin{pmatrix} 1 & 2 & 4 \\ 0 & 1 & -2 \\ 0 & 0 & 2 \end{pmatrix}$$

Now that we have the LU-factorization of A, D is given by the diagonal of U. Namely, $D = \text{diag}(1, 1, 2)$.

In the next example, we see an example of a student who printed their code, and wrote solutions over it. In general, this should probably be discouraged, since the code should stand on its own. Moreover, if changes are made to the code before the homework is submitted (which is likely), it can be hard to change out the printed copy.

3.2)

```
function x = Newton(f, df, x0, tolerance, maxIter)
%This programs solves f(x) = 0 using Newton's method

N = maxIter; %Instantiation of all variables
F = f(x0);
dF = df(x0);
a = 0;
x = x0;

for i = 1:N
    a = x;
    x = x - (F/dF); %Finds the x_(k+1) term
    if abs(a - x) < tolerance; %Checks error and ends the code if the step difference gets sufficiently small
        break
    end
    F = f(x); %Resets the values of f(x) and df(x) for each iteration
    dF = df(x);
end
display(x);
end
```

2/2

$f(x) = 5x^5 + 2x^3 - 4x^2 - 3x + 2$ | tolerance = $1e^{-15}$

$f'(x) = 25x^4 + 6x^2 - 8x - 3$ | max Iter = 10

$f''(x) = 100x^3 + 12x - 8$

To minimize $f(x)$, we want to look for solutions to N.M. for $f'(x) = 0$.

Thus, we let `f = 25*x.^4 + 6*x.^2 - 8*x -3`

`df = 100*x.^3 + 12*x -8`

Then by choosing x_0 to be $[-2, 2]$ in 0.1 step sizes, i.e. $\{-2.0, -1.9, \ldots, 1.9, 2.0\}$

we find 2 solutions for $f'(x) = 0$

$\hookrightarrow \{x_1 = -0.2899\}$
$\{x_2 = 0.6899\}$

$\Rightarrow f(-0.2899) = 2.474457$
$f(0.6899) = -0.535366$
$f(-2) = -184$
$f(2) = 156$

local minimizers: $\{-2, 0.6899\}$
local maximizers: $\{2, -0.2899\}$
global minimizer: $\{-2\}$
global maximizer: $\{2\}$

on $[-2, 2]$

A common problem in this type of calculation-based work is that students often do not show their work. The issue is that it is not at all clear if the student worked it out in their head, on another piece of paper, or just copied the answer form another student or, e.g., an online resource. Full credit is usually not awarded to this kind of work.

Dongliang Xiao Assignment #4
NUID: 7164138

1.
#2.9. $f(x) = 2x_1^2 + x_2^2 - 2x_1x_2 + 2x_1^3 + x_1^4$

$$\nabla f(x) = \begin{bmatrix} 4x_1 - 2x_2 + 6x_1^2 + 4x_1^3 \\ 2x_2 - 2x_1 \end{bmatrix}$$ ✓

how did you get these. Show work! -0.5

2.5 B/3

$$\nabla f(x) = 0 \Rightarrow \begin{cases} 4x_1 - 2x_2 + 6x_1^2 + 4x_1^3 = 0 \\ 2x_2 - 2x_1 = 0 \end{cases}$$

$x_1 = x_2 = -1$, $x_1 = x_2 = -\frac{1}{2}$, $x_1 = x_2 = 0$

$$\nabla^2 f(x) = \begin{bmatrix} 4 + 12x_1 + 12x_1^2 & -2 \\ -2 & 2 \end{bmatrix}$$ ✓

① for $x_1 = x_2 = -1$

$$\nabla^2 f(x) = \begin{bmatrix} 4 & -2 \\ -2 & 2 \end{bmatrix} \quad \begin{vmatrix} 4 & -2 \\ -2 & 2 \end{vmatrix} = 4 > 0 \Rightarrow \nabla^2 f(x) \text{ is positive definite}$$ ✓

$\therefore x_1 = x_2 = -1$ is a local minimizer

② for $x_1 = x_2 = -\frac{1}{2}$

$$\nabla^2 f(x) = \begin{bmatrix} 1 & -2 \\ -2 & 2 \end{bmatrix} \quad \begin{vmatrix} 1 & -2 \\ -2 & 2 \end{vmatrix} = -2 < 0 \Rightarrow \nabla^2 f(x) \text{ is neither positive or negative definite}$$ ✓

$\therefore x_1 = x_2 = -\frac{1}{2}$ is not a local minimizer

③ for $x_1 = x_2 = 0$

$$\nabla^2 f(x) = \begin{bmatrix} 4 & -2 \\ -2 & 2 \end{bmatrix} \quad \begin{vmatrix} 4 & -2 \\ -2 & 2 \end{vmatrix} = 4 > 0 \Rightarrow \nabla^2 f(x) \text{ is positive definite}$$ ✓

$\therefore x_1 = x_2 = 0$ is a local minimizer

$\because f(x) = 2x_1^2 + x_2^2 - 2x_1x_2 + 2x_1^3 + x_1^4$

$= (x_1 - x_2)^2 + x_1^2(x_1 + 1)^2 \geq 0$

$f(-1, -1) = 2 + 1 - 2 - 2 + 1 = 0$

$f(0, 0) = 0$

$\therefore x_1 = x_2 = -1$ and $x_1 = x_2 = 0$ are both global minimizers ✓

The next example shows one of the lower-scoring homeworks. The bonus problem was not attempted, and although a lot of work is there, it is not check for accuracy, and some of the problems are incomplete.

Math 433

Lucas Snethen HW 4

pg 367 2.9

$f(x) = 2x_1^2 + x_2^2 - 2x_1x_2 + 2x_1^3 + x_1^4$ ✓

2.5|3

$\nabla f(x) = \begin{pmatrix} 4x_1 - 2x_2 + 6x_1^2 + 4x_1^3 \\ 2x_2 - 2x_1 \end{pmatrix} \begin{matrix} (1) \\ (2) \end{matrix}$ (2) $2x_2 - 2x_1 = 0 \Rightarrow x_2 = x_1$

$H(x) = \nabla\nabla f(x) = \begin{pmatrix} 4 + 12x_1 + 12x_1^2 & -2 \\ -2 & 2 \end{pmatrix}$ ✓

plugging into (1)

$4x_1 - 2x_1 + 6x_1^2 + 4x_1^3 = 0$

$4x_1^3 + 6x_1^2 + 2x_1 = 0$

$2x(2x^2 + 3x_1 + 1) = 0$

$2x(2x+1)(x+1) = 0$

$\Rightarrow x_1 = 0, -\frac{1}{2}, -1 = x_2$ ✓

CRT pts $= (0,0), (-\frac{1}{2}, -\frac{1}{2}), (-1,-1)$

plug CRT pts into H $\Rightarrow$

$(0,0) \Rightarrow H_1 = \begin{pmatrix} 4 & -2 \\ -2 & 2 \end{pmatrix}$ ✓

H_1 is SPD by Sylvester's Criterion

so $H_1(0,0)$ is a minimum

$(-\frac{1}{2}, -\frac{1}{2}) \Rightarrow H_2 = \begin{pmatrix} 1 & -2 \\ -2 & 2 \end{pmatrix}$ ✓ ⟶ indefinite

H_2 is not SPD so it is a neither min nor max

$(-1,-1) \Rightarrow H_3 = \begin{pmatrix} 4 & -2 \\ -2 & 2 \end{pmatrix}$ ✓

H_3 is SPD so it is a minimum

-1/2

H_1 is the same as H_3 and since $f(x)$ is a convex function ... we know it is a global min ✓

H_2 is a global ✓ maximum by the same reasoning

11.2 Homework Solutions

Below are the solutions that I wrote up and distributed to the students. They are included as a resource for future instructors of the course.

11.2.1 Homework 0

1. Find the maximum and minimum of $f(x) = \dfrac{x(x-1)}{x^2+3x+3}$ on the interval $[0,6]$ and then on the real line.

   Solution

   First, we compute the derivative:

   $$f'(x) = \frac{4x^2+6x-3}{(x^2+3x+3)^2}$$

   Solving $f'(x) = 0$ means solving $4x + 6x - 3$, which, by means of the quadratic formula, gives $-3/4 \pm \sqrt{21}/4$. As $-3/4 - \sqrt{21}/4 < 0$, for the interval $[0,6]$, we can ignore this critical point. We evaluate f at 0, $-3/4 + \sqrt{21}/4$, and 6, giving 0, $-.0551$, and $10/19$. So the maximum of f on $[0,6]$ is $10/19$ at 6 and the minimum is about -0.0551 at $-3/4 + \sqrt{21}/4$.

   On the whole real line, we compute $f(-3/4 - \sqrt{21}/4) = 6.0551$. Notice that

   $$\lim_{x \to \pm\infty} f(x) = 1$$

   and since f attains values smaller and larger than 1 at its critical points, the values at these critical points are the extrema of the function. Thus, the maximum of f on $\mathbf{R}$ is 6.0551 at $-3/4 - \sqrt{21}/4$ and the minimum of f on $\mathbf{R}$ is $-.0551$ at $-3/4 + \sqrt{21}/4$.

2. Compute the gradient $\nabla f(\mathbf{x})$ of the function $f(x_1, x_2, x_3) = 3x_1^2 + 4x_1x_2 + x_2\sin(x_3)$.

   Solution

   We compute

   $$\begin{aligned}\nabla f(\mathbf{x}) &= \left(\frac{\partial f}{\partial x_1}(\mathbf{x}), \frac{\partial f}{\partial x_2}(\mathbf{x}), \frac{\partial f}{\partial x_3}(\mathbf{x})\right) \\ &= \left(6x_1 + 4x_2, 4x_1 + \sin(x_3), x_2\cos(x_3)\right).\end{aligned}$$

3. Solve the following linear system of equations. If the system is consistent, write the solution in vector form.

   $$\begin{cases} x & -y & +z & = & 0 \\ -x & +3y & +z & = & 5 \\ 3x & +y & +7z & = & 10 \end{cases}$$

   Solution

   We first compute the reduced row echelon form of the augmented matrix A :

   $$A = \begin{bmatrix} 1 & -1 & 1 & 0 \\ -1 & 3 & 1 & 5 \\ 3 & 1 & 7 & 10 \end{bmatrix}$$

   Since rank(A)=2¡# of variables , the system is consistent, and the solution has 1 free variable and hence an infinite number of solutions. From the RREF of A , we have that

   $$\begin{cases} x & & +2z & = & \frac{5}{2} \\ & y & +z & = & \frac{5}{2} \\ & & 0 & = & 0 \end{cases}$$

Hence,

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = \begin{bmatrix} -2z + \frac{5}{2} \\ -z + \frac{5}{2} \\ z \end{bmatrix} = \begin{bmatrix} -2 \\ -1 \\ 1 \end{bmatrix} z + \begin{bmatrix} \frac{5}{2} \\ \frac{5}{2} \\ 0 \end{bmatrix}.$$

4. For each matrix below, determine if it is invertible. If so, compute its inverse.

$$A = \begin{bmatrix} 1 & -1 & 0 \\ 1 & 3 & 2 \end{bmatrix}, \qquad B = \begin{bmatrix} 1 & 3 \\ -1 & -1 \end{bmatrix}, \qquad C = \begin{bmatrix} 2 & 3 & 1 \\ 2 & 1 & 3 \\ 1 & 2 & 0 \end{bmatrix}.$$

Solution

A is not invertible since it is not square.

We find the reduced row echelon form of

$$[B \mid I_{2\times 2}] : \left[\begin{array}{cc|cc} 1 & 3 & 1 & 0 \\ -1 & -1 & 0 & 1 \end{array}\right]$$

Since the reduced row echelon form of B is I , B is invertible and

$$B^{-1} = \begin{bmatrix} -\frac{1}{2} & -\frac{3}{2} \\ \frac{1}{2} & \frac{1}{2} \end{bmatrix}.$$

We find the reduced row echelon form of C :

$$\begin{bmatrix} 2 & 3 & 1 \\ 2 & 1 & 3 \\ 1 & 2 & 0 \end{bmatrix}.$$

Since the reduced row echelon form of C is not the identity $I_{3\times 3}$, C is not invertible.

5. Let

$$A = \begin{bmatrix} 1 & 3 \\ -2 & 6 \end{bmatrix}.$$

Find all of the eigenvalues of A and bases for the corresponding eigenspaces.

Solution

The characteristic polynomial of A is

$$\det \begin{bmatrix} 1-\lambda & 3 \\ -2 & 6-\lambda \end{bmatrix} = (1-\lambda)(6-\lambda) + 6 = \lambda^2 - 7\lambda + 12 = (\lambda - 3)(\lambda - 4).$$

The solutions of $\det(A - \lambda I) = 0$ are the eigenvalues of A. Hence, the eigenvalues are $\lambda_1 = 3$ and $\lambda_2 = 4$.

To find bases for the corresponding eigenspaces, we find a basis for the nullspace of $\det(A - \lambda I)$ for each eigenvalue λ. For $\lambda_1 = 3$,

$$A - 3I = \begin{bmatrix} -2 & 3 \\ -2 & 3 \end{bmatrix} \xrightarrow{\text{RREF}} \begin{bmatrix} 1 & -3/2 \\ 0 & 0 \end{bmatrix}, \quad \text{so} \quad \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 3/2 \\ 1 \end{bmatrix} x_2$$

and so

$$E_3 = \text{span}\left\{ \begin{bmatrix} 3/2 \\ 1 \end{bmatrix} \right\}.$$

For $\lambda_2 = 4$,

$$A - 4I = \begin{bmatrix} -3 & 3 \\ -2 & 2 \end{bmatrix} \xrightarrow{\text{RREF}} \begin{bmatrix} 1 & -1 \\ 0 & 0 \end{bmatrix}, \quad \text{so} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ 1 \end{bmatrix} x_2,$$

and so

$$E_4 = \text{span}\left\{ \begin{bmatrix} 1 \\ 1 \end{bmatrix} \right\}.$$

6. Are the following vectors linearly independent? Prove your answer.

$$\left\{ \begin{bmatrix} 1 \\ 2 \\ 3 \\ 7 \end{bmatrix}, \begin{bmatrix} -1 \\ 3 \\ 2 \\ 2 \end{bmatrix}, \begin{bmatrix} 1 \\ 1 \\ 1 \\ 1 \end{bmatrix} \right\}.$$

Solution

Calling the three vectors $\mathbf{v}^{(1)}, \mathbf{v}^{(2)}, \mathbf{v}^{(3)}$, we wish to know whether scalars a_1, a_2, a_3 exist, not all zero, such that $a_1\mathbf{v}^{(1)} + a_2\mathbf{v}^{(2)} + a_3\mathbf{v}^{(3)} = \mathbf{0}$. If such scalars exist, there exists a linear dependence among the vectors and they are linearly dependent. If no such scalars exist, then the vectors are linearly independent.

We thus wish to determine the number of solutions to the linear system $a_1\mathbf{v}^{(1)} + a_2\mathbf{v}^{(2)} + a_3\mathbf{v}^{(3)} = \mathbf{0}$. Putting the three vectors as columns into a matrix A and computing the RREF, we see that

$$\begin{bmatrix} 1 & -1 & 1 & 0 \\ 2 & 3 & 1 & 0 \\ 3 & 2 & 1 & 0 \\ 7 & 2 & 1 & 0 \end{bmatrix} \xrightarrow{\text{RREF}} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

and hence the only solution is $a_1 = a_2 = a_3 = 0$. Thus, the vectors are linearly independent.

11.2.2 Homework 1

1. Finish typing the following Matlab programs from the Matlab Intro on Blackboard: `factorial.m`, `fibonacci.m`, `choose.m`. Print out your three `*.m` files and attach them to this homework. (Don't copy/paste into Word or anything, just print from Matlab.)

   Solution.

   See the Matlab Intro file on Blackboard.

2. Read sections 1,1, 1.5, 1.6, and 1.7 in the book.

3. Do problem 7.1 (page 22) and problem 7.1 (page 27). (Short answers please.) **Solution to problem 7.1 (page 22)**

   Answers may vary slightly in formulation. Here is one possible formulation. Let c_{ij} be the cost of an aircraft of type i on arc j. Let x_{ij} be the number of aircraft of type i on arc j, and let X be the matrix with entries x_{ij}. Let Q be the arc-node index matrix. Let M_i be the total number of available aircraft

of type i.

$$\begin{aligned} \min \quad & \sum_j \sum_i c_{ij} x_{ij} \\ \text{subject to:} \quad & \sum_i x_{ij} = 1 \\ & XQ = 0 \\ & \sum_j x_{ij} \leq M_i \\ & x_{ij} = 0 \text{ or } 1 \end{aligned}$$

Solution to problem 7.1 (page 27)

Answers may vary slightly in formulation. Here is one possible formulation. (See the book for the notation.)

$$\begin{aligned} \max \quad & \mathbf{r}^T \mathbf{x} - \alpha \mathbf{x}^T V \mathbf{x} + r_0 x_0 \\ \text{subject to:} \quad & x_0 + \sum_{i \neq 0} \mathbf{x}_i = 1 \\ & x_i \geq 0 \end{aligned}$$

Where r_0 is the rate associated with asset 0, and x_0 is the proportion of the investment to be in vested in assent 0.

4. A factory has 9 lathes and 4 grinders. Each machine runs for 40 hours per week. The machines are used to make three different products. Each unit of product 1 requires 2 hours of time on a grinder machine, each unit of product 2 requires 4 hours on a lathe, and each unit of product 3 requires 5 units on a lathe and 3 units on a grinder. The products also have monetary costs of 25, 10, and 15, respectively. The sale price per unit depends on the supply, and is given by $p_1(x_1) = 20 + 50x_1^{-1/2}$, $p_2(x_2) = 15 + 40x_2^{-1/4}$, and $p_3(x_3) = 35 + 100x_3^{-1/3}$, where x_j is the number of units of product j per week.

   (a) What key properties do the functions p_j have? How realistic are these?

   Solution.

   The easiest way to see this is to plot the functions. First, notice that $\lim_{x \to 0^+} p_j(x) = \infty$, and selling no items shouldn't lead to infinite profit, but then, we can just modify the functions to set $p_j(0) = 0$, and note that they should only be defined on the non-negative integers. Next, note that as you make more products, the cost decreases. On the other hand, $\lim_{x \to 0^+} p_1(x) = 20$, so making more products eventually does not affect the individual price much. Similar comments hold for p_2 and p_3.

   (b) The company wants to maximize its weekly profit. Construct the appropriate objective function.

   Solution.

   If x_i items are sold at a price $p_i(x_i)$, then the revenue is $x_i p_i(x_i)$. Denote the costs of each item by c_i. Since Profit = Revenue - Cost, the profit from each type of product is $x_i p_i(x_i) - c_i x_i$. The

total profit is

$$\begin{aligned}&\sum_{i=1}^{3}(x_i p_i(x_i) - c_i x_i)\\ =&(x_1(20+50x_1^{-1/2}) - 25x_1) + (x_2(15+40x_2^{-1/4}) - 10x_2)\\ &+ (x_3(35+100x_3^{-1/3}) - 15x_3)\end{aligned}$$

This is the objective function; that is, the function to be maximized.

(c) Construct appropriate constraints for the amounts of products made. There are a total of four constraints.

Solution.

For the lathe, we must have

$$4x_2 + 5x_3 \leq 9 \cdot 40$$

For the grinder, we must have

$$2x_1 + 3x_3 \leq 4 \cdot 40$$

We also have the following two physical constraints:

$$x_1, x_2, x_3 \geq 0,$$

x_1, x_2, x_3 must be integers.

5. Suppose we have a collection of $n > 3$ data points (x_j, y_j), and we wish to find the quadratic polynomial function $f(x) = a + bx + cx^2$ that minimizes the residual sum of squares $R(a,b,c) = \sum_{j=1}^{n}(y_j - f(x_j))^2$. Let $\mathbf{u}$ be the column vector with components a, b, and c. Use the fact that the minimizer must occur at a point where the first partial derivatives of R are all 0 to construct a matrix $\mathbf{M}$ and a vector $\mathbf{v}$ such that $\mathbf{v}$ is determined by the equation $\mathbf{Mu} = \mathbf{v}$.[1]

Solution. Write

$$R = R(a,b,c) = \sum_{j=1}^{n}(y_j - a - bx_j - cx_j^2)^2$$

Note that R is quadratic in a, b, and c, and its graph will be a convex paraboloid, so in particular, it has a unique minimum. At the minimum, we must have $\nabla R = \mathbf{0}$. Using the chain-rule, we find

$$0 = \frac{\partial R}{\partial a} = \sum_{j=1}^{n} 2(y_j - a - bx_j - cx_j^2)(-1)$$

$$0 = \frac{\partial R}{\partial b} = \sum_{j=1}^{n} 2(y_j - a - bx_j - cx_j^2)(-x_j)$$

$$0 = \frac{\partial R}{\partial c} = \sum_{j=1}^{n} 2(y_j - a - bx_j - cx_j^2)(-x_j^2)$$

[1] In general, the first partial derivatives all 0 is a necessary condition for a minimizer but not a sufficient condition. We will eventually learn a theorem that gives additional restrictions needed for the condition to be sufficient as well as necessary.

Rearranging these equations to put the unknown variables a, b, and c on one side, we find (after dividing by 2),

$$\begin{aligned} a\sum_{j=1}^{n} 1 + b\sum_{j=1}^{n} x_j + c\sum_{j=1}^{n} x_j^2 &= \sum_{j=1}^{n} y_j \\ a\sum_{j=1}^{n} x_j + b\sum_{j=1}^{n} x_j^2 + c\sum_{j=1}^{n} x_j^3 &= \sum_{j=1}^{n} y_j x_j \\ a\sum_{j=1}^{n} x_j^2 + b\sum_{j=1}^{n} x_j^3 + c\sum_{j=1}^{n} x_j^4 &= \sum_{j=1}^{n} y_j x_j^2 \end{aligned}$$

Remember that the x_j and y_j are just given numbers. We can rewrite the above system of equations in matrix form as

$$\begin{bmatrix} \sum_{j=1}^{n} 1 & \sum_{j=1}^{n} x_j & \sum_{j=1}^{n} x_j^2 \\ \sum_{j=1}^{n} x_j & \sum_{j=1}^{n} x_j^2 & \sum_{j=1}^{n} x_j^3 \\ \sum_{j=1}^{n} x_j^2 & \sum_{j=1}^{n} x_j^3 & \sum_{j=1}^{n} x_j^4 \end{bmatrix} \begin{bmatrix} a \\ b \\ c \end{bmatrix} = \begin{bmatrix} \sum_{j=1}^{n} y_j \\ \sum_{j=1}^{n} y_j x_j \\ \sum_{j=1}^{n} y_j x_j^2 \end{bmatrix}$$

which is the form $\mathbf{Mu} = \mathbf{v}$ that we wanted. Note also that $\sum_{j=1}^{n} 1 = n$.

11.2.3 Homework 2

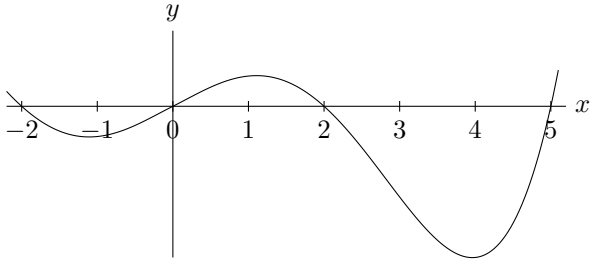
1. Read sections 2.1, 2.2, 2.3, 2.4, 2.5 in the book.

2. Problems:

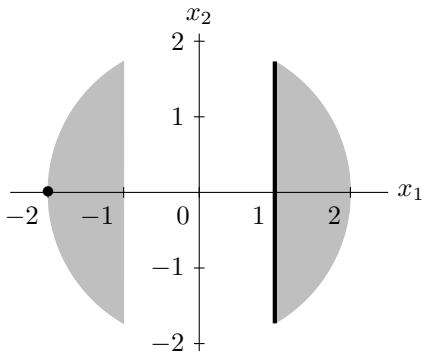
Page 47, #2.1 **Solution.** All the given points are feasible (plug them in, and see that they satisfy the inequalities). x_b, x_c, and x_e are on the boundary of the first constraint (the satisfy not just the inequality, but are actually equal). x_e is on the boundary of the second constraint. x_b, x_c, and x_d are on the boundary of the third constraint.

Page 47, #2.2 $f(x) = (x+1)x(x-2)(x-5) = x^4 - 6x^3 + 3x^2 + 10x$.

Solution. Stationary points at (roughly) $x = -1, 1, 4$. There are local minima at about $x = -1, 4$. There is a local maximum at about $x = 1$. There is no global maximum.



Page 47, #2.3 **Solution.** The feasible set is the shaded region below. The set of local minimizers is draw in black (the vertical line segment, and the points at $(-2, 0)$). The global minmimizer is the point $(-2, 0)$.



Page 52, #3.3 Prove that the set $S = \{\mathbf{x} : A\mathbf{x} \leq \mathbf{b}\}$ is convex.

Solution. To show convexity, we need to take two arbitrary points in S, and show that any point on the line joining them is also in S.

Let $\mathbf{x}, \mathbf{y} \in S$ be arbitrary, and let $0 \leq \alpha \leq 1$ be arbitrary. We will show that $\mathbf{z}$, defined by $\mathbf{z} = \alpha\mathbf{x} + (1-\alpha)\mathbf{y} \in S$. We need to show that $A\mathbf{z} \leq \mathbf{b}$.

$$\begin{aligned} A\mathbf{z} &= A(\alpha\mathbf{x} + (1-\alpha)\mathbf{y}) \\ &= \alpha A\mathbf{x} + (1-\alpha)A\mathbf{y} && \text{(by properties of matrices)} \\ &\leq \alpha\mathbf{b} + (1-\alpha)\mathbf{b} && \text{(since } \mathbf{x}, \mathbf{y} \in S\text{)} \\ &= \mathbf{b} \end{aligned}$$

Thus $A\mathbf{z} \leq \mathbf{b}$, so $\mathbf{z} \in S$. Since $\mathbf{x}, \mathbf{y}, \alpha$ were arbitrary, S is convex.

Page 52, #3.12 Let $g_1, \ldots, g_m$ be concave functions on $\mathbb{R}^n$. Prove that the set $S = \{\mathbf{x} : g_i(\mathbf{x}) \geq 0, i = 1, \ldots, m\}$ is convex.

Solution. Let $\mathbf{x}, \mathbf{y} \in S$ be arbitrary, and let $0 \leq \alpha \leq 1$ be arbitrary. Since $\mathbf{x}, \mathbf{y} \in S$, we know that

$$\begin{aligned} g_i(\mathbf{x}) \geq 0 \text{ for each } i = 1, \ldots, m, \text{ and} \\ g_i(\mathbf{y}) \geq 0 \text{ for each } i = 1, \ldots, m. \end{aligned}$$

Since each g_i is concave, we also have, for each $i = 1, \ldots, m$,

$$\begin{aligned} g_i(\alpha\mathbf{x} + (1-\alpha)\mathbf{y}) &\geq \alpha g_i(\mathbf{x}) + (1-\alpha)g_i(\mathbf{y}) \\ &\geq \alpha 0 + (1-\alpha)0 && \text{(since } \mathbf{x}, \mathbf{y} \in S\text{)} \\ &= 0. \end{aligned}$$

Thus, $g_i(\alpha\mathbf{x} + (1-\alpha)\mathbf{y}) \geq 0$, so $\alpha\mathbf{x} + (1-\alpha)\mathbf{y} \in S$. Thus, S is convex.

Page 52, #3.13 Let f be a convex function on the convex set S. Prove that the level set $T = \{\mathbf{x} : f(\mathbf{x}) \leq k\}$ is convex for all real numbers k.

Solution. Let $\mathbf{x}, \mathbf{y} \in T$ be arbitrary, and let $0 \leq \alpha \leq 1$ be arbitrary. Since $\mathbf{x}, \mathbf{y} \in S$, we know that

$$\begin{aligned} f(\mathbf{x}) \leq k, \text{ and} \\ f(\mathbf{y}) \leq k. \end{aligned}$$

Since f is convex, we also have

$$\begin{aligned} f(\alpha\mathbf{x}+(1-\alpha)\mathbf{y}) &\le \alpha f(\mathbf{x})+(1-\alpha)f(\mathbf{y}) \\ &\le \alpha k+(1-\alpha)k \qquad\qquad (\text{since } \mathbf{x},\mathbf{y}\in T) \\ &= k. \end{aligned}$$

Thus, $f(\alpha\mathbf{x}+(1-\alpha)\mathbf{y}) \le k$, so $\alpha\mathbf{x}+(1-\alpha)\mathbf{y} \in T$. Thus, T is convex.

Page 62, #5.1(v) [This problem was retracted, and will not be graded.]

Page 62, #5.2 Consider the sequence defined by $x_0 = a > 0$, and

$$x_{k+1} = \tfrac{1}{2}\left(x_k + \frac{a}{x_k}\right). \tag{11.1}$$

Prove that this sequence converges to $x_* = \sqrt{a}$.

Fun fact: This is exactly Newton's method used for computing $\sqrt{a}$, that is, finding a positive zero of $f(x) = x^2 - a$, since for Newton's method,

$$\begin{aligned} x_{k+1} &= x_k - \frac{f(x_k)}{f'(x_k)} \\ &= x_k - \frac{x_k^2 - a}{2x_k} \\ &= x_k - \frac{x_k^2}{2x_k} + \frac{a}{2x_k} \\ &= \tfrac{1}{2}\left(x_k + \frac{a}{x_k}\right). \end{aligned}$$

This method of finding square roots is also called the Babylonian method, as well as "Hero's method," after Hero of Alexandria.

Solution. (Note: the case $a = 2$ was already shown in class.) First, note that since $x_0 = a > 0$, we must always have $x_{k+1} > 0$, since $x_{k+1} = \frac{1}{2}\left(x_k + \frac{a}{x_k}\right) > 0$. (Note: This does not imply that the limit is positive.)

Next, *assume* that the sequence converges (later, we will show convergence). Then $x_k \to x_*$. We will show that $x_* = \sqrt{a}$. ***If*** it converges (to something other than zero), then we also have $x_{k+1} \to x_*$ and $a/x_{k+1} \to a/x_*$. Thus, taking the limit of the defining equation,

$$x_* = \tfrac{1}{2}\left(x_* + \frac{a}{x_*}\right)$$

After a little algebra, we find $x_*^2 = a$, so $x_* = \sqrt{a}$.

Next, denote $e_k = x_k - x_* = x_k - a$. Then, using equation (11.1), we find

$$\begin{aligned} e_{k+1} &= x_{k+1} - x_* \\ &= \tfrac{1}{2}\left(x_k + \frac{a}{x_k}\right) - \sqrt{a} \\ &= \tfrac{1}{2x_k}\left(x_k^2 + a - 2\sqrt{a}x_k\right) \\ &= \tfrac{1}{2x_k}\left(x_k - \sqrt{a}\right)^2 \\ &= \tfrac{1}{2x_k}e_k^2 \end{aligned}$$

Thus, $e_{k+1}/e_k^2 = \frac{1}{2x_k}$, so

$$\lim_{k\to\infty} \frac{e_{k+1}}{e_k^2} = \lim_{k\to\infty} \frac{1}{2x_k} = \frac{1}{2\sqrt{a}}.$$

This means that the convergence is quadratic with constant $C = \frac{1}{2\sqrt{a}}$. (Note: This means that finding square roots of larger numbers, the method will converge faster!)

Proof of convergence. (Not required for full credit.)

As already noted above, $x_k > 0$ for all k. Next, notice that, from the above calculation for the error, that

$$x_{k+1} - \sqrt{a} = e_{k+1} = \frac{1}{2x_k} e_k^2 \geq 0.$$

Thus, $x_{k+1} \geq \sqrt{a}$ for all $k \geq 1$. This means that

$$x_k^2 - a \geq 0 \text{ for all } k \geq 2. \tag{11.2}$$

Next, notice that, for all $k \geq 2$,

$$x_k - x_{k+1} = x_k - \frac{1}{2}\left(x_k + \frac{a}{x_k}\right) = \frac{x_k}{2} - \frac{a}{2x_k} = \frac{x_k^2 - a}{2x_k} \geq 0$$

thanks to (11.2), and the (already established) fact that $x_k > 0$. This means that $x_k \geq x_{k+1}$. This means that

$$x_2 \geq x_3 \geq x_4 \geq \cdots \geq \sqrt{2}.$$

Thus, for $k \geq 2$, x_k is a decreasing sequence which is bounded below by $\sqrt{2}$, so it converges to some point x_*, and also $x_* \geq \sqrt{2} > 0$.

3. Write a Matlab function to compute the limit in Section 5.2 using the sequence. Make the function input the number, `a`, and the number of iterations. Make it output the final iteration, and the absolute value of the error between the final iteration and the exact solution, `sqrt(a)`. Make the code give an error if the input `a` is negative. The whole code will be fairly short, maybe 10 lines or fewer, depending on how you code it. **If you get stuck, review *Part 10: Functions* in the Matlab Introduction file.**

   Solution. Here is one possible example code:

```
function x = sqrtNewton(a,numIter)
% A function to compute the square root of 'a'
% using Newton's method (i.e., the Babylonian method).

if ( a < 0 )
   error('Input must be non-negative');
elseif (a == 0)
   x = 0;
   return;
end

x = a;
for k = 1:numIter
   x = 0.5*(x + a/x);
end

display(sprintf('error = %g',abs(x-sqrt(a))));
```

11.2.4 Homework 3

1. Read sections 2.6, 2.7, 2.8, 11.1, and 11.2 in the book.

2. Page 65, #6.1. Find the first four terms in the Taylor series expansion of $f(x) = \log(1+x) = \log_e(1+x) = \ln(1+x)$. Evaluation for $p = 0.1$ and $p = 0.01$, and compare, deriving a bound for the accuracy.

   Solution. Note that $\frac{d^n}{dx^n}\log(1+x) = \frac{(-1)^{n-1}(n-1)!}{(1+x)^n}$ for $n = 1, 2, 3, \ldots$. For the Taylor series, we have:

   $$f(x_0+p) = f(x_0) + pf'(x_0) + \tfrac{p^2}{2}f''(x_0) + \tfrac{p^3}{3!}f'''(x_0) + \tfrac{p^4}{4!}f^{(4)}(\xi)$$

   for some ξ satisfying $x_0 < \xi < x_0 + p$. Thus, for $x_0 = 0$, we have

   $$\begin{aligned}\log(1+p) &= \log(1+0) + p\frac{1}{1+0} + \tfrac{p^2}{2}\frac{-1}{(1+0)^2} + \tfrac{p^3}{3!}\frac{2}{(1+0)^3} + \tfrac{p^4}{4!}\frac{-3}{(1+\xi)^4} \\ &= p - \tfrac{p^2}{2} + \tfrac{p^3}{3} - \tfrac{p^4}{8}\frac{1}{(1+\xi)^4}\end{aligned}$$

   These are four terms, with the first term being zero. I will leave it to you to check the decimals. To find a bound on the accuracy, we see from the above equation that, since $\xi > x_0 = 0$,

   $$\left|\log(1+p) - \left(p - \tfrac{p^2}{2} + \tfrac{p^3}{3}\right)\right| = \left|\tfrac{p^4}{8}\frac{1}{(1+\xi)^4}\right| \le \left|\tfrac{p^4}{8}\frac{1}{(1+0)^4}\right| = \frac{p^4}{8}$$

   For example, if $p = 0.1$, the error in approximating by the first four terms is at most $0.1^4/8 = 1.25 \cdot 10^{-5}$.

3. Page 66, # 6.4. Find the first three terms of the Taylor series for $f(x_1, x_2) = 3x_1^4 - 2x_1^3x_2 - 4x_1^2x_2^2 + 5x_1x_2^3 + 2x_2^4$ at $\mathbf{x}_0 = \left(\begin{smallmatrix}1\\-1\end{smallmatrix}\right)$.

   Solution. Recall the Taylor series (with ∇^2 denoting the Hessian):

   $$f(\mathbf{x}_0 + \mathbf{p}) \approx f(\mathbf{x}_0) + \mathbf{p}^T\nabla f(\mathbf{x}_0) + \mathbf{p}^T\nabla^2 f(\mathbf{x}_0)\mathbf{p}$$

We compute $f(\mathbf{x}_0) = -2$, and

$$\nabla f(\mathbf{x}) = \begin{bmatrix} 12x_1^3 - 6x_1^2x_2 - 8x_1x_2^2 + 5x_2^3 \\ -2x_1^3 - 8x_1^2x_2 + 15x_1x_2^2 + 8x_2^3 \end{bmatrix},$$

and

$$\nabla^2 f(\mathbf{x}) = \begin{bmatrix} 36x_1^2 - 12x_1x_2 - 8x_2^2 & -6x_1^2 - 16x_1x_2 + 15x_2^2 \\ -6x_1^2 - 16x_1x_2 + 15x_2^2 & -8x_1^2 + 30x_1x_2 + 24x_2^2 \end{bmatrix}.$$

Thus,

$$\nabla f(\mathbf{x}_0) = \begin{bmatrix} 5 \\ 13 \end{bmatrix}$$

and

$$\nabla^2 f(\mathbf{x}_0) = \begin{bmatrix} 40 & 25 \\ 25 & -14 \end{bmatrix}$$

We find:

$$\begin{aligned} f(\mathbf{x}_0 + \mathbf{p}) &\approx f(\mathbf{x}_0) + \mathbf{p}^T \nabla f(\mathbf{x}_0) + \tfrac{1}{2}\mathbf{p}^T \nabla^2 f(\mathbf{x}_0)\mathbf{p} \\ &= -2 + \mathbf{p}^T \begin{bmatrix} 5 \\ 13 \end{bmatrix} + \tfrac{1}{2}\mathbf{p}^T \begin{bmatrix} 40 & 25 \\ 25 & -14 \end{bmatrix} \mathbf{p} \\ &= -2 + 5p_1 + 13p_2 + 20p_1^2 + 25p_1p_2 - 7p_2^2 \end{aligned}$$

4. Page 66, # 6.6 Prove that if $\mathbf{p}^T \nabla f(\mathbf{x}_k) < 0$, then $f(\mathbf{x}_k + \epsilon\mathbf{p}) < f(\mathbf{x}_k)$ for $\epsilon > 0$ sufficiently small.

   Solution. Recall that, by first-order Taylor approximation:

   $$f(\mathbf{x}_k + \epsilon\mathbf{p}) = f(\mathbf{x}_k) + \epsilon\mathbf{p}^T \nabla f(\xi)$$

   for some ξ lying on the line between $\mathbf{x}_k$ and $\mathbf{x}_k + \epsilon\mathbf{p}$. (This is also known as the Mean-Value Theorem.) If f is smooth (as we usually assume), then ∇f is continuous, and therefore, so is $\mathbf{p}^T \nabla f(\mathbf{x})$. For a function is continuous and negative at a point $\mathbf{x}_k$, then it is negative for all $\mathbf{x}$ sufficiently close to $\mathbf{x}_k$ (to see this, draw a picture). Moreover, as ϵ gets small, $\mathbf{x}_k + \epsilon\mathbf{p}$ get closer to $\mathbf{x}_k$, so ξ approaches $\mathbf{x}_k$ (since again, ξ lies on the line between $\mathbf{x}_k$ and $\mathbf{x}_k + \epsilon\mathbf{p}$). Thus, for sufficiently small $\epsilon > 0$, $\mathbf{p}^T \nabla f(\xi) < 0$, since $\mathbf{p}^T \nabla f(\mathbf{x}_k) < 0$. Therefore,

   $$f(\mathbf{x}_k + \epsilon\mathbf{p}) = f(\mathbf{x}_k) + \epsilon\mathbf{p}^T \nabla f(\xi) < f(\mathbf{x}_k) + \epsilon 0 = f(\mathbf{x}_k).$$

5. Page 74, # 7.1. For this problem, write a code to use Newton's method. You can write a function[2] to do this. Here is an example of how a function might start:

```
1    function x = Newton(f,df,x0,tolerence,maxIter)
2    % A program to solve f(x) = 0 using Newton's method.
```

   where `maxIter` is the maximum number of iterations you will allow, and `tolerence` is the maximum error between steps you will allow. Once you write the code, you can call it from the command line, as

[2] See the Matlab Intro, Section 10, if you need a refresher on Matlab functions.

in the following example, which uses Newton's method with initial guess $x_0 = 2$ to compute $\sqrt{3}$ out to 15 decimal places. (Note that the derivative $\frac{d}{dx}(x^2 - 3) = 2x$ has been computed by hand.)

```
>> format long g
>> x = Newton(@(x) x^2 - 3, @(x) 2*x, 2, 1e-15,1000)
```

(The first line makes Matlab display more decimal places; you only need to enter it once per session.) Include your answers for page 74, # 7.1, and also print your Newton code and staple it to your homework.

Solution. One possible solution is included at the end of this document.

6. Page 362, # 2.3. First, we compute when the gradient is equal to zero:

$$\nabla f(\mathbf{x}) = \begin{pmatrix} 16x_1 + 3x_2 - 25 \\ 3x_1 + 14x_2 + 31 \end{pmatrix} = \begin{pmatrix} 0 \\ 0 \end{pmatrix}$$

Solving this system, we find the solution to be $\mathbf{x}_* = (443/215, -571/215)$. Thus, there is a single critical point. To check that $\mathbf{x}_*$ is a minimizer, we compute the Hessian at $\mathbf{x}_*$:

$$H_f(\mathbf{x}_*) = \nabla^2 f(\mathbf{x}_*) = \begin{pmatrix} 16 & 3 \\ 3 & 14 \end{pmatrix}$$

Note that since the determinate and upper left corner are both positive, by Sylvester's Criterion, $H_f(\mathbf{x}^*)$ is positive-definite. By a theorem we learned in class, any local minimizer of a convex function is a global minimizer, and moreover, it is unique. (Note: there are convex functions without any local or global minimizers; consider $f(x) = e^x$.)

One solution to problem #4:

```
function x = Newton(f,df,x0,tolerence,maxIter)
% A program to solve f(x) = 0 using Newton's method:
%     x_0 = initial guess;
%     x_{n+1} = x_n - f(x_n)/f'(x_n);
% Until either |x_{n+1} - x_n| < tolerence
%                  and |f(x_n)| < tolerence
%           or maxIter is reached.
% We calculate f' by hand and input it as df.
%
% Example run (to find sqrt(3)):
% x = Newton(@(x) x^2 -3, @(x)2*x, 3.0, 10^(-15), 1000)
%
% Example of run that doesn't converge:
% x = Newton(@(x) sin(x), @(x) cos(x), pi/2, 10^(-15), 1000)

% Initialize
x = x0;

for count = 1:maxIter
    x_old = x; % Store the old value to check for error later.

    % Newton iteration:
    x = x - f(x)/df(x);

    if ((abs(x-x_old) < tolerence) && (abs(f(x)) < tolerence))
        break;
    end
end

if (count < maxIter)
    display(sprintf('Exited after %d iterations.',count));
else
    warning('Maximum iterations reached. May not be converged.');
end
```

11.2.5 Homework 4

1. Read sections 11.3, 11.4 A.6, and A.7 in the book.
2. Page 363, #2.9. Let $f(\mathbf{x}) = 2x_1^2 + x_2^2 - 2x_1x_2 + 2x_1^3 + x_1^4$. Determine the minimizers/maximizers of f and indicate what kind of minima or maxima (local, global, etc.) they are.

   Solution. We need to find where the gradient is zero. At those points, if the Hessian is positive definite, the point is a minimum, and if the Hessian is negative-definite, the point is a maximum. We compute

$$\nabla f(\mathbf{x}) = \begin{bmatrix} 4x_1 - 2x_2 + 6x_1^2 + 4x_1^3 \\ 2x_2 - 2x_1 \end{bmatrix}$$

   and

$$H(\mathbf{x}) = \nabla^2 f(\mathbf{x}) = \begin{bmatrix} 4 + 12x_1 + 12x_1^2 & -2 \\ -2 & 2 \end{bmatrix}$$

Setting $\nabla f(\mathbf{x}) = \mathbf{0}$, we see that

$$\begin{aligned} 4x_1 - 2x_2 + 6x_1^2 + 4x_1^3 &= 0 \\ 2x_2 - 2x_1 &= 0 \end{aligned}$$

so that, from the second equation, $x_1 = x_2$. Substituting this back into the first equation yields

$$\begin{aligned} 0 &= 4x_1 - 2x_1 + 6x_1^2 + 4x_1^3 = 2x_1 + 6x_1^2 + 4x_1^3 = 2x_1(1 + 3x_1 + 2x_1^2) \\ &= 2x_1(2x_1 + 1)(x_1 + 1) \end{aligned}$$

Thus, $x_1 = 0$, $-\frac{1}{2}$, or -1. Since $x_2 = x_1$, the critical points are $(0,0)$, $(-\frac{1}{2}, -\frac{1}{2})$, and $(-1,-1)$. At these points, the Hessian is:

$$\begin{aligned} H((0,0)) &= \begin{bmatrix} 4 & -2 \\ -2 & 2 \end{bmatrix} \\ H((-\tfrac{1}{2}, -\tfrac{1}{2})) &= \begin{bmatrix} 1 & -2 \\ -2 & 2 \end{bmatrix} \\ H((-1,-1)) &= \begin{bmatrix} 4 & -2 \\ -2 & 2 \end{bmatrix} \end{aligned}$$

Coincidentally, $H((0,0)) = H((-1,-1))$, so our work is reduced. To check it is positive definite, we can use Sylvester's criterion to notice that $h_{1,1}(0,0) = 4 > 0$ and $\det(H((0,0))) = (4)(2) - (-2)(-2) = 4 > 0$, so the matrix is positive definite, and both $(0,0)$ and $(-1,-1)$ are local minima. Furthermore, the function tends ∞ as $\|\mathbf{x}\| \to \infty$, so at least one local minimum must be a global mimimum. Since $f((0,0)) = f((-1,-1)) = 0$, they are both local minima. Next, notice that $h_{1,1}(-\frac{1}{2}, -\frac{1}{2}) = 1 > 0$, but $\det(H((-\frac{1}{2}, -\frac{1}{2}))) = (1)(2) - (-2)(-2) = -2 < 0$, so $H((-\frac{1}{2}, -\frac{1}{2}))$ is not positive definite. In fact, it is indefinite, and one can see in the plot below that f has a saddle-point at $(-\frac{1}{2}, -\frac{1}{2})$.

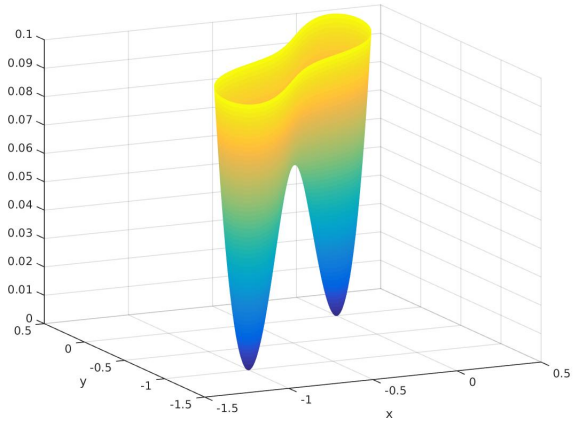


Figure 11.1: Plot of the function $f(\mathbf{x}) = 2x_1^2 + x_2^2 - 2x_1x_2 + 2x_1^3 + x_1^4$

By the way, here is how I plotted that in Matlab:

```
h = figure;
X = linspace(-1.5,0.5,200);
Y = linspace(-1.5,0.5,200);
[x y] = meshgrid(X,Y);
surf(x,y,2*x.^2 + y.^2-2*x.*y+2*x.^3+x.^4);
xlabel('x');
ylabel('y');
axis([-1.5 0.5 -1.5 0.5 0 0.1]);
shading interp;
lighting phong;
caxis([0,0.1]);
print(h,'-djpeg','HW4prob1.jpg'); % Save plot to a file.
```

3. Page 369, #3.2. Use the program you wrote in Assignment #3 for this one. Write a very brief description (no more than a short paragraph) of your methods, and report your results.

   Use Newton's method to solve:

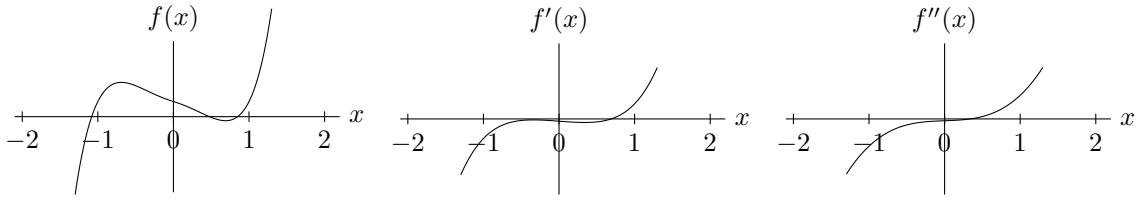
$$\text{minimize} f(x) = 5x^5 + 2x^3 - 4x^2 - 3x + 2$$

   Look for the solution in the interval $[-2, 2]$. Make sure you have found a minimum and not a maximum.

   Solution. Newton's method for minimization (via solving $f'(x) = 0$) takes the form:

$$x_{k+1} = x_k - \frac{f'(x_k)}{f''(x_k)} = x_k - \frac{25x^4 + 6x^2 - 8x - 3}{100x^3 + 12x - 8}$$

   It may help to see the plots below. Notice the very subtle oscillation near the origin.



   Here's a way to do it by brute force (which is totally overkill): check a few thousand initial guesses on the interval using a loop. (See previous homework's answer key for a coded Newton's method.)

```
df = @(x) 25*x^4+6*x^2-8*x-3;
ddf = @(x) 100*x^3+12*x-8, x0;

i=1;
x = zeros(1,1000);
for x0 = linspace(-2,2,1000);
    x(i) = Newton(df, ddf, x0, 10^(-15), 1000);
    i = i+1;
end;
display(unique(x)');
```

   I found $x = -0.289897948556636$ and $x = 0.689897948556636$ from this search. However,

```
>> ddf(-0.289897948556636)
ans =
          -13.9151015307185
>> ddf(0.689897948556636)
ans =
           33.1151015307186
```

so by the second derivative test, only $x = 0.689897948556636$ is a local minimum.

4. Page 370, #3.6 (You may want to read Appendix B.5 starting on page 696.) Consider the problem

$$\text{minimize } f(\mathbf{x}) = \tfrac{1}{2}\mathbf{x}^T Q\mathbf{x} - \mathbf{c}^T\mathbf{x}$$

Where Q is a (symmetric) positive definite matrix. Prove that Newton's method will converge in one step, regardless of the starting point.

Solution. As we saw in class,

$$\begin{aligned}\nabla f(\mathbf{x}) &= Q\mathbf{x} - \mathbf{c}\\ \nabla^2 f(\mathbf{x}) &= Q\end{aligned}$$

Thus, the critical points are solutions to $\mathbf{0} = \nabla f(\mathbf{x}) = Q\mathbf{x} - \mathbf{c}$, that is, solutions to $Q\mathbf{x} = \mathbf{c}$. Recall that very SPD matrix is intervible. Thus, the solution to $Q\mathbf{x} = \mathbf{c}$ is unique, and it is give by $\mathbf{x}_* = Q^{-1}\mathbf{c}$. Also, since $\nabla^2 f(\mathbf{x}) = Q$ is SPD, the function f is (strictly) convex, so the critical point $\mathbf{x}_* = Q^{-1}\mathbf{c}$ must be the unique global minimum, and are there are no other local minima.

Now,, Newton's method for minimization is

$$\begin{aligned}\mathbf{x}_{k+1} &= \mathbf{x}_k - (\nabla^2 f(\mathbf{x}_k))^{-1}(\nabla f(\mathbf{x}_k))\\ &= \mathbf{x}_k - Q^{-1}(Q\mathbf{x}_k - \mathbf{c})\\ &= \mathbf{x}_k - \mathbf{x}_k + Q^{-1}\mathbf{c}\\ &= Q^{-1}\mathbf{c} = \mathbf{x}_*\end{aligned}$$

Thus, no mater what the starting position is, Newton's method converges in one step. However, this is not especially useful in this case, since to perform the first step, we need to solve $Q\mathbf{x} = \mathbf{c}$ anyway, which is the main difficulty to begin with. Applying Newton's method therefore doesn't reduce the complexity of this problem. It is more useful for problems with worse nonlinearity.

5. (a) Find an LDL^T factorization of the following symmetric positive-definite matrix

$$A = \begin{bmatrix}1 & 2 & 4\\ 2 & 5 & 6\\ 4 & 6 & 22\end{bmatrix},$$

that is, write $A = LDL^T$ where L is lower-triangular, and D is diagonal.

Solution. First, we find an LU factorization using Gaussian elimination without pivoting.

$$A = \begin{bmatrix} 1 & 2 & 4 \\ 2 & 5 & 6 \\ 4 & 6 & 22 \end{bmatrix}$$
$$\sim \begin{bmatrix} 1 & 2 & 4 \\ 0 & 1 & -2 \\ 0 & -2 & 6 \end{bmatrix}; \qquad L = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 4 & * & 1 \end{bmatrix}$$
$$\sim \begin{bmatrix} 1 & 2 & 4 \\ 0 & 1 & -2 \\ 0 & 0 & 2 \end{bmatrix} = U; \qquad L = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 4 & -2 & 1 \end{bmatrix}$$

In class, we saw that $D = L^{-1}U^T$. Now, what does L^{-1} look like? Since L is lower-triangular, so is L^{-1} (to see why, write $LM = I$ where I is the identity, and see what the entries of M must be). Also, since L has only 1's on the diagonal, the same is true about L^{-1}, since

$$\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} = I = LL^{-1} = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 4 & -2 & 1 \end{bmatrix} \begin{bmatrix} a & 0 & 0 \\ * & b & 0 \\ * & * & c \end{bmatrix}$$

so the only options are $a = b = c = 1$. Thus,

$$D = L^{-1}U^T = \begin{bmatrix} 1 & 0 & 0 \\ * & 1 & 0 \\ * & * & 1 \end{bmatrix} \begin{bmatrix} 1 & 2 & 4 \\ 0 & 1 & -2 \\ 0 & 0 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix}$$

Note: We did <u>not</u> compute L^{-1} here. We only used properties of it. We now have,

$$A = LDL^T = \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 4 & -2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 4 & -2 & 1 \end{bmatrix}^T.$$

(b) Rewrite the LDL^T factorization to form an LL^T (i.e. Cholesky) factorization.

<u>Solution.</u> Using D from above, we find

$$D = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \sqrt{2} \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \sqrt{2} \end{bmatrix} := \sqrt{D}\sqrt{D}.$$

Therefore,

$$\begin{aligned} A = LDL^T &= L\sqrt{D}\sqrt{D}L^T \\ &= \left(\begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 4 & -2 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \sqrt{2} \end{bmatrix} \right) \left(\begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & \sqrt{2} \end{bmatrix} \begin{bmatrix} 1 & 2 & 4 \\ 0 & 1 & -2 \\ 0 & 0 & 1 \end{bmatrix} \right) \\ &= \begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 4 & -2 & \sqrt{2} \end{bmatrix} \begin{bmatrix} 1 & 2 & 4 \\ 0 & 1 & -2 \\ 0 & 0 & \sqrt{2} \end{bmatrix} = \widetilde{L}\widetilde{L}^T. \end{aligned}$$

This is the Cholesky factorization, which is unique.

(c) Use either LDL^T or the LL^T the factorization of the above matrix to solve $A\mathbf{x} = \mathbf{b}$, where

$$\mathbf{b} = \begin{bmatrix} 1 \\ 6 \\ -6 \end{bmatrix}.$$

Hint: Use forward substitution and back substitution.

Solution. We only demonstrate the Cholesky solve here; LDL^T is similar. The idea is to solve is stages. First, note that solving $A\mathbf{x} = \mathbf{b}$ is the same as solving $LL^T\mathbf{x} = \mathbf{b}$, since $A = LL^T$. Thus, we solve first $L\mathbf{y} = \mathbf{b}$, and then $L^T\mathbf{x} = \mathbf{y}$, and we do it **not by inverting a matrix**, but by simple substitution, which is ***much*** faster.

$$\begin{bmatrix} 1 & 0 & 0 \\ 2 & 1 & 0 \\ 4 & -2 & \sqrt{2} \end{bmatrix} \begin{bmatrix} y_1 \\ y_2 \\ y_3 \end{bmatrix} = \begin{bmatrix} 1 \\ 6 \\ -6 \end{bmatrix}$$

We quickly see that $y_1 = 1$. Then $2y_1 + 1y_2 = 6$, so $y_2 = 4$. Then $4y_1 - 2y_2 + \sqrt{2}y_3 = -6$, so $y_3 = -\sqrt{2}$. Next, we solve

$$\begin{bmatrix} 1 & 2 & 4 \\ 0 & 1 & -2 \\ 0 & 0 & \sqrt{2} \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \end{bmatrix} = \begin{bmatrix} 1 \\ -6 \\ -\sqrt{2} \end{bmatrix}$$

This solve goes backwards. We first get $x_3 = \frac{1}{\sqrt{2}}(-\sqrt{2}) = -1$. Next, we work on x_2 to find $1x_2 - 2x_3 = -6$ so that $x_2 = 2$. Finally, $1x_2 + 2x_2 + 4x_3 = 1$, so $x_3 = 1$. Thus

$$\mathbf{x} = \begin{bmatrix} 1 \\ 2 \\ -1 \end{bmatrix}.$$

6. **Bonus (1 point):** Write a program that computes the Cholesky factorization. It is OK to look online or at other resources to get ideas, but then close the webpage, book, etc., and write your code on your own. You will learn more and grow stronger if you do it this way.

   Solution. See the following Matlab code, saved as `cholesky.m`, for one possible solution.

```
function L = cholesky(A);
% Compute the Cholesky decomposition of an SPD matrix A.
n = size(A,1);
L = zeros(n,n); % Wasteful, since we only need to store the lower-tri part.

tol = 1e-15;

for j = 1:n
    LjjSquared = A(j,j) - L(j,:)*L(j,:)';
    if LjjSquared < tol
        error('Matrix is either not SPD, or badly scaled.');
    end
    L(j,j) = sqrt(LjjSquared);

    for i = (j + 1):n
        L(i,j) = (A(i,j) - L(j,:)*L(i,:)')/L(j, j);
    end
end

end
```

11.2.6 Homework 5

Note 1: This assignment involves coding. It is OK to talk with other people, but please write your own code. It is very obvious when one person just copies another persons code, or reproduces something they found on the internet without understanding it. (It is especially obvious when you ask them to explain their code!) So, try to code things by yourself, talking to other people or checking other resources only if you get *really* stuck. This will make you stronger! Also, staring at code for a few minutes and being confused does not count as getting stuck. This is part of coding! Progress is usually not continuous. Programming is a puzzle that you solve, not a bucket that you fill.

Note 2: Next week on 2016.03.10 (i.e., Thursday March 10th), we will learn about another tool to make your codes even faster for certain types of matrices. We will have a short bonus assignment over that weekend which is a contest to see who can code the best and fastest solver! The details of this contest will be released on 2016.03.10. In the meantime, try to make your codes as good as possible to be ready for the contest!

1. Review the Gram-Schmidt method from Linear Algebra.

2. Read sections 12.1, 12.2, 13.1, 13.2 in the book. You might also want to read my notes on Steepest Descent, posted on the webpage at:
   www.math.unl.edu/~alarios2/courses/2016_spring_M433/content.shtml

3. Page 408, #2.1. Do the calculations by hand (show your work). This will help you understand what is going on, and give you a feeling for how fast the computations are. Think about what the matrix A and the vector $\mathbf{b}$ are in this problem.

   Use the steepest-descent method to solve: *minimize* $f(x_1, x_2) = 4x_1^2 + 2x_2^2 + 4x_1x_2 - 3x_1$, starting from the point $(2, 2)^T$. Perform three iterations.

Solution. First we'll need to find what A and $\vec{b}$ are. We can find A by finding the Hessian of f and we can find $\vec{b}$, by finding the gradient and considering the values which contain no variables. Doing this we get:

$$A = \begin{pmatrix} 8 & 4 \\ 4 & 4 \end{pmatrix} \text{ and } \vec{b} = \begin{pmatrix} 3 \\ 0 \end{pmatrix}$$

Next, we'll perform the three iterations of steepest-descent.

first iteration:

$$\vec{r_0} = \vec{b} - A\vec{x_0} = \begin{pmatrix} 3 \\ 0 \end{pmatrix} - \begin{pmatrix} 8 & 4 \\ 4 & 4 \end{pmatrix} \begin{pmatrix} 2 \\ 2 \end{pmatrix} = \begin{pmatrix} -21 \\ -16 \end{pmatrix}$$

$$\alpha_0 = \frac{\vec{r_0}^T \vec{r_0}}{\vec{r_0}^T A \vec{r_0}} = \frac{\begin{pmatrix} -21 \\ -16 \end{pmatrix}^T \begin{pmatrix} -21 \\ -16 \end{pmatrix}}{\begin{pmatrix} -21 \\ -16 \end{pmatrix}^T \begin{pmatrix} 8 & 4 \\ 4 & 4 \end{pmatrix} \begin{pmatrix} -21 \\ -16 \end{pmatrix}} \approx 0.09627$$

$$\vec{x_1} = \vec{x_0} + \alpha_0 \vec{r_0} = \begin{pmatrix} 2 \\ 2 \end{pmatrix} + 0.09627 \begin{pmatrix} -21 \\ -16 \end{pmatrix} \approx \begin{pmatrix} -0.02169 \\ 0.45967 \end{pmatrix}$$

second iteration:

$$\vec{r_1} = \vec{b} - A\vec{x_1} = \begin{pmatrix} 3 \\ 0 \end{pmatrix} - \begin{pmatrix} 8 & 4 \\ 4 & 4 \end{pmatrix} \begin{pmatrix} -0.02169 \\ 0.45967 \end{pmatrix} \approx \begin{pmatrix} 1.33481 \\ -1.75193 \end{pmatrix}$$

$$\alpha_1 = \frac{\vec{r_1}^T \vec{r_1}}{\vec{r_1}^T A \vec{r_1}} = \frac{\begin{pmatrix} 1.33481 \\ -1.75193 \end{pmatrix}^T \begin{pmatrix} 1.33481 \\ -1.75193 \end{pmatrix}}{\begin{pmatrix} 1.33481 \\ -1.75193 \end{pmatrix}^T \begin{pmatrix} 8 & 4 \\ 4 & 4 \end{pmatrix} \begin{pmatrix} 1.33481 \\ -1.75193 \end{pmatrix}} \approx 0.62011$$

$$\vec{x_2} = \vec{x_1} + \alpha_1 \vec{r_1} = \begin{pmatrix} -0.02169 \\ 0.45967 \end{pmatrix} + 0.62011 \begin{pmatrix} 1.33481 \\ -1.75193 \end{pmatrix} \approx \begin{pmatrix} 0.80604 \\ -0.62672 \end{pmatrix}$$

third iteration:

$$\vec{r_2} = \vec{b} - A\vec{x_2} = \begin{pmatrix} 3 \\ 0 \end{pmatrix} - \begin{pmatrix} 8 & 4 \\ 4 & 4 \end{pmatrix} \begin{pmatrix} 0.80604 \\ -0.62672 \end{pmatrix} \approx \begin{pmatrix} -0.94144 \\ -0.71728 \end{pmatrix}$$

$$\alpha_2 = \frac{\vec{r_2}^T \vec{r_2}}{\vec{r_2}^T A \vec{r_2}} = \frac{\begin{pmatrix} -0.94144 \\ -0.71728 \end{pmatrix}^T \begin{pmatrix} -0.94144 \\ -0.71728 \end{pmatrix}}{\begin{pmatrix} -0.94144 \\ -0.71728 \end{pmatrix}^T \begin{pmatrix} 8 & 4 \\ 4 & 4 \end{pmatrix} \begin{pmatrix} -0.94144 \\ -0.71728 \end{pmatrix}} \approx 0.09627$$

$$\vec{x_3} = \vec{x_2} + \alpha_2 \vec{r_2} = \begin{pmatrix} 0.80604 \\ -0.62672 \end{pmatrix} + 0.09627 \begin{pmatrix} -0.94144 \\ -0.71728 \end{pmatrix} \approx \begin{pmatrix} 0.71541 \\ -0.69577 \end{pmatrix}$$

4. Code the Steepest Descent algorithm. First try the "Naïve form" we saw in class, then the "improved form". Is one actually faster than the other? Does the speed depend on anything, such as the matrix size, etc.? If so, when at what size can you see a difference? To keep things uniform, please start you code like this:

```
function [x,iter] = steepestDescent(A,b,x0,maxIter,tol)
% Use Steepest Descent algorithm to solve Ax = b.
```

where `x0` is the initial guess. `maxIter` is the maximum number of iterations, and `tol` is the tolerance. The code should stop once `maxIter` iterations have occured, or the desired level of accuracy is reached, determined by the tolerance `tol`. It is up to you what to base the tolerance on though (make it reasonable though).

Solution. Answers may vary considerably. Below is one possible solution.

```
function [x,iter] = steepestDescent(A,b,x0,maxIter,tol)
% Use Steepest Descent algorithm to solve Ax = b.
% Example usage:
%   n = 100; A = rand(n)-0.5; A = A'+A+n*eye(n);
%   x = rand(n,1); b = A*x;
%   [x_sd,iter] = steepestDescent(A,b,b,100,1e-6);
%   display([norm(x-x_sd)/norm(x),iter]);

x = x0;
r = b-A*x;
for iter = 1:maxIter
    r_sq = r'*r;
    if (r_sq < tol)
        break
    end
    Ar = A*r;
    alpha = r_sq/(r'*Ar);
    x = x + alpha*r;
    r = r - alpha*Ar;
end
```

Naïve version of the algorithm:

```
function [x,iter] = steepestDescentNaive(A,b,x0,maxIter,tol)
% Use Steepest Descent algorithm to solve Ax = b.
% Example usage:
%   n = 100; A = rand(n)-0.5; A = A'+A+n*eye(n);
%   x = rand(n,1); b = A*x;
%   [x_sd,iter] = steepestDescentNaive(A,b,b,100,1e-6);
%   display([norm(x-x_sd)/norm(x),iter]);

x = x0;
for iter = 1:maxIter
    r = b-A*x;
    r_sq = r'*r;
    if (r_sq < tol)
        break
    end
    alpha = r'*r/(r'*(A*r));
    x = x + alpha*r;
end
```

Next, test you code, maybe try something like this:

```
% A program to test linear solvers.
n = 100;
L = tril(rand(n,n)); % Random lower-triangular matrix
A = L*L'; % Random SPD matrix;
x_exact = rand(n,1); % Random vector (the exact solution)
b = A*x_exact; % Now x_exact is the exact solution of Ax = b.
tic; % Mark the starting time
%
% Here, call your steepestDescent program to solve Ax = b
% WITHOUT using x_exact.  Find x and the number of iterations.
%
time = toc;
error = norm(x_exact - x)/norm(x);
output_string = 'Error = %g, time = %g, iterations = %d';
display(sprintf(output_string,error,time,iter));
```

5. Plot a graph of the matrix size `n` vs. the number of iterations for tolerance `tol = 1e-3` for `n = 100, 110, 120,...,1000`. You may want to review the Matlab intro on plotting, located at:

www.math.unl.edu/~alarios2/courses/2016_spring_M433/documents/matlabIntroLarios.pdf

   It may help to turn the above test code into a function, or wrap it in a loop. Does the plot show what you expect it to show?

   Solution. Answers may vary greatly here. Roughly speaking, the number of iterations should increase as the matrix size increases. However, `maxIter` can easily be reached quickly.

6. Repeat problems 2 and 3 for the Conjugate Gradient algorithm (see page 454 in your book for the algorithm). Try to make your code as fast and efficient as you can, while still being readable!

Print out all your codes and the two graphs (plot titles and labels for the x and y axes are required!), and turn them in, stapled to your homework.

Solution. Answers may vary considerably. Below is one possible implementation of the conjugate gradient method. It is not optimal, but hopefully it is somewhat easier to read that an optimized code.

```
function [x,iter] = conjugateGradient(A,b,x0,maxIter,tol)
% Use Conjugate Gradient algorithm to solve Ax = b.
% Example usage:
%    n = 1000; A = tril(rand(n)); A = A*A';
%    x = rand(n,1); b = A*x;
%    [x_sd,iter] = CG(A,b,b,100,1e-6);
%    display([norm(x-x_sd)/norm(x),iter]);

x = x0;
r = b - A*x; % residual
p = r;       % search direction
for iter = 1:maxIter
    r_sq = r'*r; % Precompute for speed.
    if (sqrt(r_sq) < tol)
        % If residual is small enough, stop.
        break
    end
    Ap = A*p; % Precompute for speed.
    alpha = r_sq/(p'*Ap);
    x = x + alpha*p;
    r = r - alpha*Ap;
    beta = r'*r/r_sq; % r is now the new r, r_sq uses old r
    p = r + beta*p;
end
```

11.2.7 Homework 6

1. Read sections 3.1, 3.2, 14.1, and 14.2 in the book. (Note: in Assignment # 5, you were asked to read section 12.3. Make sure you heave read that before beginning this homework.)

2. Page 420, #3.3. Let f be a strictly convex quadratic function of one variable. Prove that the secant method for minimization will terminate in exactly one iteration for any initial start points x_0 and x_1. **Solution.** Since f is quadratic and of one variable we can write $f(x) = ax^2 + bx + c$ for some scalars (numbers) a, b, c and $a \neq 0$. Since f is strictly convex we also know that $a > 0$. Next, since we have $f(x)$ we can write $f'(x) = 2ax + b$. To minimize the function, note that $f'(x) = 2ax + b$, so if we set $0 = f'(x_*) = 2ax_* + b$, then $x_* = \frac{-b}{2a}$ where x_* is the minimizer. (Note that it is easy to find the minimizer by basic algebra, so we are doing this exercise mainly to test if the secant method behaves like we expect it to behave.) Recall the secant method in one dimension, given by:

$$x_{k+1} = x_k - \frac{(x_k - x_{k-1})}{f'(x_k) - f'(x_{k-1})} f'(x_k)$$

Using this and $f(x) = ax^2 + bx + c$ in the secant method with $k = 1$, we find:

$$\begin{aligned}
x_2 &= x_1 - \frac{(x_1 - x_0)}{f'(x_1) - f'(x_0)} f'(x_1) \\
&= x_1 - \frac{(x_1 - x_0)}{(2ax_1 + b) - (2ax_0 + b)} (2ax_1 + b) \\
&= x_1 - \frac{(x_1 - x_0)}{2ax_1 - 2ax_0} (2ax_1 + b) \\
&= x_1 - \frac{\cancel{(x_1 - x_0)}}{2a\cancel{(x_1 - x_0)}} (2ax_1 + b) \\
&= x_1 - \frac{1}{2a} (2ax_1 + b) \\
&= x_1 - x_1 - \frac{b}{2a} \\
&= \frac{-b}{2a}.
\end{aligned}$$

Note that exactly the same result would hold if we looked for x_3, x_4, etc. Thus, the sequences is constant after x_1. Moreover, since the first iteration finding x_2 is equal to $x_* = \frac{-b}{2a}$ where x_* is the minimizer, x_2 is the minimizer and the secant method terminated after one iteration.

3. Page 421, #3.7. Let B_{k+1} be obtained from B_k using the update formula

$$B_{k+1} = B_k + \frac{(y_k - B_k s_k) v^T}{v^T s_k}$$

where v is a vector such that $v^T s_k \neq 0$. Prove that $B_{k+1} s_k = y_k$.

Proof.

$$\begin{aligned}
B_{k+1} &= B_k + \frac{(y_k - B_k s_k) v^T}{v^T s_k} \\
&= B_k + \frac{(y_k - B_k s_k)}{s_k} \\
&= \frac{B_k s_k + y_k - B_k s_k}{s_k} \\
&= \frac{y_k}{s_k} \\
B_{k+1} s_k &= y_k
\end{aligned}$$

□

4. Page 82, #1.3. Consider the system of inequality constraints $Ax \geq b$ with

$$A = \begin{pmatrix} 9 & 4 & 1 & 9 & -7 \\ 6 & -7 & 8 & -4 & -6 \\ 1 & 6 & 3 & -7 & 6 \end{pmatrix} \text{ and } b = \begin{pmatrix} -15 \\ -30 \\ -20 \end{pmatrix}$$

For the given values of x and p, perform a ratio test to determine the maximum step length $\overline{\alpha}$ such that $x + \overline{\alpha} p$ remains feasible.

Solution.

We want to find the a_is that satisfy $a_i^T p < 0$. Then we can use the ratio test to find the smallest of such a_is to determine the value of $\overline{\alpha}$. Let

$$a_1^T = (9, 4, 1, 9, -7) \text{ and } b_1 = -15$$
$$a_2^T = (6, -7, 8, -4, -6) \text{ and } b_2 = -30$$
$$a_3^T = (1, 6, 3, -7, 6) \text{ and } b_3 = -20$$

(i) $x = (8, 4, -3, 4, 1)^T$ and $p = (1, 1, 1, 1, 1)^T$

$$a_1^T p = 16 \not< 0 \qquad a_2^T p = -3 < 0 \qquad a_3^T p = 9 \not< 0$$

$$\overline{\alpha} = \min\{\frac{a_2^T x - b_2}{(-a_2^T p)}\} = \min\{\frac{-26 - (-30)}{-(-3)}\} = \frac{4}{3}$$

So the maximum step length will occur when $\overline{\alpha} = \frac{4}{3}$.

(ii) $x = (7, -4, -3, -3, 3)^T)$ and $p = (3, 2, 0, 1, -2)^T$

$$a_1^T p = 58 \not< 0 \qquad a_2^T p = 12 \not< 0 \qquad a_3^T p = -4 < 0$$

$$\overline{\alpha} = \min\{\frac{a_3^T x - b_3}{(-a_3^T p)}\} = \min\{\frac{13 - (-20)}{-(-4)}\} = \frac{33}{4}$$

So the maximum step length will occur when $\overline{\alpha} = \frac{33}{4}$.

(iii) $x = (5, 0, -6, -8, -3)^T)$ and $p = (5, 0, 5, 1, 3)^T$

$$a_1^T p = 38 \not< 0 \qquad a_2^T p = 48 \not< 0 \qquad a_3^T p = 31 \not< 0$$

Since $a_i^T p \geq 0$ for all a_i, the constraint will remain satisfied for any $\alpha \geq 0$.

(iv) $x = (9, 1, -1, 6, 3)^T)$ and $p = (-4, -2, 4, -2, 2)^T$

$$a_1^T p = -72 < 0 \qquad a_2^T p = 18 \not< 0 \qquad a_3^T p = 22 \not< 0$$

$$\overline{\alpha} = \min\{\frac{a_1^T x - b_1}{(-a_1^T p)}\} = \min\{\frac{117 - (-15)}{-(-72)}\} = \frac{11}{6}$$

So the maximum step length will occur when $\overline{\alpha} = \frac{11}{6}$.

5. Page 84-85, #2.1 (i) and (iii). In each of the following cases, compute a basis matrix for the null space of the matrix and express the points x_i as $x_i = p_i + q_i$ where p_i is in the null space of A and q_i is in the range space of A^T.

Solution.

(i) $A = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & -1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}$, $x_1 = \begin{pmatrix} 1 \\ 3 \\ 1 \\ 2 \end{pmatrix}$, $x_2 = \begin{pmatrix} 0 \\ -2 \\ -3 \\ 4 \end{pmatrix}$

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & -1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix} \sim A = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 0 & 1 \\ 0 & 0 & 1 & -1 \end{pmatrix} \Rightarrow \text{basis for Nul}(A) = Z = \begin{pmatrix} -1 \\ -1 \\ 1 \\ 1 \end{pmatrix}$$

For this problem we'll find $\vec{p}$ via the equation $\vec{p} = (I - A^T(AA^T)^{-1}A)\vec{x}$ and then $\vec{q}$ via $\vec{q} = \vec{x} - \vec{p}$.

$$A^T = \begin{pmatrix} 1 & 1 & 0 \\ 1 & -1 & 1 \\ 1 & -1 & 0 \\ 1 & 1 & 1 \end{pmatrix}$$

$$AA^T = \begin{pmatrix} 4 & 0 & 2 \\ 0 & 4 & 0 \\ 2 & 0 & 2 \end{pmatrix}$$

$$(AA^T)^{-1} = \begin{pmatrix} \frac{1}{2} & 0 & -\frac{1}{2} \\ 0 & \frac{1}{4} & 0 \\ -\frac{1}{2} & 0 & 1 \end{pmatrix}$$

$$A^T(AA^T)^{-1} = \begin{pmatrix} \frac{1}{2} & \frac{1}{4} & -\frac{1}{2} \\ 0 & -\frac{1}{4} & \frac{1}{2} \\ \frac{1}{2} & -\frac{1}{4} & -\frac{1}{2} \\ 0 & \frac{1}{4} & \frac{1}{2} \end{pmatrix}$$

$$A^T(AA^T)^{-1}A = \begin{pmatrix} \frac{3}{4} & -\frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ -\frac{1}{4} & \frac{3}{4} & \frac{1}{4} & \frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} & \frac{3}{4} & -\frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} & -\frac{1}{4} & \frac{3}{4} \end{pmatrix}$$

$$I - (A^T(AA^T)^{-1}A) = \begin{pmatrix} \frac{1}{4} & \frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} \\ \frac{1}{4} & \frac{1}{4} & -\frac{1}{4} & -\frac{1}{4} \\ -\frac{1}{4} & -\frac{1}{4} & \frac{1}{4} & \frac{1}{4} \\ -\frac{1}{4} & -\frac{1}{4} & \frac{1}{4} & \frac{1}{4} \end{pmatrix}$$

$$\vec{p}_1 = I - (A^T(AA^T)^{-1}A)\vec{x}_1 = \begin{pmatrix} \frac{1}{4} \\ \frac{1}{4} \\ -\frac{1}{4} \\ -\frac{1}{4} \end{pmatrix} \quad \text{and thus, } \vec{q}_1 = \vec{x}_1 - \vec{p}_1 = \begin{pmatrix} 1 \\ 3 \\ 1 \\ 2 \end{pmatrix} - \begin{pmatrix} \frac{1}{4} \\ \frac{1}{4} \\ -\frac{1}{4} \\ -\frac{1}{4} \end{pmatrix} = \begin{pmatrix} \frac{3}{4} \\ \frac{11}{4} \\ \frac{5}{4} \\ \frac{9}{4} \end{pmatrix}$$

$$\vec{p}_2 = I - (A^T(AA^T)^{-1}A)\vec{x}_2 = \begin{pmatrix} -\frac{3}{4} \\ -\frac{3}{4} \\ \frac{3}{4} \\ \frac{3}{4} \end{pmatrix}$$

and thus,

$$\vec{q}_2 = \vec{x}_2 - \vec{p}_2 = \begin{pmatrix} 0 \\ -2 \\ -3 \\ 4 \end{pmatrix} - \begin{pmatrix} -\frac{3}{4} \\ -\frac{3}{4} \\ \frac{3}{4} \\ \frac{3}{4} \end{pmatrix} = \begin{pmatrix} \frac{3}{4} \\ -\frac{5}{4} \\ -\frac{15}{4} \\ \frac{13}{4} \end{pmatrix}$$

(iii) $A = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & -1 & 1 \end{pmatrix}$, $x_1 = \begin{pmatrix} 4 \\ 3 \\ 4 \\ 0 \end{pmatrix}$, $x_2 = \begin{pmatrix} -1 \\ 1 \\ 5 \\ -5 \end{pmatrix}$

$$A = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & -1 & 1 \end{pmatrix} \sim A = \begin{pmatrix} 1 & 0 & 0 & 1 \\ 0 & 1 & 1 & 0 \end{pmatrix} \Rightarrow$$

basis for $\text{Nul}(A) = Z = \begin{pmatrix} 0 & -1 \\ -1 & 0 \\ 1 & 0 \\ 0 & 1 \end{pmatrix}$

For this method we'l find $\vec{p}$ by orthogonalizing the columns of Z and projecting the basis. Let $\hat{z}_i$ be the i^{th} column vector of Z.

$$\vec{p}_1 = \frac{\vec{x}_1 \cdot \hat{z}_1}{\|\hat{z}_1\|^2}\hat{z}_1 + \frac{\vec{x}_1 \cdot \hat{z}_2}{\|\hat{z}_2\|^2}\hat{z}_2 = \frac{1}{2}\hat{z}_1 + -\frac{4}{2}\hat{z}_2 = \begin{pmatrix} 0 \\ -\frac{1}{2} \\ \frac{1}{2} \\ 0 \end{pmatrix} + \begin{pmatrix} 2 \\ 0 \\ 0 \\ -2 \end{pmatrix} = \begin{pmatrix} 2 \\ -\frac{1}{2} \\ \frac{1}{2} \\ -2 \end{pmatrix} \text{ and thus,}$$

$$\vec{q}_1 = \vec{x}_1 - \vec{p}_1 = \begin{pmatrix} 4 \\ 3 \\ 4 \\ 0 \end{pmatrix} - \begin{pmatrix} 2 \\ -\frac{1}{2} \\ \frac{1}{2} \\ -2 \end{pmatrix} = \begin{pmatrix} 2 \\ \frac{7}{2} \\ \frac{7}{2} \\ 2 \end{pmatrix}$$

$$\vec{p}_2 = \frac{\vec{x}_2 \cdot \hat{z}_1}{\|\hat{z}_1\|^2}\hat{z}_1 + \frac{\vec{x}_2 \cdot \hat{z}_2}{\|\hat{z}_2\|^2}\hat{z}_2 = \frac{4}{2}\hat{z}_1 + -\frac{4}{2}\hat{z}_2 = \begin{pmatrix} 0 \\ -2 \\ 2 \\ 0 \end{pmatrix} + \begin{pmatrix} 2 \\ 0 \\ 0 \\ -2 \end{pmatrix} = \begin{pmatrix} 2 \\ -2 \\ 2 \\ -2 \end{pmatrix} \text{ and thus,}$$

$$\vec{q}_2 = \vec{x}_2 - \vec{p}_2 = \begin{pmatrix} -1 \\ 1 \\ 5 \\ -5 \end{pmatrix} - \begin{pmatrix} 2 \\ -2 \\ 2 \\ -2 \end{pmatrix} = \vec{q}_2 = \begin{pmatrix} -3 \\ 3 \\ 3 \\ -3 \end{pmatrix}$$

6. Page 85, #2.6. Suppose that you are given a matrix A and a vector p and are told that p is in the null space of A. On a computer, you cannot expect that Ap will be exactly equal to zero because of rounding errors. How large would the computed value of $\|Ap\|$ have to be before you could conclude that p was not in the null space of A? If the computed value of $\|Ap\|$ is zero, can you conclude that p is in the null space of A? (Note: This probably is somewhat more open-ended.)

   Solution. Answers may vary widely. The key is to remember that in a computer, when numbers get very small (very close to zero), they are indistinguishable from zero. Try some test matrices in Matlab to see if you can come up with some ideas!

11.2.8 Homework 7

1. Read sections 14.1, 14.2, 14.3, and 14.5 in the book.

2. *Problem 2.2(i) on page 489.* Determine the minimizers/maximizers of the following functions subject to the given constraints.

$$f(x_1, x_2) = x_1 x_2^3 \text{ subject to } 2x_1 + 3x_2 = 4.$$

   Solution. For this problem we will use the $x = \bar{x} + Zv$ method and minimize $\phi(v)$. So we will need to

determine an $\bar{x}$, find Z, and determine values for v.

$$A = \begin{pmatrix} 2 & 3 \end{pmatrix} \sim \begin{pmatrix} 1 & \frac{3}{2} \end{pmatrix} \text{ and thus,}$$
$$Z = \begin{pmatrix} -\frac{3}{2} & 1 \end{pmatrix}^T.$$
$$\text{Next, let } \bar{x} = \begin{pmatrix} 2 & 0 \end{pmatrix}^T \text{ (it is a solution to } Ax = b\text{). Then,}$$
$$\begin{pmatrix} x_1 \\ x_2 \end{pmatrix} = x = \bar{x} + Zv = \begin{pmatrix} 2 \\ 0 \end{pmatrix} + \begin{pmatrix} -\frac{3}{2} \\ 1 \end{pmatrix} v = \begin{pmatrix} 2 - \frac{3}{2}v \\ v \end{pmatrix}$$
$$\text{Note that } v \text{ is a scalar. Plugging } x \text{ into } f \text{ gives}$$
$$\phi(v) = f(\bar{x} + Zv) = (2 - \frac{3}{2}v)(v)^3 = 2v^3 - \frac{3}{2}v^4.$$

Now, we have reduced the constrained optimization problem to an unconstrained optimization problem. We simply take the derivative and set it equal to zero, like in calculus. For a more general problem , we might need to use an iterative solver like Newton's method or BFGS, but this problem is nice enough that we can solve it with algebra.

$$0 \overset{\text{set}}{=} \nabla\phi(v) = 6v^2 - 6v^3 \Rightarrow 6v^2 = 6v^3 \Rightarrow v = 0 \text{ or } v = 1.$$

From here we can see that when $v = 0$ we will get $x_1 = 2$ and $x_2 = 0$. On the other hand when $v = 1$ we will get $x_1 = \frac{1}{2}$ and $x_2 = 1$. Next, we'll use these values to determine the type of stationary points these values may be.

$$\nabla f(x) = \begin{pmatrix} x_2^3 \\ 3x_1x_2^2 \end{pmatrix} \text{ and } \nabla^2 f(x) = \begin{pmatrix} 0 & 3x_2^2 \\ 3x_2^2 & 6x_1x_2 \end{pmatrix}$$

Next, we'll use Lemma 14.2 to check the stationary point $x_* = \begin{pmatrix} 2 & 0 \end{pmatrix}^T$. So we need to check that $Z^T\nabla f(x_*) = 0$ and $Z^T\nabla^2 f(x_*)Z$ is positive definite.

$$Z^T\nabla f(x_*) = \begin{pmatrix} -\frac{3}{2} & 1 \end{pmatrix} \begin{pmatrix} 0 \\ 0 \end{pmatrix} = 0$$
$$Z^T\nabla^2 f(x_*)Z = \begin{pmatrix} -\frac{3}{2} & 1 \end{pmatrix} \begin{pmatrix} 0 & 0 \\ 0 & 0 \end{pmatrix} \begin{pmatrix} -\frac{3}{2} \\ 1 \end{pmatrix} = 0$$

Thus, the reduced Hessian is only semidefinite, and we cannot conclude that $x_* = (2, 0)$ is a local minimizer. (Lemma 14.2 only gives a necessary, but not sufficient condition.)

Next, we'll use Lemma 14.3 to check the stationary point $x_* = \begin{pmatrix} \frac{1}{2} & 1 \end{pmatrix}^T$. For Lemma 14.3 we need to check that $Ax_* = b$, $Z^T\nabla f(x_*) = 0$, and $Z^T\nabla^2 f(x_*)Z$ is positive definite.

$$Ax_* = \begin{pmatrix} 2 & 3 \end{pmatrix} \begin{pmatrix} \frac{1}{2} \\ 1 \end{pmatrix} = \begin{pmatrix} 4 \end{pmatrix} = b$$
$$Z^T\nabla f(x_*) = \begin{pmatrix} -\frac{3}{2} & 1 \end{pmatrix} \begin{pmatrix} 1 \\ \frac{3}{2} \end{pmatrix} = 0$$
$$Z^T\nabla^2 f(x_*)Z = \begin{pmatrix} -\frac{3}{2} & 1 \end{pmatrix} \begin{pmatrix} 0 & 3 \\ 3 & 3 \end{pmatrix} \begin{pmatrix} -\frac{3}{2} \\ 1 \end{pmatrix} = -6$$

Although we can see that $Z^T\nabla^2 f(x_*)Z$ is not positive definite, it is negative definite which implies that $x_* = \begin{pmatrix} \frac{1}{2} & 1 \end{pmatrix}^T$ is a strict local maximizer of f.

3. *Problem 2.2(vii) on page 490.* Determine the minimizers/maximizers of the following functions subject to the given constraints.

$$f(x_1, x_2) = \frac{1}{3}x_1^3 + x_2 \text{ subject to } x_1^2 + x_2^2 = 1$$

Solution. This problem has a nonlinear constraint, so the method used in the previous problem does not apply. Instead, we will use the Lagrangian function in its nonlinear form, namely, $\mathcal{L}(x, \lambda) = f(x) - \lambda^T \nabla g(x)$ (note that since we only have one constraint in this problem, λ is a scalar, so $\lambda^T = \lambda$).

$$\mathcal{L}(x_1, x_2, \lambda) = \frac{1}{3}x_1^3 + x_2 - \lambda(x_1^2 + x_2^2 - 1) = \frac{1}{3}x_1^3 + x_2 - \lambda x_1^2 - \lambda x_2^2 + \lambda$$

Our task is to find x_* and λ_* satisfying $\nabla\mathcal{L}(x_*, \lambda_*) = 0$. Thus, we proceed as follows.

$$\vec{0} \overset{\text{set}}{=} \nabla\mathcal{L} = \begin{pmatrix} x_1^2 - 2\lambda x_1 \\ 1 - 2\lambda x_2 \\ -x_1^2 - x_2^2 + 1 \end{pmatrix} \Rightarrow \begin{array}{c} x_1^2 = 2\lambda x_1 \\ 1 = 2\lambda x_2 \\ 1 = x_1^2 + x_2^2 \end{array}$$

Here we can see we have a few choices that we can make. First, **assuming x_1 is not zero**, we can solve for x_1 and x_2 in terms of λ to arrive at:

$$\begin{gathered} x_1 = 2\lambda \\ x_2 = \frac{1}{2\lambda} \\ (2\lambda)^2 + \left(\frac{1}{2\lambda}\right)^2 = 1 \;\Rightarrow\; 4\lambda^2 + \frac{1}{4\lambda^2} = 1 \;\Rightarrow\; 16\lambda^4 - 4\lambda^2 + 1 = 0. \end{gathered}$$

Note that in general, solving a 4^{th} degree polynomial equation can be hard. However, there is a special form to this polynomial. Namely, it is 2^{nd} degree in λ^2:

$$\begin{aligned} 0 &= 16\lambda^4 - 4\lambda^2 + 1 \\ &= 16(\lambda^2)^2 - 4(\lambda^2) + 1 \end{aligned}$$

This means we can use the quadratic formula to solve for λ^2!

$$\lambda^2 = \frac{4 \pm \sqrt{4^2 - 4 \cdot 16 \cdot 1}}{2 \cdot 16} = \frac{4 \pm \sqrt{16 - 64}}{32}.$$

We have a negative under the square root, so there are no real solutions for λ. Moreover, since $x_1 = 2\lambda$ and $x_2 = \frac{1}{2\lambda}$, if λ is complex, x_1 and x_2 will be complex as well, but they are assumed to be real, so we must choose different values. Thus, the only possibility is that $x_1 = 0$.

Thus, we assume $x_1 = 0$ and similarly from above we can solve for λ and x_2 as follows.

$$\begin{gathered} x_1 = 0 \\ 1 = (0)^2 + (x_2)^2 \;\Rightarrow\; x_2 = \pm 1 \quad (\textit{from constraint}) \\ 1 = 2\lambda(\pm 1) \;\Rightarrow\; \lambda = \pm\frac{1}{2} \end{gathered}$$

Thus, we can check these two possibilities $x_1 = 0$, $x_2 = 1$, and $\lambda = \frac{1}{2}$ and $x_1 = 0$, $x_2 = -1$, and $\lambda = -\frac{1}{2}$. We will use Theorem 14.16, so we'll need to check that $\nabla_x\mathcal{L}(x_*, \lambda_*) = 0$ and $Z(x_*)^T \nabla^2_{xx}\mathcal{L}(x_*, \lambda_*)Z(x_*)$

is positive definite. First, we'll consider the first stationary point $x_* = \begin{pmatrix} 0 & 1 \end{pmatrix}^T$ and $\lambda_* = \frac{1}{2}$.

$$\nabla g(x) = \begin{pmatrix} 2x_1 \\ 2x_2 \end{pmatrix} \quad \text{and} \quad \nabla g(x_*) = \begin{pmatrix} 0 \\ 2 \end{pmatrix} \quad \text{so let} \quad Z = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$\nabla^2_{xx}\mathcal{L}(x,\lambda) = \begin{pmatrix} 2x_1 & 0 \\ 0 & 0 \end{pmatrix} - \lambda \begin{pmatrix} 2 & 0 \\ 0 & 2 \end{pmatrix} = \begin{pmatrix} 2x_1 - 2\lambda & 0 \\ 0 & -2\lambda \end{pmatrix}$$

$$\text{so} \quad \nabla^2_{xx}\mathcal{L}(x_*,\lambda_*) = \begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix}$$

$$Z(x_*)^T\nabla^2_{xx}\mathcal{L}(x_*,\lambda_*)Z(x_*) \;\Rightarrow\; \begin{pmatrix} 1 & 0 \end{pmatrix}\begin{pmatrix} -1 & 0 \\ 0 & -1 \end{pmatrix}\begin{pmatrix} 1 \\ 0 \end{pmatrix} = -1$$

Although $Z(x_*)^T\nabla^2_{xx}\mathcal{L}(x_*,\lambda_*)Z(x_*)$ is not positive definite, it is negative definite and thus $x_* = \begin{pmatrix} 0 & 1 \end{pmatrix}^T$ is a local maximizer of f. Finally, we'll consider the other stationary point $x_* = \begin{pmatrix} 0 & -1 \end{pmatrix}^T$ and $\lambda_* = -\frac{1}{2}$.

$$\nabla g(x_*) = \begin{pmatrix} 0 \\ -2 \end{pmatrix} \quad \text{so let} \quad Z = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$$

$$\nabla^2_{xx}\mathcal{L}(x_*,\lambda_*) = \begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}$$

$$Z(x_*)^T\nabla^2_{xx}\mathcal{L}(x_*,\lambda_*)Z(x_*) \;\Rightarrow\; \begin{pmatrix} 1 & 0 \end{pmatrix}\begin{pmatrix} 1 & 0 \\ 0 & 1 \end{pmatrix}\begin{pmatrix} 1 \\ 0 \end{pmatrix} = 1$$

The stationary point $x_* = \begin{pmatrix} 0 & -1 \end{pmatrix}^T$ and associated $\lambda_* = -\frac{1}{2}$ satisfies all the conditions for Theorem 14.16 and thus x_* is a strict local minimizer of f.

4. *Problem 5.2 on page 509.* Solve the problem

$$\begin{aligned} \text{minimize} \quad & f(x) = c^T x \\ \text{subject to} \quad & \sum_{i=1}^n x_i = 0 \\ & \sum_{i=1}^n x_1^2 = 1 \end{aligned}$$

Solution. We solve this problem using the Lagrangian method. First, we'll make the constraints easier to use in the function by defining them in a different way.

$$\text{Let} \quad w = \begin{pmatrix} 1 & 1 & \cdots & 1 \end{pmatrix}^T.$$

$$\text{Note that:} \quad w^T x = \sum_{i=1}^n x_i = 0$$

$$\text{and} \quad x^T x = \sum_{i=1}^n x_1^2 = 1.$$

Next, we write down the Lagrangian function and set its gradient equal to zero

$$\mathcal{L} = c^T x - \lambda^T \begin{pmatrix} w^T x \\ x^T x - 1 \end{pmatrix} = c^T x - \lambda_1 w^T x - \lambda_2 x^T x + \lambda_2$$

$$0 \overset{\text{set}}{=} \nabla\mathcal{L} = c - \lambda_1 w - \lambda_2 x$$

$$\Rightarrow \; x = \frac{1}{\lambda_2}(c - \lambda_1 w)$$

We take the dot-product of the last relationship with w, x, and c, to obtain:

$$\begin{cases} w^T x = \dfrac{1}{\lambda_2}(w^T c - \lambda_1 w^T w) \\ x^T x = \dfrac{1}{\lambda_2}(x^T c - \lambda_1 x^T w) \\ c^T x = \dfrac{1}{\lambda_2}(c^T c - \lambda_1 c^T w) \end{cases} \tag{11.3}$$

From the constraints, we know that $w^T x = 0$, and $x^T x = 1$. Also, note that $c^T c = \|c\|^2$,$w^T w = n$, and $c^T w = \sum_{i=1}^n c_i = n\frac{1}{n}\sum_{i=1}^n c_i = n\bar{c}$, where $\bar{c} = \frac{1}{n}\sum_{i=1}^n c_i$ denotes the average value of the components of the vector c. Using these relations, we obtain

$$\begin{cases} 0 = \dfrac{1}{\lambda_2}(n\bar{c} - n\lambda_1) \\ 1 = \dfrac{1}{\lambda_2}x^T c \\ c^T x = \dfrac{1}{\lambda_2}(\|c\|^2 - \lambda_1 n\bar{c}) \end{cases} \tag{11.4}$$

From the first equation, we see that $\lambda_1 = \bar{c}$. Note that $x^T c = x \cdot c = c \cdot x = c^T x$, and that, from the second equation, $\lambda_2 = x^T c$. Thus, we can combine this with the third equation to obtain:

$$\begin{aligned} \lambda_2 &= \frac{1}{\lambda_2}(\|c\|^2 - \lambda_1 n\bar{c}) \\ \Rightarrow \lambda_2^2 &= \|c\|^2 - \lambda_1 n\bar{c} \\ &= \|c\|^2 - n\bar{c}^2 && \text{(using } \lambda_1 = \bar{c}\text{)} \\ &= n\left(\frac{1}{n}\sum_{i=1}^n c_i^2 - \left(\frac{1}{n}\sum_{i=1}^n c_i\right)^2\right) && \text{(by definitions and algebra)} \end{aligned}$$

Now, we would like to take a square root to find λ_2, but how do we know the right-hand side is non-negative? Note that the question of whether the right-hand side is non-negative is the question of *whether the average of the squares is greater than or equal to the square of the average.* To prove that it is (you are not required to for your homework), we can use the Cauchy-Schwarz inequality:

$$|\vec{x} \cdot \vec{y}| = \|\vec{x}\|\|\vec{y}\||\cos(\theta)| \leq \|\vec{x}\|\|\vec{y}\|$$

Letting $\vec{x} = c$ and $\vec{y} = w$, we find that

$$|n\bar{c}| = \left|c^T w\right| = |c \cdot w| \leq \|c\|\|w\| = \sqrt{n}\|c\|$$

Squaring both sides and dividing by n, we find,

$$n\bar{c}^2 \leq \|c\|^2$$

so that $\|c\|^2 - n\bar{c}^2 \geq 0$. Thus, we can indeed take a square root, and we find

$$\lambda_2 = \pm\sqrt{\|c\|^2 - n\bar{c}^2}.$$

Substituting λ_1 and λ_2 back into x, we obtain

$$x = \frac{1}{\lambda_2}(c - \lambda_1 w) = \pm \frac{c - \bar{c}w}{\sqrt{\|c\|^2 - n\bar{c}^2}}.$$

Therefore, at these two points, we find

$$f(x) = c^T x = \pm \frac{\|c\|^2 - n\bar{c}^2}{\sqrt{\|c\|^2 - n\bar{c}^2}} = \pm\sqrt{\|c\|^2 - n\bar{c}^2}$$

One point is positive, the other is negative. Therefore, we have found that the maximum is $\sqrt{\|c\|^2 - n\bar{c}^2}$ and occurs at $x = \frac{c-\bar{c}w}{\sqrt{\|c\|^2 - n\bar{c}^2}}$, and the minimum is $-\sqrt{\|c\|^2 - n\bar{c}^2}$ and occurs at $x = -\frac{c-\bar{c}w}{\sqrt{\|c\|^2 - n\bar{c}^2}}$.