Contents

Chapter 1

Following Instructions

1.1 Introduction

Computers only do what they are told and we must be very literal when programming them. The ability to read between the lines and determine what was meant rather than what was said is a skill computers lack.

During class groups will write directions for a visitor who is new to Boise to visit three landmarks. The directions should be very literal so that a computer, possibly one that may be part of a driver-less car, can follow them. Each team will swap directions and the goal is to identify the landmarks.

1.2 Individual Lab

Watch this video on what it takes to give instructions to make a peanut butter and jelly sandwich: `https://youtu.be/euFj8D1A1Kw`.

1.3 Group work

Please complete the following questions in groups of 3-4 people. Each team will consist of:

- The **manager**, responsible for coordinating the work of the team. The manager is the person who was born the furthest from Boise.
- The **spokesperson**, who will report your group's work to the rest of the class. The spokesperson is the person commuting longest to class today.
- The **scribe**, who will be responsible for writing down your team's findings. The scribe is the person who's birthdays is coming up next.
- The **timekeeper**, who will keep track of time for each exercise. The timekeeper is the person who was not assigned a role already. In case of a three-person group, the manager takes the role of a timekeeper.

One person should not have more than one role (except for the manager/timekeeper combo), so if a person who already was assigned a task meets the criteria for another one, the person who is next in line (i.e., second-longest commute) takes this role.

1. (10 min) Write directions for a visitor who is new to Boise using the starting point, destination, and required landmarks in the attached envelope. Each instruction should be simple and start on a new line, as if you are programming a computer. You may use the provided map of Boise to help you with planning the route and assume that the visitor has the same map to guide them. In your directions use the words landmark 1, landmark 2, destination, etc. and not Capitol Building, Century Link Arena, etc. The scribe will write down the directions on a separate piece of paper with the team number at the top of the page.

2. (10 min) Swap your directions with another team. Using the other team's instructions and your map, try to follow the route from the hotel to the destination point. Identify which landmarks you pass by, and what is your destination.

   Please be literal in following instructions and do not use your knowledge of Boise to identify a landmark. If a step in the other group's instructions seems impossible or requires additional explanation, proceed by attempting to complete the instruction in an obtuse manner.

3. (5 min) Once the answers are posted, identify whether you got the destination and landmarks correctly. Reflect on the following

   - Which part of the exercise did you find to be the easiest?
   - How was this activity similar/different to following an algorithm?
   - What characteristics of a good algorithm did your directions have (the ones you created and the ones you followed)?
   - What characteristics of a good algorithm did your directions not have that may have helped?

   The spokesperson should be prepared to report your reflections to the rest of the class.

Chapter 2

Loops and Conditional Statements

2.1 Introduction

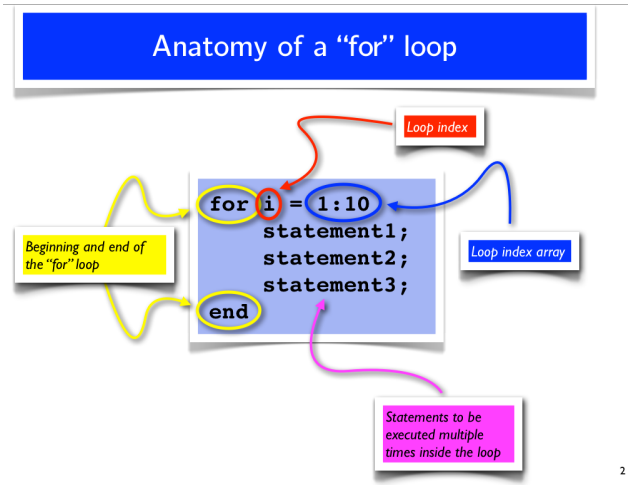
Every procedural programming language has conditional statements (if-statements) and iterative statements (loops). You should be familiar with the concepts so here we will only review them in order to understand how to control order of execution.

2.1.1 Loops

The **for loop** is a type of flow control statement that allows us to execute a set of commands multiple times. For example, in the following code fragment the statement *Hello World* gets executed 10 times, even though it only appears once in the code fragment.

```
for i = 1:10
    print("Hello, World!")
end
```

The for loop has three basic components: (i) the enclosing keywords *for* and *end* which enclose the set of statements to be carried out multiple times by the loop (the statement print("Hello, World!"), in the above example), (ii) a loop index which is the variable that stores the current value of the loop counter (i in the above example), and (iii) an index array containing the range of values over which the loop index should vary (the array 1:10 in the above example). See below for the Anatomy of a "for" loop.



Here's another example of a for loop

```
for count = 10:-5:1,
    print(count)
end
```

and it will print *10 5*.

The **while loop** is used to repeat a segment of code an unknown number of times until a specific condition is met. For example, say we want to know how many times a given number can be divided by 2 before it is less than or equal to 1. If we know a specific number, such as 32, we can say 5 times, but for a given symbolic variable NUMBER which represents any number in the world, how many times is not known a priori (before hand). In this case, we could use a while loop to determine that answer:

```
count = 0
while NUMBER > 1
    NUMBER = NUMBER /2
    count = count + 1
end
```

WARNING: If the action inside the loop does not modify the variables being tested in the loops condition, the loop will "run" forever. For example

```
while y < 10
    x = x + 1
end
```

is an infinite loop.

The **break statement** terminates execution of **for or while loops**. Statements in the loop that appear after the break statement are not executed. For example

```
fruits = ["apple", "banana", "cherry"]
for x in fruits
```

```
    if x == "banana"
        break
    end
    print(x)
end
```

will print *apple*. Pretend you are computer to see why this is the result!

The **continue statement** is used for passing control to the next iteration of a for or while loop. For example

```
fruits = ["apple", "banana", "cherry"]
for x in fruits
    if x == "banana"
        continue
    end
    print(x)
end
```

will print *apple cherry*. Again, pretend you are a computer so that you understand the logic of the conditional and iterative statements.

2.2 Individual Lab

Please answer the following questions in Blackboard.

1. How many times does the following loop print the statement *Three brown bears*?

```
for i = 1:10
    print("Three brown bears")
end
```

2. What is printed with the following statements?

```
fruits = ["apple", "banana", "cherry"]
for x in fruits
    print(x)
    if x == "banana"
        break
    end
end
```

3. What will print with the following statements?

```
for j = 2:3:5
    print(j)
end
```

2.3 Group work

Please complete the following questions in groups of 3-4 people. At the end of class, hand in your team's findings with each team member's name clearly written at the top of the page. Each team will consist of:

- The **manager**, responsible for coordinating the work of the team. The manager is the person who sitting furthest from the door.
- The **spokesperson**, who will report your group's work to the rest of the class. The spokesperson is the person who ate the healthiest breakfast.
- The **scribe**, who will be responsible for writing down your team's findings. The scribe is the person who is taking the fewest credit hours this semester.
- The **timekeeper**, who will keep track of time for each exercise. The timekeeper is the person who was not assigned a role already. In case of a three-person group, the manager takes the role of a timekeeper.

One person should not have more than one role (except for the manager/timekeeper combo), so if a person who already was assigned a task meets the criteria for another one, the person who is next in line (i.e., second-longest commute) takes this role.

1. (5 min)

   (a) Consider the pseudocode

   ```
   x = 4
   s = 0
   for i = 1:3
       s = s + x
   end
   ```

   What is the resulting value of s?

   (b) Consider the following pseudocode:

   ```
   x = 4
   s = 0
   for i = 1:n
       s = s + x
   end
   ```

   i. If you programmed it, there would be an error message. Why?

   ii. Since you cannot successfully program it, write down the sequence of values of s defined by the for loop. Once you identify a pattern in the sequence, write down a mathematical expression for s.

   The spokesperson should be prepared to report your answers written by the scribe to the rest of the class.

2. (5 min) Consider the following pseudocode:

```
x = [0.2  0.15  0.004  0.55  0.77]
findx = 100
for i = 1:5
    if x(i) < findx
        findx = x(i)
    end
end
```

(a) What is the resulting value of findx?

(b) How could you alter this code to find the maximum value in the array (vector) x?

The spokesperson should be prepared to report your answers written by the scribe to the rest of the class.

3. (10 min) Consider the following pseudocodes:

(a)

```
yfor = 1
for i = 1:n
    yfor = yfor*x(i)
end
```

i. Identify two reasons why coding this would result in an error message.

ii. Even though we can't code it, write the sequence defined by the for loop by hand. Identify the pattern in this sequence and write down the mathematical expression for the sequence using $\prod$ notation.

(b)

```
s = 0
for i = 1:n
    s = s + x(i)
end
mys = s/n
```

i. Again we cannot successfully code this, but write the sequence defined by the for loop by hand and identify the mathematical expression for the sequence using Σ notation.

(c) ii. Identify what mys represents.

```
mu = 0
for i = 1:n
    mu = mu + x(i)
end
mu = mu/n
s=0
for i = 1:n
    s = s + (x(i)-mu)^2
end
s = sqrt(s/(n-1))
```

i. Again we cannot successfully code this, but write the sequence defined by the second for loop by hand. Identify the mathematical expression for this sequence using Σ notation.

ii. Identify what s represents.

The spokesperson should be prepared to report your answers written by the scribe to the rest of the class.

4. (10 min) This pseudocode is a bit more challenging.

```
for j = 1:n
    s = 1
    for i = 1:j
        s = s*x(i)
    end
    yarray(j) = s
end
```

   (a) Identify the reasons why we would get error messages.

   (b) Notice that yarray if formed with two for loops and is an array, or vector. Consider only one iteration of the outer loop j, i.e. j=1, and write the sequence defined by the inner for loop by hand. Identify the mathematical expression for yarray(1).

   (c) Now consider the second iteration in the outer loop, i.e. j=2, and identify the mathematical expression for yarray(2).

   (d) Now consider the third iteration in the outer loop, i.e. j=3, and identify the mathematical expression for yarray(3) using $\prod$ notation.

   (e) Identify the mathematical expression of the sequence formed by the inner for loop using $\prod$ notation for any value of j.

   (f) Explain how your answer to 4e is different than your answer to 3(a)ii

The spokesperson should be prepared to report your answers written by the scribe to the rest of the class.

Chapter 3

Python and Jupyter Notebooks

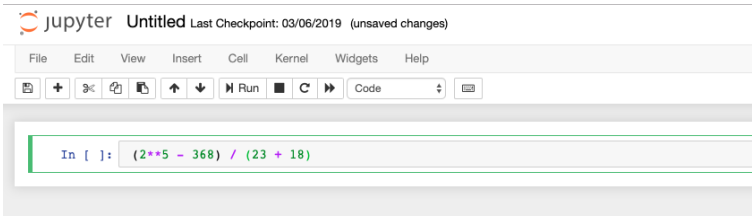
3.1 Introduction

3.1.1 Jupyter Notebooks

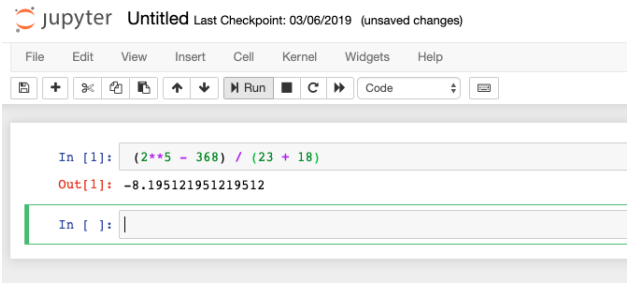
Python is a programming language and Jupyter is an interactive computational environment where you can combine code, text and graphs. A Jupyter notebook consists of a sequence of cells of different types. In a code cell you can edit and write Python code. A code cell has an input section containing your code and output section after executing the cell. You can execute or run a cell by

- clicking the Run button in the tool bar,
- selecting Cell → Run Cells in the menu bar,
- pressing Shift-Enter

For example, to compute $\frac{2^5-365}{23+18}$ you would type



and once you run it you would see



Note that the prompt numbers next to the code cells (e.g. In [1] and Out [1]) indicate which cells have been run and in which order. This is very useful, especially if you are running cells out-of-sequence. Note that not all operations have outputs, for example, print statements. This is because the print statement doesn't affect the output. If you would like to see a demonstration of someone using Jupyter notebooks I recommend watching this youtube video between 6:00 and 15:00 minutes: `https://youtu.be/CwFq3YDU6_Y`.

Jupyter notebooks are available in MB 136 and you may also choose to install it on your own computer. I suggest using the Anaconda distribution to install Python and Jupyter. Go to `https://www.anaconda.com`, choose to download the most recent version of Python, and follow the instructions.

3.1.2 Python Syntax

Python syntax may be different than our pseudocode in the Loops and Conditionals lab. The first significant difference is that in our pseudocode we had an *end* statement to terminate the *if*, *for* and *while* loops. Alternatively, Python uses indentation to indicate if commands sit inside a loop. The : symbol is used to start the indent suite of statements. For example, in Python we would write a while loop in the following manner

```
x = 0
while x < 4:
    x = x + 1
```

This code segment would assign value of 4 to x (justify this to yourself) and you would verify this by running

```
print(x)
```

Another significant programming difference in Python as compared to our pseudocode in the Loops and Conditionals lab is how we wrote *for loops* over consecutive integers. In Python we would do this with the range() function.

- range(n) is equivalent to to the list [0, 1, ..., n - 1]
- range(start, stop) is equivalent to the list [start, start + 1, ..., stop - 1]

Python has both lists and arrays. In this class we will mostly use arrays but we need the package NumPy to define arrays, which we will do in the next section.

If we wanted to print the first 10 integers, starting at 0 we could run

```
for i in range(10):
    print(i)
```

which is equivalent to

```
for i in range(0,10):
    print(i)
```

and equivalent to

```
list=[0,1,2,3,4,5,6,7,8,9]
for i in list:
    print(i)
```

This would produce output 0 1 2 3 4 5 6 7 8 9 since we started at 0.

We can also increment the loop and just print the even integers

```
x = range(0, 10, 2)
for n in x:
    print(n)
```

It is important to note that range() can take only integers as arguments. To iterate over more general floating point numbers we need the package NumPy.

3.1.3 NumPy

NumPy (numerical Python, pronounced num-pie) is a library that provides advanced mathematical operations involving statistics and linear algebra. Numpy's standard data type is an array and right now that is our main interest in using it.

NumPy is one of many modules in Python. Modules are pre-written collections of operators, usually designed for a specific purpose or task. To use a module, you must first import it, e.g.

```
import numpy
```

To invoke one of it's operations, precede the operator's name with the name of the module, followed by a period. For example, if we want to use an array of floating point numbers rather than a list of integers

```
arr = numpy.array( [ 2.1, 3.7, 4.2 ] )
```

It can be a bit tedious to write numpy repeadly so typically a pseudonym is used when we import it, e.g.

```
import numpy as np
```

so that we can write

```
arr=np.array( [ 2.1, 3.7, 4.2 ] )
```

There are numerous NumPy operations e.g.

```
arr = np.zeros( 5 )
print(arr)
```

will print `[0. 0. 0. 0. 0.]` while

```
arr = np.arange( 10, 30, 5 )
print(arr)
```

will print `[10 15 20 25]`.

Numpy provides access to elements of an array using the standard indexing operator [ ], e.g. in the last example `arr[2]` is 20 (note that counting starts at 0). It's also possible to ask for the shape of an array using numpy.shape. In the last example

print(np.shape(arr))

will output `(4,)` which means that arr is a row vector with 4 elements. The shape can also be found with the command

print(arr.shape)

Please read `https://www.pluralsight.com/guides/overview-basic-numpy-operations` for more examples. As you read these examples, for those who are more comfortable with matrix algebra than programming, think of 1D arrays as vectors and 2D arrays as matrices. This introduction is showing you how to do matrix algebra on a computer.

3.1.4 Plotting

Another useful Python package is matplotlib, which is a library for constructing plots. One module within this larger library is pyplot, which presents a simple and easy to use interface for constructing plots.

import matplotlib.pyplot as plt

Please read the Pyplot tutorial `https://matplotlib.org/users/pyplot_tutorial.html`.

3.2 Individual Lab

Part 1:
Please answer the following questions in Blackboard after reading Sections 1.1 and 1.2.

1. True or False: The pseudocode in the Loops and Conditionals Lab can be programmed in Python exactly at it is written in the Lab.
2. What will happen if we run the following statements in Python

```
sum = 0
for d in range(0, 10, 0.1):
    sum = sum + d
```

3. Which of the following commands returns a sequence 0, 1, 2, 3?

   (a) range(0, 3)
   (b) range(0, 4)
   (c) range(3)

(d) range(4)

4. What is the output for y?

(a)
```
y = 0
for i in range(0, 10, 2):
    y = y+i
print(y)
```

(b)
```
y = 0
for i in range(10, 1, -2):
    y = y+i
print(y)
```

5. True or False: The following program will terminate:

```
balance = 10
while True:
    if balance < 9:
        break
balance = balance - 9
```

6. What is sum after the following loop terminates?

```
sum = 0
item = 0
while item < 5:
    item = item + 1
    sum = sum + item
    if sum > 4:
        break
print(sum)
```

Part 2:

Please answer the following questions in Blackboard after reading Sections 1.3 and 1.4.

1. What will be the result in Python

```
import numpy as np
integers = np.arange(1, 5)
primes = np.array([2, 3, 5, 7])
print(integers * primes)
```

2. What will be the result in Python

```
a = np.array([1.0,2.0,3.0])
b = 2.0
print(a * b)
```

3. Which Matplotlib function generates a histogram?

4. Describe the plot generated by

```
import numpy as np
import matplotlib.pyplot as plt
x = [1, 2, 3]
y = [1, 2, 1]
plt.plot(x, y, "ko")
plt.show()
```

3.3 Group work

Part 1:
Please work in pairs to program the pseudocode in the Loops and Conditionals Lab in Python using Jupyter notebooks. The person who identifies as the one with the least programming experience will be the scribe who types in the Jupyter notebook. The second person will be the spokesperson and should be prepared to present to the class the following reflections.

- Which part of programming did you find to be the easiest?
- What strategies did you use when you got an error message?
- How did you verify your code was correct?

Here are some suggestions to verify your code is correct:

- Start by choosing a value of n that is small enough you can check your answer by hand. Then show a result for a large value of n that would be too tedious to check by hand.
- Choose x to be a vector containing random numbers from a normal distirbution with mean mu and standard deviation sigma. Do this with the Python commands

  ```
  import numpy as np
  x = np.random.normal(mu, sigma, n)
  ```

  with n representing the number of elements in the vector. Use this strategy to check if your calculation of the mean is the same as mu. Similarly compare your calculation of the standard deviation to sigma.

Once you are finished, delete the cells with errors and save a copy of the notebook with output for each pseudocode in a pdf file. The name of the file should contain the last names of each person in the group. Upload the file into Blackboard under the Group Assignment for today.

Part 2:

1. Please write your answers to the following questions on the supplied note cards.
   Assume you import numpy as np in Python.

   (a) Identify the length of each of the following arrays and order them from longest to shortest:

array1=np.arange(5)
array2=np.arange(1,5)
array3=np.arange(100)
array4=np.arange(1,2,0.1)

(b) Consider that
array1=np.linspace(start = 0, stop = 1, num = 50)

is an array with 50 elements that starts at 0 and ends at 1. Let
array2=np.random.normal(0, 1, 50)

Describe the difference between array1 and array2.

(c) Estimate the maximum element in each of the following arrays and order them from the one with the largest element, to the one with the smallest element:
array1=np.random.normal(0, 2, 40)
array2= np.random.normal(2, 0, 40)
array3=np.random.normal(40, 2, 40)

2. Please work in pairs to create the following plot. The person who identifies as the one with the least programming experience will be the scribe who types in the Jupyter notebook. Use plt.savefig to save the plot as a .pdf file. The name of the file should contain the last names of each person in the group. Upload the file into Blackboard under the Group Assignment for today.

(a) Plot $y = 1 + 2x$ with x linearly spaced in the interval $[2, 3]$.

(b) Label the axes and make a title for the plot.

(c) Plot $y = 8 - x$ on the same plot that you created in 2a.

(d) Research documentation on how to create a legend. Create a legend that labels the two lines on your plot in 2c.

Chapter 4

Functions

4.1 Introduction

In many cases, you will want to evaluate expressions multiple times using different values of the variables. While you might be able to cut and paste the expressions multiple times, this is error prone. It is much better to create a "function" that can be called multiple times using different arguments.

For example from the Loops and Conditional Statements lab we could simplify the standard deviation calculation

```
mu = 0
for i = 1:n
    mu = mu + x(i)
end
mu = mu/n
s=0
for i = 1:n
    s = s + (x(i)-mu)^2
end
s = sqrt(s/(n-1))
```

Instead define

```
function mymean(x)
    n=len(x)
    mu = 0
    for i = 1:n
        mu = mu + x(i)
    end
    mu = mu/n
return mu
end
```

so that the standard deviation calculation becomes

```
mu=mymean(x)
s=0
for i = 1:n
    s = s + (x(i)-mu)^2
end
s = sqrt(s/(n-1))
```

Once a function is defined it can be used anywhere, including in other functions. For example, we could also define a function for the standard deviation that uses the function we created for the mean:

```
function mystd(x)
    mu=mymean(x)
    n=len(x)
    s=0
    for i = 1:n
        s = s + (x(i)-mu)^2
    end
    s = sqrt(s/(n-1))
return s
end
```

so that we only need to type

```
s=mystd(x)
```

to get the standard deviation.

You will notice that I did not call these functions *mean* and *std*. That is because many libraries already have functions with those names and it can cause errors to use them in a different way.

Some programming languages make a distinction between "functions" that return values and "subroutines" that do not return anything but rather *do* something like produce a plot. In Python and MATLAB there is only one kind, functions, and they can return single, multiple, or no values at all.

Argument variables within functions exist in their own namespace. This means that assignment of an argument to a new value does not affect the original value outside of the function. For example using the function mymean above if we type

```
mu=5
x=[1, 1, 1, 1]
xbar=mymean(x)
print(mu)
```

the answer will be 5, which has nothing to do with the mean of x. On the other hand, print(xbar) will produce the mean of x which is equal to 1.

4.1.1 Anonymous Functions

Anonymous functions are in-line functions that can be generated on the fly to accomplish some small task. You can assign them a name, but you don't need to; hence, they are often called anonymous functions. You may find them convenient to send a function like $f(x) = \cos(x)$ into a named function. Most likely the named function will be the important part of the code while $f(x)$ just tests it. If you use an anonymous function for $f(x)$ then you wont clutter your code with a lot of one line statements.

4.1.2 Functions in Python

The function to calculate the average that we wrote in pseudocode in the previous section is written in Python as

```
def mymean(x):
    n=len(x)
    mu = 0
    for i in range(n):
        mu = mu + x[i]
    mu = mu/n
    return mu
```

The **return** statement causes a function, loop or conditional to exit or terminate immediately, even if it is not the last statement of the function, loop or conditional. The return statement also identifies what variables should be passed out. If no return statement is present within a function, or if the return statement is used without a return value, Python automatically returns the special value None when the function is called.

To return both the mean and the standard deviation, we would also create the function

```
def mystd(x):
    mu=mymean(x)
    n=len(x)
    s=0
    for i in range(n):
        s = s + (x[i]-mu)**2
    s = (s/(n-1))**(1/2)
    return mu, s
```

and call it with

```
xbar, sigma = mystd(x)
```

Note that if we typed the statement

```
print(s)
```

we would get the error statement *NameError: name 's' is not defined.*

An **anonymous function** is also called a lambda expression so Python uses the general form

lambda arg1, arg2, ... : output

The arguments arg1, arg2, ... are inputs to a lambda, just as for a functions, and the output is an expression using the arguments. For example, these two functions are equivalent in Python

```
def f(a, b):
    return 3*a+b**2
```

```
g = lambda a, b : 3*a+b**2
```

i.e. f(a,b)=g(a,b).

4.2 Individual Lab

Please answer the following questions in Blackboard Part 1:

1. What value does this function return if you pass in 4?

```
function fact(N)
    var =1
    while N > 0
        var = var*N
        N = N-1
    end
    return var
end
```

2. What value does this function return if you run mystery(2,5)?

```
def mystery(X, Y):
    if X = Y :
        return X*Y
    if X > Y:
        return X - Y
    else:
        return Y - X
```

3. What statement will correctly find the slope of the line passing through the points (1,-2) and (-1,2)?

```
def myslope(x1,x2,y1,y2):
    slope=(y2-y1)/(x2-x1)
    return slope
```

4. Define the functions

```
def fun_f(x):
    fun=np.cos(x)
    return fun
```

```
def fun_g(x):
    fun=x**2
    return fun
```

and identify how to evaluate $\cos(x^2)$, $(\cos(x))^2$, $x^2\cos(x)$ and $x^2 + \cos(x)$.

5. Review Netwon's method from Calculus I: `http://tutorial.math.lamar.edu/Classes/CalcI/NewtonsMethod.aspx` and answer the question in Blackboard.

6. Review Taylor series from Calculus II: `http://tutorial.math.lamar.edu/Classes/CalcII/TaylorSeries.aspx` and answer the question in Blackboard.

7. Review numerical integration from Calculus II: `http://tutorial.math.lamar.edu/Classes/CalcII/ApproximatingDefIntegrals.aspx` and answer the question in Blackboard.

Part 2:

1. What is the nth degree Taylor series polynomial for e^x?

2. Assume we want to find the roots of $f(x) = x^3 - 7x^2 + 8x - 3$. Start with an initial guess of $x_0 = 5$ and find the first estimate x_1 using Newton's method.

3. Approximate the derivative of $f(x) = \sin(x)$ at $x = 1.0$ with the formula $\frac{f(x+h)-f(x)}{h}$ and a step size $h = 0.1$.

4. Approximate $\int_1^5 \frac{1}{x^3+1}\,dx$ with midpoint rule. Divide the interval into 4 subintervals of equal length in your approximation.

4.3 Group work

1. Identify which function does (i) Netwon's method, (ii) Taylor series, (iii) numerical differentiation, or (iv) numerical integration. Please write your answer on the supplied note cards e.g. func1 and Newton, func2 and Taylor, etc.

```
def func1(f, x, a,b,h):
    x=np.arange(a,b,h)
    g = (f(x+h) - f(x))/h
    return g
```

```
def func2(n,x):
    f = 0
    for i in range(n):
        f+= x**i/np.math.factorial(i)
    return f
```

```
def func3(x, f, fprime, max_iter, tol):
    for i in range(max_iter):
        step = f(x)/fprime(x)
        if abs(step) < tol:
            return i, x
        x -= step
    return i, x
```

```
def func4(f,a,b,h):
    sum = 0.0
    x = a + h/2
    while (x < b):
        sum += h * f(x)
        x += h
    return sum
```

2. Please form a new group of 3-4 people. Each team will consist of:

   - The **manager**, responsible for coordinating the work of the team. The manager is the person who has the most siblings.
   - The **spokesperson**, who will report your group's work to the rest of the class. The spokesperson is the person who has the shortest time to graduation.
   - The **scribe**, who will be responsible for writing down your team's findings. The scribe is the person who has taken the most foreign language courses.
   - The **timekeeper**, who will keep track of time for each exercise. The timekeeper is the person who was not assigned a role already. In case of a three-person group, the manager takes the role of a timekeeper.

   One person should not have more than one role (except for the manager/timekeeper combo), so if a person who already was assigned a task meets the criteria for another one, the person who is next in line (i.e., second-longest commute) takes this role.

   Each group will be assigned one method and the spokesperson will present their answers to the following questions to the class.

   (a) Newton's method
      i. Explain the inputs and outputs to the supplied function and their dimension e.g. a scalar or a vector with specific dimension.
      ii. Describe x and fprime in the supplied function and their role in Newton's method.
      iii. Describe the role of max_iter and tol in the supplied function. Explain what happens when they are adjusted.
      iv. In the supplied function, there is an initial guess of the root, but it is implicitly defined. Explain.
      v. Give a specific example where you call the function and print or plot the output.

   (b) Taylor series
      i. Explain the inputs and outputs to the supplied function and their dimension e.g. a scalar or a vector with specific dimension.
      ii. Describe the role of n. Give an example to describe the effect of it being adjusted.
      iii. If the supplied function can be used to find Taylor series of any mathematical function, give an example. If it can't be used for any mathematical function, how would you adjust the function to estimate the Taylor series for a different mathematical function?
      iv. Give a specific example where you call the function and print or plot the output.

   (c) Numerical differentiation
      i. Explain the inputs and outputs to the supplied function and their dimension e.g. a scalar or a vector with specific dimension.
      ii. What do a,b, and h represent? What is the effect of adjusting h?

iii. Give an example of how you would input a specific f.

iv. How could you change the differentiation function so that it takes data rather than a function as input? What would be potential problems/errors?

v. Give a specific example where you call the function and print or plot the output.

(d) Numerical integration

i. Explain the inputs and outputs to the supplied function and their dimension e.g. a scalar or a vector with specific dimension.

ii. Describe what happens in the supplied function when we input $h<0$.

iii. Explain why this is called the midpoint rule.

iv. Give a specific example where you call the function and print or plot the output.

Chapter 5

Working with data files

5.1 Introduction

An important component of any scientific program is the ability to read and write data to files, and manipulate the data. Consider a generic file data.txt whose contents are:

data.txt

```
#pressure temperature average_energy
1.0 1.0 -50.0
1.0 1.5 -27.8
2.0 1.0 -14.5
2.0 1.5 -11.2
```

5.1.1 Reading and writing with Numpy

NumPy routines loadtxt() and savetxt() can be used useful for quickly loading and saving simple array data like that contained in data.txt. For example

mydata = np.loadtxt('data.txt')

creates an array with 4 rows and 3 columns. If a line starts with a number sign (#), it will be ignored as a comment. Blank lines are also ignored. You can see the dimension of the array with

mydata.shape

which in this case returns (4,3).

If we want to alter the data, say change the average_energy in the third row to -15.4 excecute

mydata[2,2]=-15.4

We could then save the new data into a new file

np.savetxt('data_alt.txt', mydata)

We can also load each variable into a separate 1-dimensional array. Setting the argument unpack=True and providing a variable for each column accomplishes this

p, T, AE= np.loadtxt('data.txt', unpack=True)

If you want to read in only some columns, you can use the usecols argument to specify which ones

p, AE= np.loadtxt('data.txt', unpack=True, usecols=[0,2])

Similarly, if you can save each variable as a separate 1-dimensional array. If we want to save the first and third columns

np.savetxt('data_alt.txt', mydata[:,range(0,3,2)])

Note the indexing of the array.

5.1.2 Formatting

By default, when using savetxt in Numpy the numbers will be written in scientific notation. The *fmt* argument can be used to specify the formatting. If one format is supplied, it will be used for all of the numbers.

The general form of the fmt argument is: *fmt = '%(width).(precision)(specifier)'* where width specifies the maximum number of digits, precision specifies the number of digits after the decimal point, and the possibilities for specifier are shown below. For integer formatting, the precision argument is ignored if you give it. For scientific notation and floating point formatting, the width argument is optional.

Specifier	Meaning	Example	Format Output for -34.5678
i	signed integer	%5i	-34
e	scientific notation	%5.4e	-3.4568e+001
f	floating point	%5.2f	-34.57

For example

np.savetxt('data_alt.txt', mydata, fmt=('%3.1f', '%3.1f', '%4.1f'))

saves the columns as floating point numbers, rather than scientific notation, in the file data_alt.txt.

5.1.3 Pandas

Pandas is a Python software library written for data manipulation and analysis. It can give more functionality with data than is directly available with Numpy arrays.

import pandas as pd

The name Pandas comes from an econometrics term for data sets: panel data. While there are many different important features of Pandas, here we will just talk about them in terms of CSV files.

CSV is an acronym for "comma separated values" but really most interpretations of it can be assumed to mean delimter-separated values files. This means data are separated in each row with delimiter characters. Most spreadsheet programs do this, as does the file data.txt. For example

csv_data = pd.read_csv('data.txt')
print(csv_data)

results in the following

```
  #pressure temperature average_energy
0                        1.0 1.0 -50.0
1                        1.0 1.5 -27.8
2                        2.0 1.0 -14.5
3                        2.0 1.5 -11.2
```

In order to access the data as arrays as we did with Numpy, we need to sort them in columns. This is done by adding the option *sep=' '* in pd.read_csv, leaving a blank space between the quotes so pandas can detect spaces between values. In addition, we can skip the first line that contains a comment by setting the parameter *header* to None and using *skiprows*

csv_data = pd.read_csv('data.txt',skiprows = 1,header=None, sep=' ')

which results in

```
     0    1      2
0  1.0  1.0 -50.0
1  1.0  1.5 -27.8
2  2.0  1.0 -14.5
3  2.0  1.5 -11.2
```

Before working with the data, it is a good idea to check the information that is stored in csv_data. First, check the dimension by

csv_data.shape

Second, check the data type (e.g. string, integer, floating point number) by

csv_data.dtypes

This gives output

```
0    float64
1    float64
2    float64
dtype: object
```

This means that columns 0-2 consist of floating point numbers, which is what we would expect.

In order to work with the floating point numbers in the data structure we need to assign a label to each column

csv_data.columns = ['Pressure','Temperature', 'Average_Energy']

Then we can access the first column with

csv_data.Pressure

and the second element in the first column with

csv_data.Pressure[1]

If we want to manipulate the data and do arithmetic operations, e.g. see if pressure and average energy are proportional we can find the constant of proportionality for the second row

const=csv_data.Pressure[1]/csv_data.Average_Energy[1]

We can also find an array of constants for each row

const_array=csv_data.Pressure/csv_data.Average_Energy

5.2 Individual Lab

You may use the file data.txt available in Blackboard to answer the following questions after reading the Working with data files lab.

1. As described in Section 1.1, what is the result in Python if you execute p.shape?
2. As described in Section 1.1, data is saved to the file data_alt.txt twice. Note that each time you use the command np.savetxt it overwrites the existing file (if there is one). What does the file data_alt.txt look like after the second execution, i.e. np.savetxt('data_alt.txt', mydata[:,range(0,3,2)])?
3. As described in Section 1.2, what does the file data_alt.txt look like after the execution np.savetxt('data_alt.txt', mydata, fmt=('%i3', '%3.1f', '%4.1f'))
4. As described in Section 1.3, what is the output when csv_data.shape is executed?
5. As described inSection 1.3 what is the output when const=csv_data.Pressure[1]/csv data.Average Energy[1] is executed?

5.3 Group work

Please work in pairs to load, plot and analyze the data in the file galton.dta available in Blackboard. This data file contains the heights of children and their parents. The file extension .dta comes form the statistical software package STATA which integrates well with Python. You will plot these data to see if the heights of parents and children look to be related linearly. Next week we will talk about fitting a least squares regression line to the data.

1. Let's start by loading the data and get comfortable with its contents by executing the following commands

```
import pandas as pd
df = pd.read_stata('galton.dta')
df.shape
df.dtypes
```

Be prepared to describe the categories, their formats and values. I suggest printing a subset of the data values to help understand each category, e.g.

```
print(df.family[1:10])
```

Note that the category family is a family ID variable, as the data include multiple children per family.

2. Plot the heights of mother vs kid and father vs kid. Make sure to plot data points using either '.' or plt.scatter and not a continuous line. **Be prepared to discuss the potential linear relationship between the mothers' height and the kids' height. Also, the potential for the fathers' height to be linearly related to the kids' height.** It may help to make both axes equal to each other and try different scales to come to your conclusions.

3. Investigate the linear relationship between just the boys height and their father's height. In order to do this we must only use those rows of the data that refer to males. Start by creating an array with only the boys' height

   ```
   boy_height=df.height[df.male == 1]
   ```

   In order to compare it to the father's height we need to shorten the father array in the same manner so it only contains rows corresponding to their boy's height

   ```
   boy_father=df.father[df.male == 1]
   ```

   Plot the heights of the fathers vs their boys and **be prepared to discuss if you think there is a stronger linear relationship than those you saw in 2.**

4. Now investigate if there is looks like there is a linear relationship between just the girls's height and their mother's height. **Compare how their linear relationship looks to those you observed in 2 and 3.**

Chapter 6

Linear Regression

6.1 Introduction

Linear regression is a way to understand the linear relationship between two or more variables. This relationship can be written as a mathematical formula and used to

- describe the linear dependence of one variable on another.
- predict values of one variable from values of another.
- correct for the linear dependence of one variable on another, in order to clarify other features of its variability.

The **correlation coefficient**, on the other hand, measures the strength of a linear relationship between variables. This is different from regression which focuses on the mathematical form of the relationship.

6.2 Simple Linear Least Squares Regression

In simple linear regression the mathematical problem is as follows:

$$\text{Given a set of } k \text{ points } (x_i, y_i) \quad i = 1, 2, ..., k$$

assume they are related through the equation

$$y_i \approx b_0 + b_1 x_i$$

where b_0 and b_1 are constant (unknown) coefficients. Most data (x_i, y_i) contain noise and it is not possible for every data point to satisfy this linear relationship. Therefore we seek to find b_0 and b_1 so that

$$y_i = b_0 + b_1 x_i + n_i$$

where n_i represents the noise in each data point.

For least squares regression we assume the noise n_i follows a Gaussian or normal distribution. Our goal is to find coefficients b_0 and b_1 and form the best fit, or **regression line**

$$\hat{y}(x) = b_0 + b_1 x. \tag{6.1}$$

If the noise has zero-mean we find b_0 and b_1 by minimizing S, the sum of squared errors:

$$S = \sum_{i=1}^{k} (y_i - \hat{y}_i)^2.$$

We can use multivariable calculus to find the values of b_0 and b_1 that minimize S but we will not go through the derivations here. If we let $\mu_{\mathbf{x}}$ denote the mean of $\mathbf{x}$ and $\mu_{\mathbf{y}}$ the mean of $\mathbf{y}$, then the coefficients b_0 and b_1 can be written in the convenient form

$$\begin{aligned} b_0 &= \mu_{\mathbf{y}} - b_1 \mu_{\mathbf{x}} \\ b_1 &= \frac{\sum_{i=1}^{k} (x_i - \mu_{\mathbf{x}})(y_i - \mu_{\mathbf{y}})}{\sum_{i=1}^{k} (x_i - \mu_{\mathbf{x}})^2}. \end{aligned}$$

The predicted value of y at any point x is given by $\hat{y}(x)$ in (6.1). At the data points x_i, $\hat{y}(x_i)$ will most likely not equal the observed value y_i. The residual $e_i = y_i - \hat{y}(x_i)$ measures how far the prediction is from the observations. The variance in the residual is

$$\hat{\sigma}^2 = \frac{\sum_{i=1}^{k} e_i^2}{k-2}. \tag{6.2}$$

The $k-2$ in the denominator is known as the "degrees of freedom", and is computed by subtracting the number of parameters estimated (b_0 and b_1) from the number of observations.

6.3 Correlation Coefficient

The **coefficient of determination** r^2 is a measure of how well the regression line represents the data. It is defined as:

$$r^2 = 1 - \frac{\sum_{i=1}^{k} e_i^2}{\sum_{i=1}^{k} (y_i - \mu_{\mathbf{y}})^2.} \tag{6.3}$$

In simple linear regression, the **correlation coefficient** r is simply the square root of the coefficient of determination. One advantage of r is that it is unitless, allowing researchers to make sense of correlation coefficients calculated on different data sets with different units.

As an example consider $r = 0.9$, i.e. $r^2 = 0.81$. This means that 81% of the total variation in $\mathbf{y}$ can be explained by the linear relationship between $\mathbf{x}$ and $\mathbf{y}$. The other 19% of the total variation in $\mathbf{y}$ remains unexplained.

6.4 Multivariate Linear Least Squares Regression

Simple linear regression is limited because we can only consider two variables of interest. For example, a child's height (y) may depend on both their mother's (x_1) and father's (x_2) height. In

simple linear regression we only have the following two equations to work with

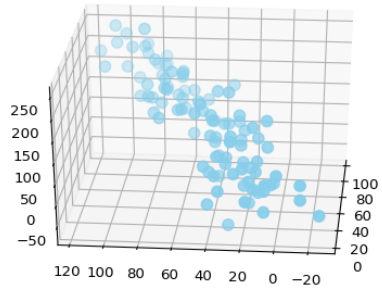
$$\hat{y}(x_1) = b_{10} + b_{11}x_1 \quad \text{and} \quad \hat{y}(x_2) = b_{20} + b_{21}x_2$$

where b_{10}, b_{11} are the constants for simple regression with their mother's height b_{20}, b_{21} and are the constants for the father's. The problem with two separate regression lines is that neither captures how y depends on x_1 **and** x_2 simultaneously. In fact, by not including all *confounding factors* in the simple linear regression equations we create bias in our estimate of the child's height.

Multivariate regression is a step towards remedying this by adding variables to the simple linear regression case, i.e.

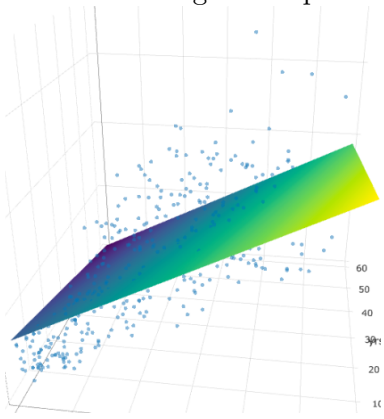
$$\hat{y}(x_1, x_2) = b_0 + b_1x_1 + b_2x_2.$$

A good way to think about regression is that we're finding weights (b_1, b_2) that determine the importance of each of the variables $(x_1,\ x_2)$. To visualize this relationship, we need to think in three dimensions. Before, with simple regression, we could plot the relationship on a 2D graph. Now the data points are in a 3D space of child's height, mother's height, and father's height. Here's a picture of a 3D scatterplot.



Note that this graph does not represent the height data we have been talking about, i.e. $x_1 \in [-20, 120]$, $x_2 \in [0, 100]$ and $y \in [-50, 250]$.

When we run a regression in this space, we find a regression plane, not a regression line. This plane in 3D space has two different slopes: the amount it increases when you increase the x_1 variable and the amount when you increase x_2. The variables b_1 and b_2 describe the two slopes but there is only one intercept b_0 which is the value of the plane where it touches the vertical y-axis. Here is a picture of a different 3D scatterplot with a 2D regression plane.



Visualizing multvariate regression helps our understanding of the process. However, we will not focus on this aspect as it needs more attention than we have time and can be found in a course focused on visualization.

In order to find equations for the multivariate coefficients b_0, b_1 and b_2 we use the matrix form of regression. If we have two independent variables (e.g. mother's and father's height) the data are (x_{i1}, x_{i2}, y_i) $i = 1, \ldots, k$. Each data point satisfies

$$y_i = b_0 + b_1 x_{i1} + b_2 x_{i2} + n_i.$$

All the data can be written simultaneously with the the matrix equation

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_k \end{bmatrix} = \begin{bmatrix} 1 & x_{11} & x_{12} \\ 1 & x_{21} & x_{22} \\ \vdots & \vdots & \vdots \\ 1 & x_{k1} & x_{k2} \end{bmatrix} \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix} + \begin{bmatrix} n_1 \\ n_2 \\ \vdots \\ n_k \end{bmatrix}.$$

We can write this matrix equation more simply as

$$\mathbf{y} = \mathbf{X}\mathbf{b} + \mathbf{n}. \tag{6.4}$$

The least squares strategy for finding $\mathbf{b}$ is the same as in the simple linear regression model, but in multi-dimensions. We minimize S, the sum of squared errors

$$S = \sum_{i=1}^{k} (y_i - \hat{y}_i(x_1, x_2))^2 = \sum_{i=1}^{k} (\mathbf{y} - \mathbf{X}\mathbf{b})_i^2.$$

We again use multivariable calculus to find the values in $\mathbf{b}$ that minimize S but we will not go through the derivations here. The coefficients can be written in the convenient form

$$\mathbf{b} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}. \tag{6.5}$$

The variance in the residual $\hat{\sigma}^2$ and the correlation coefficient r are calculated in the same manner as for simple linear regression in (6.2) and (6.3), respectively. The R^2 value in multivariate linear regression is often called the coefficient of multiple determination.

6.5 Matrices in Python

Multivariate regression requires us to create matrices and solve systems of equations. Vectors such as $\mathbf{v} \in \mathbb{R}^k$ can be thought of as a special case of a matrix like $\mathbf{A} \in \mathbb{R}^{k \times p}$. With that thought in mind creating a matrix in Python is very similar to creating a vector. Recall to create the vector we type

```
v=np.array([1,2,3])
```

This is a row vector $[1, 2, 3]$, i.e. $\mathbf{v} \in \mathbb{R}^{1\times 3}$. Instead, let's create it as a column vector $\begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix} \in \mathbb{R}^{3\times 1}$

```
v_col = np.array([[1],
                  [2],
                  [3]])
```

In Python v has shape (3,) while v_col has shape (3, 1).

Viewing the difference between row and column vectors helps us see how a vector is a special case of a matrix. For example the matrix $\mathbf{A} = \begin{bmatrix} 1 & 2 & 3 \\ 3 & 4 & 5 \\ 6 & 7 & 8 \end{bmatrix} \in \mathbb{R}^{3\times 3}$ is created in Python by

```
A = np.array([[1,2,3],
              [3,4,5],
              [6,7,8]])
```

Matrix indexing is similar to vector indexing. For example, $\mathbf{A}_{23} = 5$ and this is accessed in Python with A[1,2]. (Recall that Python indexing starts as 0). If we want to access the last two columns of the first two rows

```
print(A[:2, 1:3])
```

returns

```
[ [2  3]
  [4  5]]
```

Matrix algebra such as addition, subtraction and multiplication requires that the dimension of the matrix and vectors *"agree"*. For addition and subtraction, the objects must have the same dimension. In Python we can simply type v+v or A+A.
WARNING: If we type v+v_col this should give us an error but instead we get

```
[[2,3,4],
 [3,4,5],
 [4,5,6]])
```

I suggest thinking about how Python is interpreting this algebra.

Matrix multiplication such as $\mathbf{A}\mathbf{v}_{col}$ requires that the number of columns in $\mathbf{A}$ equal the number of rows in $\mathbf{v}_{col}$. The process of matrix multiplication involves multiplying each row by each column. If you haven't done this in awhile, I recommend reviewing it. Matrix multiplication in Python is done with the *np.dot()* command, e.g.

```
print(np.dot(A,v_col))
```

produces the correct answer

```
[[14]
 [26]
 [44]]
```

WARNING: If we type print(np.dot(A,v)) this should give us an error (why?) but instead we get

[14 26 44]

Fortunately, if we type print(np.dot(v_col,A)) we will get an error message while if we type print(np.dot(v,A)) we get the correct answer (what is it?). While Python has advanced capabilities for data analysis, unfortunately it falls short with matrix algebra.

Our focus here is to solve the linear system $\mathbf{Xb} = \mathbf{y}$ for the multivariate regression coefficients. More advanced courses in computational math focus on algorithms to solve linear systems, while in this introductory course we will just the existing library *np.linalg.solve()*.

For example solve $\mathbf{A}\mathbf{v}_{col} = \mathbf{y}$ for $\mathbf{v}_{col}$. First we form $\mathbf{y}$, then use np.linalg.solve to estimate $\mathbf{v}_{col}$:

```
y =np.array( [[14],
              [26],
              [44]])
v_col_est = np.linalg.solve(A,y)
```

print(v_col_est) gives us the estimate

```
[[2.]
 [0.]
 [4.]]
```

Interesting! The correct answer is $\mathbf{v}_{col} = \begin{bmatrix} 1 \\ 2 \\ 3 \end{bmatrix}$ and the estimate v_col_est is very different. The issue is not that Python is doing something fishy with our matrices and vectors. Rather, the issue is the mathematical solution of $\mathbf{A}\mathbf{v}_{col} = \mathbf{y}$ has issues. We'll talk more about this later.

Let's say we don't know know the correct answer is $\mathbf{v}_{col}$. How can we check if our answer is correct? Naively, we could take our estimate v_col_est, multiply it by A and see if we get y

```
print(np.dot(A,v_col_est))
```

The result is

```
[[14]
 [26]
 [44]]
```

Interesting! The matrix multiplication gave us the correct y even though our estimate v_col_est was incorrect. More on this later.

6.6 Individual Lab

Simple Linear Regression

These questions should be completed in Blackboard before class. For the data (0,0), (-1,1) and (4,-1) answer the following questions in Blackboard

1. Find the regression coefficients b_0 and b_1.
2. Find the predicted value of y at $x = 2$.
3. Find the residual $e_2 = y_2 - \hat{y}(x_2)$.
4. Find the variance in the residuals.
5. Find the correlation coefficient between between **x** and **y**.

Multivariate Linear Regression

1. Consider the multivariate data (x_1, x_2, y): $(-2, 4, 1), (-2, 1, 0), (0, 0, 2), (1, 1, 3)$ and assume we fit it to the line $\hat{y}(x_1, x_2) = b_0 + b_1 x_1 + b_2 x_2$.
   (a) Identify the matrix **X** in (6.4).
   (b) Identify the vector **y** in (6.4).
   (c) Identify the formula for the coefficients $\mathbf{b} = \begin{bmatrix} b_0 \\ b_1 \\ b_2 \end{bmatrix}$ in (6.5).
   (d) The estimates of the coefficients are $b_0 = 1.83, b_1 = 1, b_2 = 0.28$. How would you interpret these values?
2. If $\mathbf{X} = \begin{bmatrix} 1 & -2 & 4 \\ 1 & -2 & 1 \\ 1 & 0 & 0 \\ 1 & 1 & 1 \end{bmatrix}$ then the following values of **b** and **y** satisfy $\mathbf{Xb} = \mathbf{y}$.

6.7 Group Work

Simple Linear Regression

Please write your answers to the following questions on the supplied note cards.

1. Given the data (0,0), (-1,1) and (4,-1) write the commands that create Numpy arrays x and y containing the x and y coordinates, respectively.
2. Given only a Numpy array x, write a function that computes the mean of the elements in x. Please complete the remaining questions in groups of 2-3 people.
3. Using only pen and paper, create a function regcoef(x,y) that returns the linear regression coefficients b_0 and b_1. Use these tips and be prepared to explain to the rest of the class why one should follow them.
   (a) First write a function to compute the mean and call it within regcoef
   (b) Form b_1 before b_0.
   (c) Form the numerator and denominator in b_1 separately

(d) Call the function that computes the mean first.

4. Code your functions from 3 in Python and test them by finding the linear regression coefficients for the data (0,0), (-1,1) and (4,-1). Use the answer you found in the homework due today to check that your function is working correctly. Be prepared to discuss with the class the errors you encountered and your debugging strategies.

5. Define two anonymous functions: one to compute $\hat{y}$ and a second to compute residuals e_i. Define the functions so that you can form the residual vector $\mathbf{e}$ with one line. Be prepared to discuss the function inputs and how you would access each element e_i.

6. Write a regular function that computes the variance in the residual and use your anonymous functions from 5. Check that your function works by testing it with data (0,0), (-1,1) and (4,-1) for which you know the correct answer. Be prepared to discuss with the class the errors you encountered and your debugging strategies.

7. Write a regular function that computes the correlation coefficient. Again check your answer with data (0,0), (-1,1) and (4,-1) for which you know the correct answer. In addition, plot the data and the regression line on the same graph. Be prepared to discuss with the class how well the line predicts points other than the data.

Multiple Linear Regression

An important characteristic of a semiconductor product is the pull strength of a wire bond. We will investigate the suitability of using a linear multiple regression model to predict pull strength y as a function of wire length x_1 and die height x_2.

1. Load the data in wire.txt and form one dimensional arrays y, x_1 and x_2. Here are some tips

   (a) Refer to the Working with data files Lab to remind yourself how to read and work with a data file.

   (b) Identify the shape of the array you created when loading the data. All the data will be loaded into one column. Separate it into the correct number of rows and columns by following the instructions on the bottom of page 2 and top of page 3 in the Working with data files Lab.

   Be prepared to discuss the shape of the arrays y, x_1 and x_2 and how they relate to our formula for the regression coefficients $\mathbf{b}$ in equation (6.5).

2. Form the regression matrix $\mathbf{X}$ in (6.4). Note that it has 3 columns and here are some tips

   (a) You can transpose a matrix $\mathbf{A}$ by executing A.transpose()

   (b) You can create an array of 1s using np.ones.

   (c) You can stack 1-D arrays as columns using np.column_stack

   Be prepared to discuss challenges in getting the dimension of $\mathbf{X}$ correct.

3. The formula for $\mathbf{b}$ in (6.5) involves finding the inverse of the matrix $\mathbf{X}^T\mathbf{X}$. Don't use that formula, rather find $\mathbf{b}$ by solving the system $\mathbf{X}^T\mathbf{X}\mathbf{b} = \mathbf{X}^T\mathbf{y}$ because that is a more accurate and efficient approach on a computer.

(a) Use matrix algebra to explain why forming $\mathbf{b}$ as described in (6.5) is the same as solving the system $\mathbf{X}^T\mathbf{X}\mathbf{b} = \mathbf{X}^T\mathbf{y}$ for $\mathbf{b}$.

(b) Form the matrix $\mathbf{A} = \mathbf{X}^T\mathbf{X}$ and the vector $\mathbf{rhs} = \mathbf{X}^T\mathbf{y}$ and use np.linalg.solve to solve $\mathbf{A}\mathbf{b} = \mathbf{rhs}$.

Be prepared to report the dimension of $\mathbf{A}$ and $\mathbf{rhs}$ and your values for the coefficients $b_i,\ i = 1, \ldots 3$.

Chapter 7

MATLAB

7.1 Introduction

This semester we have be using Python, a free, open source programming language. Python has matured significantly over the past years and Juypter notebooks have made it more accessible. However, as we saw at the end of the Linear Regression Lab, the linear algebra capabilities can be awkward. For example, Python does not inherently distinguish between row and column vectors, it gives non-unique solutions to linear systems without warning, while indexing from 0 is not how vectors and matrices are typically referenced in mathematics.

Since this is a computational math course with significant focus on numerical linear algebra, we will begin to implement more complex algorithms in MATLAB. MATLAB is a commercial numerical computing environment and programming language. Since it is a commercial product, we cannot see the code for the built in functions and have to trust they are implemented correctly. MATLAB also has problems with portability, meaning it is quite possible if you run your MATLAB code on a different computer you may get a different answer. On the plus side, MATLAB originated as a matrix manipulation package and doesn't suffer from the problems mentioned above that we with found with Python.

You may continue using Python, but this may be more work when we use specific MATLAB functions. In most cases, there are Python functions equivalent to those we use in MATLAB, but it will take time for you to identify and use them on your own. Please keep in mind that the focus of this class is not the particular language you program in, rather understanding the computational algorithms. The algorithms and computational thinking you develop in this class are foundational, while particular languages may change in the next 5-10 years.

MATLAB is available on most all university computers. As a Boise State student you may also download it on your computer for free. Instructions can be found in the link in the syllabus. You will need to create a Mathworks user name and password, which is the parent company of MATLAB.

7.1.1 Working with data in MATLAB

Let's start using MATLAB with a new data set that contains the US census for regarding population and race in Billings, Montana. The file is named census.txt:

	Year	TotalPop	AsianPop	BlackPop	HispanicPop	NativePop	WhitePop
0	1960	13806	NaN	187	NaN	NaN	13619
1	1970	19373	NaN	74	NaN	NaN	18888
2	1980	25210	NaN	121	781.0	NaN	23802
3	1990	24877	187.0	159	871.0	945.0	23254
4	2000	24642	165.0	153	1257.0	1083.0	22227
5	2010	29297	282.0	326	1835.0	1620.0	25577

You can load this into MATLAB with the command

C_table=readtable('census.txt')

To see the contents of the file type C_table without a print statement.

Two big differences between Python and MATLAB that may take a bit to getting used to are that indexing starts at 1 and elements in an array elements are accessed with ( ) in MATLAB rather than [ ] in Python. In particular if you want to access the first two rows, this is how it works in MATLAB

```
>> C_table(1:2,:)

ans =

  2×7 table

    Year    TotalPop    AsianPop    BlackPop    HispanicPop    NativePop    WhitePop
    ____    ________    ________    ________    ___________    _________    ________

    1960      13806       NaN         187          NaN           NaN         13619
    1970      19373       NaN          74          NaN           NaN         18888
```

while if you want to access the second row and third column it works like this

```
>> C_table(2,3)

ans =

  table

    AsianPop
    ________

      NaN
```

The notation NaN stands for Not a Number and MATLAB will overlook those values if we try to use them. In order to access elements in the table we first have to convert the table to an array with the *table2array* function

C_array=table2array(C_table)

If we want to get rid of the NaN data we could use the *isfinite* logical command. If there is a finite numerical value in an element of the array the logical argument is "true" and MATLAB returns a 1. Otherwise the logical argument is false it returns a 0. For example, if we look for finite values in the Native American population we see that only the last three elements in the column have finite value

```
>> isfinite(C_array(:,6))

ans =

  6×1 logical array

   0
   0
   0
   1
   1
   1
```

We can use *isfinite* to form a shortened array of data with only finite values. If we continue just viewing the Native American population the commands would be

isdata=isfinite(C_array(:,6))
N_pop=C_array(isdata,6)
N_years=C_array(isdata,1)

Note that N_years creates a three dimensional array that corresponds to the years for the data in N_pop. Since MATLAB ignores the NaN value the following two commands produce the same scatter plot:

figure(1);plot(C_array(:,1),C_array(:,6),'*')
figure(2);plot(N_years,N_pop,'*')

If we wanted to plot Hispanic and Native American population data on the same graph with a legend these are the commands in MATLAB

plot(C_array(:,1),C_array(:,5),'*',C_array(:,1),C_array(:,6),'*','LineWidth',3)
legend('Hispanic','Native American')
xlabel('Year'); ylabel('Population')
title('Census data for Billings, Montana')
shg

7.2 Group work

You may work in groups to answer the following questions however, everyone must create their own scripts and print their own plots.

1. Create a script that loads the census data and creates arrays N_pop and N_years that contains only finite values of census data as described in the lab. In addition, in your script create arrays with only finite values of census data for the Hispanic population. Be prepared to discuss with the class how do you verified that you correctly created the arrays.

2. Plot the Hispanic population and Native American population with C_array as described in the lab with a legend, title and axis labels. Be prepared to discuss with the class the purpose of the shg command and how to change the size of the data points.

3. Plot the Hispanic, Native American and White populations in one plot, similar to what you plotted in 2. Then plot the Native American and White populations in separate plots. You can make two separate plots by using the commands

   figure(1); plot(C_array(:,1),C_array(:,6),'*','LineWidth',3)
   figure(2); plot(C_array(:,1),C_array(:,5),'*','LineWidth',3)

   Include legends, title and axes label in each graph. Be prepared to discuss with the class challenges and benefits of plotting multiple graphs with one plot and with different plots.

4. Use the *axis* command to view each plot in 3. for the years 1960 to 2020. You can learn about the *axis* command by typing *help axis* at the MATLAB prompt. Print your graphs and draw a curve through the data points on each graph with a pen or pencil.

5. Extend your drawn in curves in 2b. so that they start in 1960 and end in 2020.

Chapter 8

Polynomial Interpolation

8.1 Introduction

Linear regression uses a line, plane or hyperplane in high dimensional space to describe (model) a collection of data points. More specifically, in two-dimensions, given a set of n points (x_i, y_i), $i = 1, 2, ..., n$, simple linear regression involves the assumption that the data are related through the equation $y_i \approx b_0 + b_1 x_i$ and the goal is to find coefficients b_0 and b_1.

In two dimensions it is not possible for every data point to satisfy $y_i = b_0 + b_1 x_i$ unless $n = 2$ i.e. there are only two data points. In linear algebra terminology, when $n = 2$, we can fit each data point to a line because there are the same number of equations as unknowns. This means we can set up a square linear system $\mathbf{y} = \mathbf{Xb}$ where

$$\mathbf{X} = \begin{bmatrix} 1 & x_1 \\ 1 & x_2 \end{bmatrix} \text{ and } \mathbf{b} = \begin{bmatrix} b_0 \\ b_1 \end{bmatrix}.$$

As long as the two data points have two different y values corresponding to x_1 and x_2, these two points uniquely determine a line.

When $n > 2$ the linear system of equations that results when we fit the data to a line is *overdetermined*, i.e. there is a greater number of equations than unknowns. Most of the time this means there is no solution and hence no coefficients b_0 and b_1 that produce a line that fits all of the data. We got around this issue by finding b_i that minimize the sum of squared errors: $S = \sum_{i=1}^{n} (y_i - \hat{y}_i)^2$.

8.2 Interpolation with the Vandermonde matrix

Rather than minimize the error in our data and the predicted line at the data points, let's extend the idea of creating square systems of equations so that we can fit (or model) each data point exactly. If we have $n + 1$ data points we can create the same number of equations by fitting them to a nth degree polynomial

$$p_n(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \ldots + a_n x^n.$$

This is called polynomial interpolation when $y_i = p_n(x_i)$ for all (x_i, y_i), $i = 1, 2, ..., n + 1$.

The system of equations for polynomial interpolation is

$$\begin{aligned} y_1 &= a_0 + a_1x_1 + a_2x_1^2 + a_3x_1^3 + \ldots + a_nx_1^n \\ y_2 &= a_0 + a_1x_2 + a_2x_2^2 + a_3x_2^3 + \ldots + a_nx_2^n \\ &\vdots \\ y_{n+1} &= a_0 + a_1x_{n+1} + a_2x_{n+1}^2 + a_3x_{n+1}^3 + \ldots + a_nx_{n+1}^n. \end{aligned}$$

The matrix equation that results from this system is $\mathbf{y} = \mathbf{Va}$ with

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_1^2 & x_1^3 & \ldots & x_1^n \\ 1 & x_2 & x_2^2 & x_2^3 & \ldots & x_2^n \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 1 & x_{n+1} & x_{n+1}^2 & x_{n+1}^3 & \ldots & x_{n+1}^n \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix}.$$

where the matrix $\mathbf{V}$ in the system is called a *Vandermonde* matrix.

Polynomial interpolation using the Vandermonde matrix is not typically used in practice. It is not practical because for a large number of data points it is expensive to solve the system of equations and the matrix has a large condition number. A large condition number means if you change the data slightly you will get a very different result. In other words the problem is unstable. Also, if you add data points, you have to solve an entirely new problem.

8.3 Lagrange Interpolation

Lagrange interpolation is an attractive alternative to using the Vandermonde matrix because you don't have to solve a system of equations to find the interpolating polynomial. In addition, you can easily add data points without having to start the process from scratch.

The difference in between using the Vandermonde matrix and Lagrange interpolation is how we write the polynomial. For example, $y = 2 - 3x + x^2$ and $y = (x-2)(x-1)$ are different ways of writing the same quadratic polynomial. The former would be the view for the Vandermonde matrix while the latter is how Lagrange Interpolation is written.

If we have two data points (x_i, y_i), $i = 1, 2$, the unique line that fits them is represented with the Lagrange polynomials as

$$p_1(x) = y_1l_1(x) + y_2l_2(x); \quad l_1(x) = \frac{(x-x_2)}{(x_1-x_2)}, l_2(x) = \frac{(x-x_1)}{(x_2-x_1)}.$$

Similarly if we have three data points (x_i, y_i), $i = 1, 2, 3$, the unique quadratic polynomial that fits them is represented with the Lagrange polynomials as

$$\begin{aligned} p_2(x) &= y_1l_1(x) + y_2l_2(x) + y_3l_3(x); \\ & l_1(x) = \frac{(x-x_2)(x-x_3)}{(x_1-x_2)(x_1-x_3)},\ l_2(x) = \frac{(x-x_1)(x-x_3)}{(x_2-x_1)(x_2-x_3)},\ l_3(x) = \frac{(x-x_1)(x-x_2)}{(x_3-x_1)(x_3-x_2)}. \end{aligned}$$

Notice that the coefficients for the polynomials are simply the y_i data which is why we don't have to solve a system of equations. In addition, we can add data more simply than in the Vandermonde matrix case.

For the more general case with n data points the Lagrange polynomial form of interpolation is written

$$p_n(x) = y_1 l_1(x) + y_2 l_2(x) + \ldots + y_n l_n(x) = \sum_{i=1}^{n} y_i l_i(x)$$

where

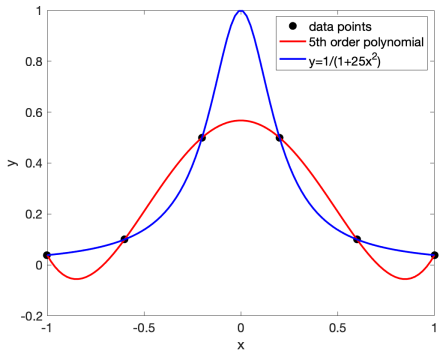
$$l_i(x) = \frac{(x-x_1)(x-x_2)\cdots(x-x_{i-1})(x-x_{i+1})\cdots(x-x_n)}{(x_i-x_1)(x_i-x_2)\cdots(x_i-x_{i-1})(x_i-x_{i+1})\cdots(x_i-x_n)} = \frac{\Pi_{j\neq i}(x-x_j)}{\Pi_{j\neq i}(x_i-x_j)}$$

8.4 Runge Phenomenon

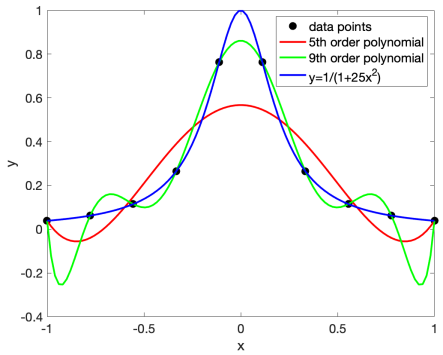
There are some pitfalls that occur when fitting a polynomial to data. As an example, let's simulate six data points by sampling the function $y = \frac{1}{1+25x^2}$ at equally spaced points on the interval $[-1, 1]$. We can create the (x, y) data pairs $(-1, 0.0385)$, $(-0.6, 0.1)$ $(-0.2, 0.5)$ $(0.2, 0.5)$ $(0.6, 0.1)$ $(1, 0.0385)$ in MATLAB with the following commands

```
n=6;
f=@(x) 1./(1 + 25*x.^2);
x=linspace(-1,1,n);
y=f(x);
```

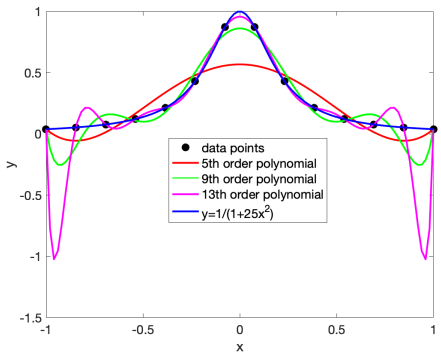
Polynomial interpolation with a 5th degree polynomial results in the following graph



We see that the 5th degree interpolating polynomial does a particularly poor job of interpolating at the peak and near the endpoints of the interval. Lets try to increase the accuracy by adding more data points and letting $n = 10$.

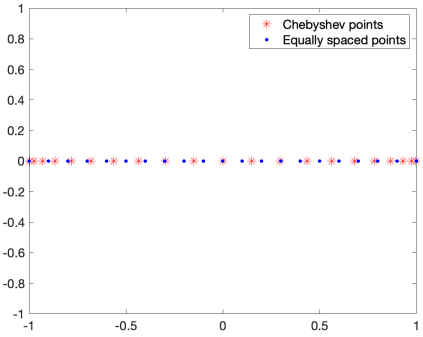


We are getting closer to modeling the peak, but the end points are getting worse. Notice that the y axis range increased to -0.4. Let's again try to improve the approximation by interpolating 14 points



Notice that the approximation is getting worse at the endpoints as we increase the number of data. This is called Runge phenomenon and unfortunately it can happen in situations where data is collected on an equally spaced grid.

We can avoid Runge phenomenon by using data at points that are not equally spaced. The data spacing that gives the best approximation are the Chebyshev points. Chebyshev points are defined on the interval $[-1, 1]$ as $x_k = \cos\left(\frac{2k-1}{2n}\pi\right)$ for $k = 1, \ldots, n$. Here is a plot of them compared to an equally spaced grid:

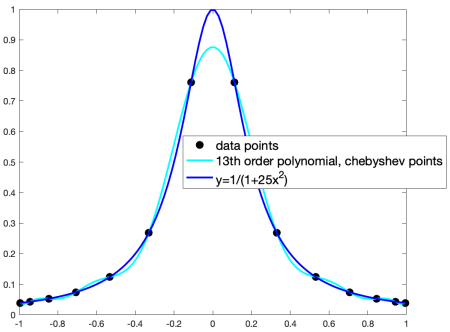


We see that the Chebyshev points are more clustered at the endpoints which gives some intuition as to why they help prevent Runge phenomenon. The Chebyshev points can be defined on any interval $[a, b]$ by $x_k = \frac{1}{2}(a + b) + \frac{1}{2}(b - a)\cos\left(\frac{2k-1}{2n}\pi\right)$ for $k = 1, \ldots, n$.

Let's use the Chebyshev points to interpolate $y = \frac{1}{1+25x^2}$ on the interval $[-1, 1]$ with the MATLAB commands

```
n=14;
f=@(x) 1./(1 + 25*x.^2);
x=cos((2.*[1:n]-1)*pi./(2*(n)));
y=f(x);
```

Then we get a good approximation



8.5 Splines

We can improve the behavior of the single polynomial interpolant by adjusting the location of the points where we interpolate as we did with Chebyshev points. However, if we do not have a choice as to where to place the data points then we are not able to interpolate all of the data. Another option is to construct piecewise polynomials through subsets of the data through **spline interpolation**.

The most common approach to spline interpolation is to create separate cubic polynomials

$$S_i(x) = a(x - x_i)^3 + b(x - x_i)^2 + c(x - x_i) + d$$

over subintervals $[x_i, x_{i+1}]$ with $S_i(x_i) = y_i$ and $S_i(x_{i+1}) = y_{i+1}$. Note that there are four unknowns in each subinterval and only two interpolation points. A square system of equations is formed by adding continuity conditions such as $S_i(x_{i+1}) = S_{i+1}(x_{i+1})$ and $S_i'(x_{i+1}) = S_{i+1}'(x_{i+1})$. The result is a tridiagonal linear system that is solved for coefficients of each of the i cubic polynomials that make up the interpolating spline over the whole interval.

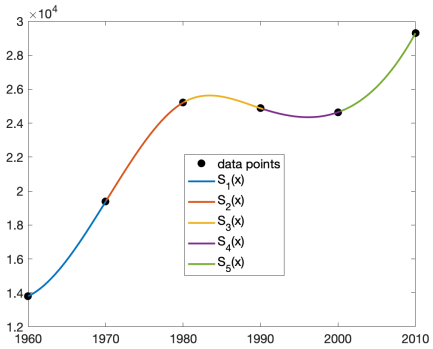
As an example, consider the census data from Billing, Montana in ten year intervals from 1960 to 2010. Since there are six data points there are five cubic polynomials in the spline interpolant. We find the coefficients for each of the five cubics all at once using the *spline* function in MATLAB. Let's consider only the white population:

```
C_table=readtable('census.txt');
C_array=table2array(C_table);
x=C_array(:,1);
y=C_array(:,2);
p_spline = spline(x,y);
```

The coefficients a, b, c, d for each piecewise polynomial are in p_spline. Given these coefficients we can find the interpolating polynomial over the whole interval using *ppval* . As usual we will evaluate the polynomial at 100 points to get a continuous looking interpolant

```
a=min(x);b=max(x);
xs = linspace(a,b,100);
y_spline = ppval(p_spline,xs);
```

The following is a graph of y_spline. To the naked eye our result looks like the fifth order interpolant even though y_spline is a series of cubic polynomials rather than one single fifth order interpolant. I've highlighted each piecewise cubic polynomial in the graph, but normally you would just plot(xs,p_spline) and not highlight the fact that there are different polynomials on each subinterval.



You can predict the population in 2005 by typing

y_spline = ppval(p_spline,2005);

and the prediction is 26,136 people.

The piecewise polynomial found by spline is returned as a structure in p_spline. Since the functions *spline* and *ppval* are compatible we didn't have to think about this when creating and plotting the interpolating polynomial. However, if we want to understand how many piecewise polynomials are in the whole interpolating polynomial and possibly values their coefficients, we'll need to understand the structure. The structure has the following fields which can be displayed by typing p_spline:

```
p_spline =

  struct with fields:

      form: 'pp'
    breaks: [1960 1970 1980 1990 2000 2010]
     coefs: [5×4 double]
    pieces: 5
     order: 4
       dim: 1
```

A description of each part of the structure is given in the following table

output	meaning
`form: 'pp'`	the form is a piecewise polynomial
`breaks: [1960 1970 1980 1990 2000 2010]`	the start and end of each S_i interval
`coefs: [5x4 double]`	each row contains the coefficients of the ith polynomial S_i
`pieces: 5`	number of piecewise polynomials on the interval
`order: 4`	order of the polynomial
`dim: 1`	dimension of your interpolating polynomial

You access each part of the structure by typing p_spline.form, p_spline.breaks, p_spline.coefs etc.

For example, p_spline.coefs is a 5x4 matrix and we access the second row by typing p_spline.coefs(2,:). The result is

```
ans =

   1.0e+04 *

  -0.000165453333333   0.000135000000000   0.073565333333333   1.937300000000000
```

With this information we can form $S_2(x)$ that is defined on the interval $[1970, 1980]$

$$S_2(x) = -1.655(x - 1970)^3 + 1.35(x - 1970)^2 + 735.65(x - 1970) + 19,373.$$

8.6 Individual Lab

Part 1

1. Given the data points (-2,4) and (-1,3) identify the Vandermonde matrix.
2. Given the data points (-2,4), (-1,3) and (0,5) identify the Vandermonde matrix.
3. Given the data points (-2,4) and (-1,3) identify the Lagrange polynomials.
4. Given the data points (-2,4), (-1,3) and (0,5) identify the Lagrange polynomials.

Part 2

1. The following are Chebyshev points on the interval $[-1, 1]$.
2. The following are Chebyshev points on the interval $[2, 3]$.
3. The following are values of $y = e^{-x^2}$ evaluated at the Chebyshev points on the interval $[-1, 1]$.

Part 3

Use the Census data from Billings, Montana to interpolate the population of black people from 1960 to 2010 and answer the following questions:

1. How many piecewise polynomials are defined on the interval $[1960, 2010]$?
2. The formula for the piecewise cubic spline interpolant is the same as the formula for the fifth degree polynomial interpolant.
3. The graph of the piecewise cubic spline interpolant looks the same as the graph for the fifth degree polynomial interpolant.
4. The coefficients for the cubic spline on the interval $[1980, 1990]$ are:

8.7 Group Work

You may work in groups or individually to answer the following questions.

Part 1

1. (a) Use the *vander* command in MATLAB to create the Vandermonde system with the census data in census.txt. You may type `>>` help vander at the MATLAB prompt to learn how to use the function, or Google it.
   (b) Find the coefficients for polynomial interpolation of the total population. The backslash command \ solves a linear system of equations in MATLAB. For example `x = A\b` solves the system of linear equations $Ax = b$.

   Be prepared to discuss: (1) inputs to the MATLAB function *vander*, (2) any issues you had with solving the system of equations with the \ command and (3) the degree of the polynomial that interpolates the data.

2. Plot the data and the interpolating polynomial on the same graph and consider the following

   (a) Test your understanding of polynomial interpolation by coding the fifth degree polynomial defined by the coefficients you found in 9.2, i.e. $p_5(x) = a_0 + a_1x + \ldots a_5x^5$.
   (b) Note that the interpolating polynomial is defined at infinitely many points x. To get a good view of it you will need to create a linearly spaced vector `x` with a 100 or so points. You can do this in MATLAB with `linspace(min(xdata),max(xdata),100);` where `xdata` are the x values of the data points.
   (c) Try using the MATLAB function *polyval* to create $p_5(x)$ rather than typing it out yourself. Type `>>` help polyval at the MATLAB prompt to learn how to use the function, or Google it.

   Be prepared to discuss: (1) the difference between plotting the data points and the polynomial (2) how you chose the x-values in your plot, (3) how plotting helps you see if you have the right answer or not (4) how the coefficients a_i are arranged in the Vandermonde matrix and *polyval* and (5) any error messages you got from MATLAB.

3. Download the file lagrange.m into the directory where you are running MATLAB. You will use this function to find the same $p_5(x)$ as in 2a but use Lagrange interpolation so that it will be in the form $p_5(x) = y_1l_1(x) + y_2l_2(x) + \ldots + y_6l_6(x)$. The inputs into the function `lagrange(xdata,x,j)` are (i) `xdata` - the x values of the data points, (ii) `x` - the linearly spaced vector of 100 or so x values and (iii) `j`, the jth polynomial $l_j(x)$.

   (a) Use lagrange.m to find $l_4(x)$ and plot it.
   (b) Use lagrange.m to find $l_2(x)$ and plot it on the same graph as $l_4(x)$. Use a legend and upload your plot in pdf, jpg or png format into Blackboard. Do not upload a file in .fig format.
   (c) Write a loop that finds all six $l_j(x)$ and sums them up to form $p_5(x)$ which is called `px` in this code segment:

```
px = 0;
for j = 1:6
    px = px + lagrange(xdata,x,j)*y(j);
end
```

where `y(j)` are the total population data points.

(d) Plot the data with points and the interpolating polynomial with a line.

Be prepared to discuss: (1) What the graph of $l_2(x)$ and $l_4(x)$ as compared to the graph of $p_5(x)$, (2) how the number of data points relates to the degree and number of Lagrange polynomials, (3) issues with writing a loop that calls lagrange.m many times and (4) any issues you had with plotting the data points and fitted polynomial.

Part 2

1. The dataset in the file air_data_day.txt available in Blackboard contains the hourly concentration of PM2.5 in micrograms per cubic meter for a particular measuring station in Salt Lake County for the first day of 2016. PM2.5 is the concentration in the air of fine particulates. The Environmental Protection Agency (EPA) set the threshold the of 35 micrograms per cubic meter, which corresponds to an Air Quality Index (AQI) of 101. If the concentration of PM2.5 exceeds this threshold the air is considered unhealthy for sensitive individuals or populations.

   (a) Load the dataset into MATLAB and plot the data.

   (b) Use Lagrange interpolation to find and plot the interpolating polynomial. Use a legend and upload your plot of the data and interpolating polynomial in pdf, jpg or png format into Blackboard. Do not upload a file in fig format.

   Be prepared to discuss: (1) When looking at the plot of just the data, identify a function(s) that would model the data well, (2) the order of the polynomial that interpolates the data in air_data_day.txt (3) the reliability of your interpolating polynomial, i.e. if it is reliable, why? If it is not reliable, why not?

2. Your plot in 1b should show Runge phenomenon. Interpolation at the Chebyshev points would fix this problem but data at these locations is not possible because the data is collected on the hour, in even intervals. However, we can still get a better approximation if we only use the data points that are close to the Chebyshev points.

   (a) The following MATLAB code segment will find N points in the data that most closely match measurements at the Chebyshev points and store them in (x_new,y_new). Note that N should be less than the number of data points in the original data set (x,y) which lies on the interval x∈[a,b].

   ```
   x_cheb = (a+b)/2 + (b-a)/2*cos(pi/N*((1:N)-.5));
   [minValue,closestIndex] = min(abs(bsxfun(@minus,x_cheb, x')));
   x_new=x(closestIndex);
   y_new=y(closestIndex);
   ```

   Find the set of 5, 10, 15, and 20 points that most closely match data at the Chebyshev points. Identify if data points are repeated in any of the sets. Recall from the lab the formulae for Lagrange interpolation and be prepared to discuss what happens when you use a data set that has repeated data points.

   (b) Find the interpolating polynomial at the $N = 5, 10, 15, 20$ data points you found in 2a. Plot the polynomials along with the original data set. Use a legend and upload your plot of the data and interpolating polynomials in pdf, jpg or png format into Blackboard.

Do not upload a file in fig format. Be prepared to discuss which polynomial you think best approximates the data and why. Consider both how well the polynonmial you recommend represents the data and also how well it can predict future air quality.

(c) You'll notice that your polynomial in 2b does not go through all of the data points. This means that there are errors, or residuals, at the data points.

i. Evaluate the polynomial interpolant at all data points. You can do this in MATLAB with the following code segment

```
px_data = 0;
for j = 1:N
    px_data = px_data + lagrange(x_new,xdata,j)*y_new(j);
end
```

where (x_new,y_new) are the N points that most closely match Chebyshev points and xdata are the original data points. The value of the polynomial interpolant at all data points is stored in px_data.

ii. Use your values of the polynomial interpolant at all data points to find and plot the error, or residuals at all the data points when $N = 20$.

iii. Use the MATLAB function *subplot* to plot the error for all values of N

```
subplot(2,2,1),plot(xdata,resid_5),title('N=5')
subplot(2,2,2),plot(xdata,resid_10),title('N=10')
subplot(2,2,3),plot(xdata,resid_15),title('N=15')
subplot(2,2,4),plot(xdata,resid_20),title('N=20')
```

Upload your figure into Blackboard.

iv. Write MATLAB functions that calculates the variance in the residual as we did for linear regression. The function should takes in the original values of the data points (y) and the value of the interpolating polynomial at the times the data were collected (x).

```
function sigma2= var_resid(ydata,px_data)
        .
        .
        .
```

Use this functions to find the variance in the residual with $N = 5, 10, 15, 20$ data points.

Chapter 9

Curve Fitting

9.1 Introduction

Curve fitting is closely related to interpolation. Both are used to study how well a specific experiment's data is consistent with conventional ideas about the phenomena. For example, population growth often occurs exponentially. The process of interpolation or curve fitting involves using data to find coefficients or parameters in the model that describes our pre-conceived notion about the how the observed phenomena is expected to behave.

Curve fitting and interpolation differ in that interpolation is used to get an exact fit to data points while curve fitting typically only approximately matches the data. Recall for interpolation we find a unique polynomial of degree n given $n+1$ data points. If the the data set is large we can find a polynomial that fits a subset of the data. Curve fitting on the other hand, may produce a curve that doesn't fit any of the data.

9.2 Least squares curve fitting

The most common way to find coefficients or parameters in a mathematical model it to use the least squares method that minimizes the distance between the data and curve. Let's begin curve fitting with fitting the Billing's, MT census data to a polynomial. The (x, y) data points for the hispanic population are (1980, 781), (1990, 871), (2000, 1257), (2010, 1835). If we chose to interpolate all of the data the result would be a cubic polynomial (why?).

9.2.1 Quadratic curve

Rather than interpolate the census data, let's fit it to the quadratic polynomial

$$y(x) = c_0 + c_1 x + c_2 x^2.$$

This results in the non-square system of equations

$$\begin{bmatrix} 781 \\ 871 \\ 1257 \\ 1835 \end{bmatrix} = \begin{bmatrix} 1 & 1980 & 1980^2 \\ 1 & 1990 & 1990^2 \\ 1 & 2000 & 2000^2 \\ 1 & 2100 & 2100^2 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \\ c_2 \end{bmatrix}$$

or

$$\mathbf{y} = \mathbf{A}\mathbf{c}.$$

There are not a unique coefficients c_0, c_1, c_2 that satisfy this system of equation because there are more equations than unknowns. Instead let's find a set of coefficients $\mathbf{c}$ and form

$$\hat{y}(x) = c(1) + c(2)x + c(3)x^2 \tag{9.1}$$

so that $\hat{y}(x_i) \approx y_i$. This means there will be residuals or errors in our quadratic polynomial.

In least squares curve fitting we find coefficients $\mathbf{c}$ that minimize the sum of squared errors or residuals

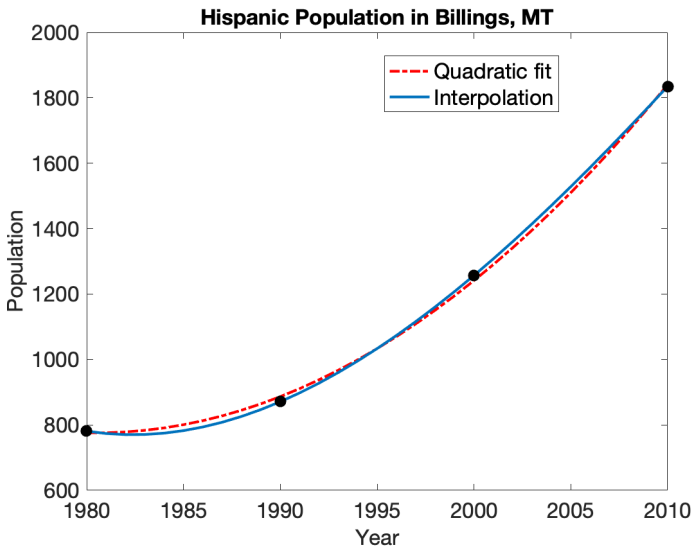
$$S = \sum_{i=1}^{n} (y_i - \hat{y}(x_i))^2.$$

The coefficients in $\mathbf{c}$ that minimize S are given by

$$\mathbf{c} = (\mathbf{A}^T\mathbf{A})^{-1}\mathbf{A}^T\mathbf{y}. \tag{9.2}$$

With most software packages you will get a more accurate answer if you find $\mathbf{c}$ by solving the system of equations $\mathbf{A}^T\mathbf{A}\mathbf{c} = \mathbf{A}^T\mathbf{y}$ rather than computing $(\mathbf{A}^T\mathbf{A})^{-1}$ and using it to find $\mathbf{c}$.

Here is a plot of the data, the 3rd degree interpolating polynomial, and the curve that results when the data are fit to the quadratic polynomial (9.1):



The graph shows that there is not a significant difference between the cubic and quadratic polynomial models. We can find when they differ the most, and the corresponding predictions with the following commands:

```
[maxValue,maxIndex] = max(abs(yhat-px));
```

```
year=xs(maxIndex)*10+1960;
quad_est=yhat(maxIndex);
interp_est=px(maxIndex);
```

Note that values of the quadratic polynomial are located in yhat, the cubic interpolating polynomial is px, and each polynomial is evaluated at xs $\in [2, 5]$. The result of the commands are year = 1987, quad_est = 827.74 and interp_est = 808.19. Based on these values we conclude that the year the models differ the most is in 1987 when the quadratic curve fit predicts there were 828 hispanic people and the cubic interpolant predicts 808 people.

9.2.2 Exponential curve

Since we are dealing with population data, let's look at fitting the data to an exponential curve. Define the curve as

$$y(x) = c_0 + c_1 e^x.$$

Assume that we have transformed the years in the census data points so that they are (2, 781), (3, 871), (4, 1257), (5, 1835). When we fit these data to the exponential curve we get the non-square system of equations

$$\begin{bmatrix} 781 \\ 871 \\ 1257 \\ 1835 \end{bmatrix} = \begin{bmatrix} 1 & e^2 \\ 1 & e^3 \\ 1 & e^4 \\ 1 & e^5 \end{bmatrix} \begin{bmatrix} c_0 \\ c_1 \end{bmatrix}.$$

This can also be written as a matrix system,

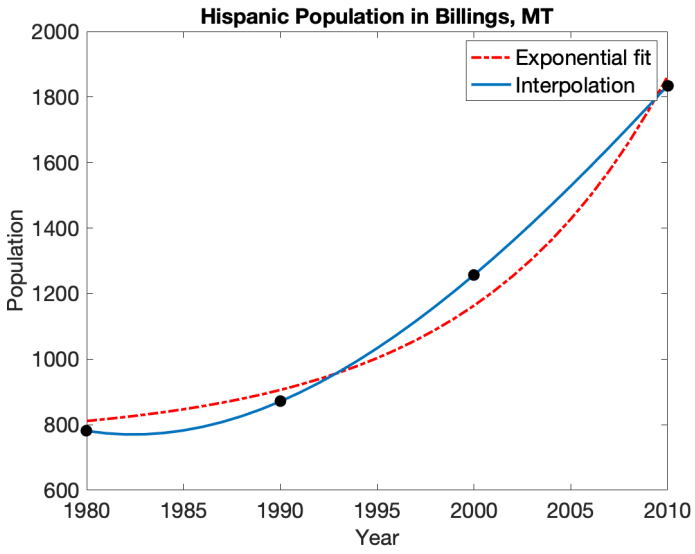
$$\mathbf{y} = \mathbf{A}_e \mathbf{c}_e$$

where the data vector $\mathbf{y}$ stayed the same as when we fit the data to a quadratic, but the matrix and corresponding coefficient vector changed because we changed the model to an exponential one.

We solve the system $\mathbf{A}_e^T \mathbf{A}_e \mathbf{c}_e = \mathbf{A}_e^T \mathbf{y}$ for the coefficients $\mathbf{c}_e$ and then form the model

$$\hat{y}_e(x) = c_e(1) + c_e(2)e^x. \tag{9.3}$$

Lastly, evaluate $\hat{y}_e(x)$ at a large number of values for x to get an idea of how well your model fits the data. Here is a plot of the data, the 3rd degree interpolating polynomial, and the curve that results when the data are fit to the exponential model (9.3):



Just by looking at the graph, it appears the exponential model (9.3) is the worse model for the hispanic population data, as compared to interpolation and the quadratic curve. This is because it does the worst job representing the data.

9.3 Rank and Pseudoinverse

The conditioning of the matrix $\mathbf{A}^T\mathbf{A}$ determines the stability of our fit to the curve $\hat{y}(x)$ or $\hat{y}_e(x)$. A problem is unstable if we change the data slightly and get a drastically different result for the coefficients $\mathbf{c}$. As we saw with interpolation it is best to transform the Census data years to smaller numbers by $(x - 1960)/10$ so that for the hispanic population, we have $x \in [2, 5]$. However, it is not always possible to transform the data to create a better conditioned matrix $\mathbf{A}^T\mathbf{A}$.

We cannot create a better conditioned matrix by transforming the data when $\mathbf{A}$ is not full rank. The rank of a matrix is the size of the largest collection of linearly independent columns or rows of $\mathbf{A}$. Columns or rows are linearly independent if none of them can be written as a linear combination of the others. Unfortunately many problems in real applications result in a rank deficient matrix because the model is not compatible with the data, or because there is not enough data to resolve the model. You can find the rank of a matrix in MATLAB by typing `rank(A)`. If $\mathbf{A} \in \mathbb{R}^{m\times n}$ a full rank matrix has rank $= \min(m, n)$, which means the matrix $\mathbf{A}^T\mathbf{A}$ is well-conditioned.

When $\mathbf{A}$ is not full rank we use the pseudoinverse to find the least squares estimate. A true inverse matrix satisfies $\mathbf{A}\mathbf{A}^{-1} = \mathbf{A}^{-1}\mathbf{A} = \mathbf{I}$. The pseudoinverse $\mathbf{A}^\dagger$ satisfies $\mathbf{A}\mathbf{A}^\dagger\mathbf{A} = \mathbf{A}$ In MATLAB we can find the least squares solution with the *pinv* command

```
c=pinv(A)*y
```

This function works both when $\mathbf{A}$ is full rank, and when it is not.

9.4 Epidemic Modeling

An epidemic model characterizes the spread of an infectious or contagious illness through a population. It requires parameters such as the current number of infected people and the probability a person will become infected. Data are used to identify these parameters and then the model can be used to understand the effect of different interventions, such as vaccinations. These results are often used to inform public health intervention decisions.

For the remainder of this section we make the following assumptions

- The number of infectious people at the start of day t is denoted by $I(t)$.
- The total population is denoted by N and we assume it is constant for all time.
- Once a person is *infected* they become *infectious* and stay *infectious* forever.
- If a person is infectious, they are equally likely with probability p to infect each noninfectious person on a given day.

9.4.1 Stochastic Modeling

Before using data to find parameters, let's assume we know the parameters and simulate how an illness spreads through a population. Let $x_n(t)$ be the infectious status of person n at the start of day t:

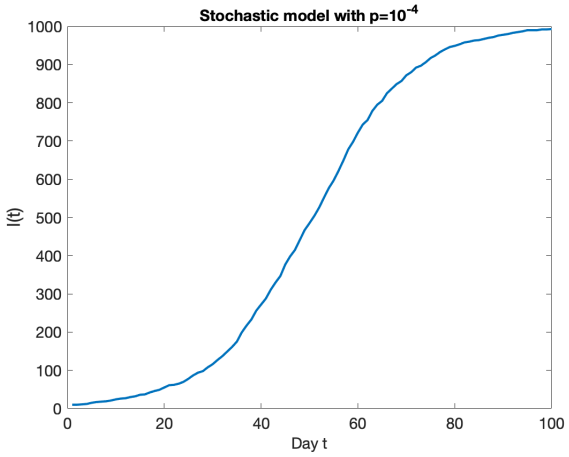
$$x_n(t) = \begin{cases} 1 & \text{if infectious} \\ 0 & \text{if not infectious} \end{cases}.$$

This means $I(t) = \sum_{n=1}^{N} x_n(t)$.

Let's build a stochastic model by assuming that the probability p a person becomes infected is known. Here is pseudocode to calculate the infectious status of the whole population over the time period $t = 1, \ldots, T$.

```
for t=1:T-1
    for i=1:N
        x(i,t+1)=x(i,t);   % Initialize infection status to be same as previous day
        for j=1:N
            find myrand   % Randomly choose a number between 0 and 1
            if myrand < p and x(j,t)=1
                x(i,t+1)=1   % Person i is infected by person j
            end
        end
    end
end
```

We view results from the stochastic simulation by plotting the number of infected people $I(t)$ over time t. Here are results of the first 100 days with $p = 10^{-4}$, a total population $N = 1000$, and an initial infected population of 10.



9.4.2 Deterministic Modeling

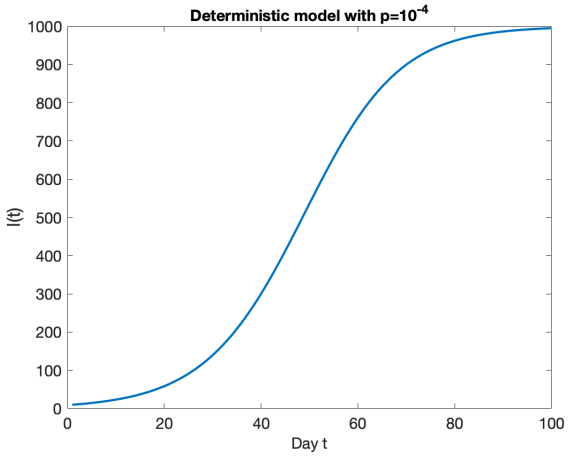
Each time we run the stochastic model we will get a different curve. That is because myrand is randomly chosen and we get a different value for it each time it is chosen. Rather than randomly choose if a person is infected, we can calculate the expected number of people infected on a day t. This results in the following deterministic model:

$$I(t+1) = I(t) + pI(t)\,(N - I(t)) \tag{9.4}$$

For these models, given $I(1)$ we can compute $I(2)$, $I(3), \ldots$

The deterministic models are much more efficient to compute than the stochastic models and their predictions may be just as reasonable. The advantage of the stochastic model is that it can give some idea of the uncertainty of its predictions via multiple simulations.

We view results from the deterministic model simulation by plotting the number of infected people $I(t)$ over time t. Here are results of the first 100 days with $p = 10^{-4}$, a total population $N = 1000$, and an initial infected population of 10.



This graph looks nearly identical to that created by the stochastic model. However, this curve is a bit smoother.

9.4.3 Continuous time modeling

The epidemic models we have discussed so far are called discrete-time models. They are discrete because time t takes on only integer values. Now we will approximate these models by continuous-time processes.

Consider that $I'(t) \approx \frac{I(t+1)-I(t)}{1}$. This means that (9.4) can be approximated by

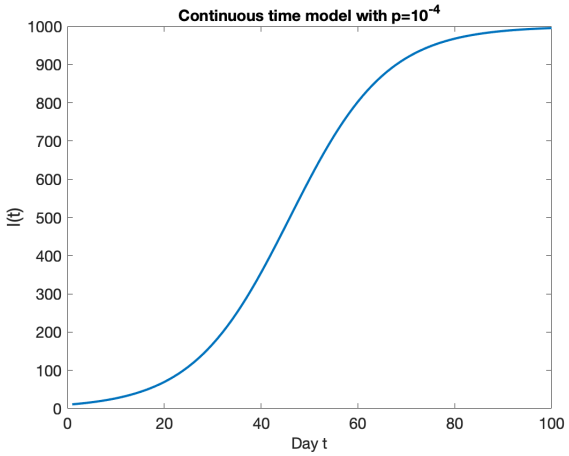
$$I'(t) = pI(t)\,(N - I(t)) \tag{9.5}$$

We can solve this differential equation exactly

$$I(t) = \frac{NI0}{I0 + (N - I0)\,e^{-pNt}} \tag{9.6}$$

where $I0$ is the initial population. I recommend differentiating (9.6) and verifying yourself that (9.5) is true.

We view results from the continuous time model simulation by plotting the number of infected people $I(t)$ over time t. Here are results of the first 100 days with $p = 10^{-4}$, a total population $N = 1000$, and an initial infected population of $I0 = 10$.



This graph looks identical to the one created with the deterministic model.

9.4.4 Fitting the model to data

In Sections 9.4.1-9.4.3 we input parameters representing the total population N, probability an infectious person will infect another person p and the initial number of infected people. Given these parameters all three models produced similar predictions of how the epidemic will spread over the next 100 days. Now we will address the situation where we want to use data to find values for these parameters.

Given a set of data points (t_j, I_j), $j = 1, \ldots, n$ representing the number of infected people on a specific day the goal is to find values for the parameters N, p and the number of people infected initially. We could try different values of the parameters, simulate the model, and see if we can find a set that produces a curve that looks like the data. While trying different values for the parameters may be a good way to understand how an epidemic could propagate, this approach is

not very practical. This is because we may never find a curve that matches the data, or never be sure if our curve is "close" enough to the data.

The most common way to find parameters using data is to use the same least squares curve fitting from Section 2 and in the Regression lab. In the case of epidemic modeling, we want to find parameters N, p and $I0$ (initial number of infected people) that minimize the sum of squared errors or residuals

$$S = \sum_{j=1}^{n} (I_j - I(t_j))^2. \tag{9.7}$$

We can do this for the deterministic and continuous time models. Note that there is no closed form expression for $I(t)$ in the stochastic model. There are approaches to finding parameters using a least squares fit when there is no closed form expression for the model, but we will not cover them in this class.

Recall formula (9.2) that gives parameters that minimize S. If we are to use formula (9.2) we need to write (9.4) or (9.6) as a matrix system like

$$\begin{bmatrix} I_1 \\ I_2 \\ \vdots \\ I_n \end{bmatrix} = \mathbf{A} \begin{bmatrix} N \\ p \\ I0 \end{bmatrix}. \tag{9.8}$$

For the deterministic model (9.4) we could write

$$I(t+1) - I(t) = pI(t)\,(N - I(t)) \tag{9.9}$$

and use differences of data points for the left hand side of (9.8). However, the right-hand side of (9.8) can be written as a linear function of p or N, but not both. It is also not clear how to find $I0$. If we let $t_1 = 0$ for the first data point, then we could let $I_1 = I0$. However, this might not be the best choice of $I0$ in order to make the residuals $I_j - I(t_j)$ small.

The problem is that we cannot find a matrix $\mathbf{A}$ for (9.8) because the model is not linear in the parameters. This leaves us with two choices: re-formulate the problem so that it is linear or use nonlinear least squares. We will explore both.

Linear deterministic model

Let's assume we know the total population N and that the initial number of infected people is given by the data I_1. We will use data (t_j, I_j), $j = 1, \ldots, n$ to find p that minimizes (9.7).

Plugging data into (9.9) gives the following system of equations

$$\begin{bmatrix} I_2 - I_1 \\ I_3 - I_2 \\ \vdots \\ I_n - I_{n-1} \end{bmatrix} = \begin{bmatrix} I_1(N - I_1) \\ I_2(N - I_2) \\ \vdots \\ I_{n-1}(N - I_{n-1}) \end{bmatrix} \begin{bmatrix} p \end{bmatrix}.$$

Now we can use formula (9.2), or more accurately solve the corresponding system of equations, to find the single parameter p.

Given p we can use the deterministic model to predict the number of infected people. Once you do this, graph the simulation along with the data to see how well your curve fits the data. Since we used least squares to find the fit there's a good chance your curve will not go through all of the data points.

Linear continuous time model

Let's assume we know the total population N, and we use data (t_j, I_j), $j = 1, \ldots, n$ to find the initial number of infected people and p. We will make a transformation of variables to get a transformed model that depends linearly on its parameters.

Let $Z(t) = \log(N/I(t) - 1)$, then the continuous time model (9.6) becomes

$$Z(t) = Z0 - pNt. \tag{9.10}$$

The model (9.10) depends linearly on pN and $Z0$. If we plug data into (9.10) we get

$$\begin{bmatrix} Z_1 \\ Z_2 \\ \vdots \\ Z_n \end{bmatrix} = \begin{bmatrix} 1 & -t_1 \\ 1 & -t_2 \\ \vdots & \vdots \\ 1 & -t_n \end{bmatrix} \begin{bmatrix} Z0 \\ pN \end{bmatrix}.$$

There are a few things to keep track of with this transformation

- Begin the procedure by transforming the data (t_j, I_j) to (t_j, Z_j) where $Z_j = \log(N/I_j - 1)$
- The vector of solutions contains pN so don't forget to divide by N to get p.
- The vector of solutions contains $Z0$ so transform back to get the initial population: $I0 = \frac{N}{1+e^{z0}}$.

Given p and $I0$ we can use the continuous time model to predict the number of infected people. Once you have values for $I(t)$, again graph them along with the data to see how well your curve fits the data. Since we used least squares to find the fit there's a good chance your curve will not go through all of the data points.

Nonlinear least squares

Nonlinear least squares uses the same sum of squares in (9.7) to find parameters that best fit the data. However, when $I(t)$ is nonlinear in the parameters, we don't have an analytical expression for parameters that minimize the sum of squares, like (9.2).

Methods for finding parameters that minimize (9.7) in the nonlinear situation are beyond the scope of this class. Instead we will use the MATLAB function *lsqcurvefit*. I recommend you read the MATLAB documentation for it. This function requires the following inputs:

- An initial estimate for the parameters. Put the parameters in a vector $\mathbf{c} = [I0, p, N]^T$ and call initial estimates c0.

- The data, let's call them (tdata, Idata).
- A function for $I(t)$. For the deterministic model this would be

```
function Id = Idfun(c,t)
    I0=c(1); p=c(2); N=c(3);
    T=t(end);
    Id=zeros(T,1);Id(1)=I0;
    for t=1:T-1
        Id(t+1)=Id(t)+p*Id(t)*(N-Id(t));
    end
end
```

  For the continuous time model this would be

```
function Ic = Icfun(c,t)
    I0=c(1); p=c(2); N=c(3);
    Ic=N*I0./(I0+(N-I0)*exp(-p*N*t));
end
```

  Pass the $I(t)$ function into the *lsqcurvefit* function by using the function handle `@` in the input argument.

Given a set of data (tdata,Idata), the following MATLAB commands estimates the parameters in a vector c.

```
c0 = [10, 1e-4, 1000];
c = lsqcurvefit(@Idfun, c0, tdata, Idata)
```

Similar commands can be given to find the parameters in the continuous time model.

Once you've found the parameters, plug them into the deterministic or continuous time model to predict the number of infected people. When you graph your curve for $I(t)$ along with the data there's still a chance your curve will not go through all of the data points.

9.5 Individual Lab questions

Please answer the following questions in Blackboard.

Part 1

1. Assume we fit the census data (2, 781), (3, 871), (4, 1257), (5, 1835) to the quadratic curve $y(x) = c_0 + c_1x + c_2x^2$ by solving $\mathbf{A}^T\mathbf{A}\mathbf{c} = \mathbf{A}^T\mathbf{y}$. Find $\mathbf{A}^T\mathbf{A}$.
2. Assume we fit the census data (2, 781), (3, 871), (4, 1257), (5, 1835) to the exponential curve curve $y(x) = c_0 + c_1e^x$ by solving $\mathbf{A}_e^T\mathbf{A}_e\mathbf{c}_e = \mathbf{A}_e^T\mathbf{y}$. Find $\mathbf{A}_e^T\mathbf{A}_e$.
3. Assume we fit the census data of the hispanic population in Billings, MT to the quadratic curve $y(x) = c_0 + c_1x + c_2x^2$ by solving $\mathbf{A}^T\mathbf{A}\mathbf{c} = \mathbf{A}^T\mathbf{y}$. Use the quadratic curve to estimate the population in 1987.

4. Assume we fit the census data of the hispanic population in Billings, MT to the exponential curve $y(x) = c_0 + c_1 e^x$ by solving $\mathbf{A}_e^T \mathbf{A}_e \mathbf{c}_e = \mathbf{A}_e^T \mathbf{y}$. Use the exponential curve to estimate the population in 1987.

Part 2

1. For the stochastic model pseudocode on page 5, how would you specify the initial population of 10?
2. For the deterministic model in (9.4), how would you specify the initial population 10 in your code?
3. For the continuous time model in (9.6), how would you specify the initial population 10?
4. If $I(t) = \frac{NI0}{I0+(N-I0)e^{-pNt}}$ find $I'(t)$

Part 3

1. Assume we use data to estimate the parameter p in the linear deterministic model. The formula for p follows (9.2) as described in Section 9.4.4. For this situation, what is the dimension of the matrix $\mathbf{A}$ in (9.2)?
2. If $Z(t) = \log(N/I(t) - 1)$ then what is $I(t)$?
3. When fitting the curve to data, what are the unknown parameters in the linear deterministic model
4. When fitting the curve to data, what are the unknown parameters in the nonlinear deterministic model
5. When fitting the curve to data, what are the unknown parameters in the linear continuous time model
6. When fitting the curve to data, what are the unknown parameters in the nonlinear continuous model

9.6 Group Work

Part 1

1. Please write answers to the following questions on notecards.
   (a) Data for the black population in Billings, MT is (1960, 187), (1970, 74), (1980, 121), (1990, 159), (2000, 153), (2010, 326). Write the system of equations that results when you fit the data to the quadratic polynomial $y(x) = c_0 + c_1 x + c_2 x^2$.
   (b) Data for the black population in Billings, MT is (1960, 187), (1970, 74), (1980, 121), (1990, 159), (2000, 153), (2010, 326). Write the system of equations that results when you fit the data to the exponential curve $y(x) = c_0 + c_1 e^x$.

(c) Data for the black population in Billings, MT is (1960, 187), (1970, 74), (1980, 121), (1990, 159), (2000, 153), (2010, 326). Write the system of equations that results when you fit the data to the exponential curve $y(x) = c_0 + c_1 e^x + c_2 \frac{e^x}{1+x}$.

2. Consider the following integral

$$erf(t) = \frac{2}{\sqrt{\pi}} \int_0^t e^{-x^2} \, dx.$$

It is called the error function and arises in the analytical solution of the heat equation, a partial differential equation in physics. It can also be used to express the standard normal distribution if we evaluate it at $t/\sqrt{2}$ rather than t. There is no analytical solution for the integral and people often use a table to look up values for it.

(a) Create a your own "table" of values by generating 11 data points $y_k = erf(t_k)$ with $t_k = (k-1)/10$, $k = 1, \ldots, 11$. Use the MATLAB function *erf* to generate the data points.

(b) Fit the data to a polynomial of degree 4:

$$p_4(t) = c_0 + c_1 t + c_2 t^2 + c_3 t^3 + c_4 t^4,$$

i.e. find least squares estimates for $\mathbf{c}$ by solving $\mathbf{A}^T\mathbf{A}\mathbf{c} = \mathbf{A}^T\mathbf{y}$. This means you need to form the matrix $\mathbf{A}$, data vector $\mathbf{y}$, and use the backslash operator to solve for $\mathbf{c}$.

(c) Assume the MATLAB function *erf* finds the exact solution. Plot the error $|p_4(ts) - erf(ts)|$ where $ts = 0 : 0.01 : 1$.

(d) Polynomials are not good basis function with which to approximate $erf(t)$. Using the same data as in 2a, fit it to the curve

$$f(t) = c_0 + e^{-t^2}(c_1 + c_2 z + c_3 z^2 + c_4 z^3), \quad z = \frac{1}{1+t}.$$

Similar to 2b Form the matrix $\mathbf{A}$, data vector $\mathbf{y}$ and use the backslash operator to solve for $\mathbf{c}$.

(e) Again assume the MATLAB function *erf* finds the exact solution. Plot the error $|f(ts) - erf(ts)|$ where $ts = 0 : 0.01 : 1$.

(f) Use your function $f(t)$ to evaluate $erf(.05)$.

Part 2

1. Implement the pseudocode on page 5.

(a) Things to think about: (i) Blackboard question 1. explains how to specify the initial population, (ii) use the MATLAB function *rand* to find a scalar random number between 0 and 1 (iii) the *if* statement in the pseudo code has two conditions, use the & command to make sure both are satisfied (iv) test the equality in the *if* statement using ==, not = (v) use the *sum* command in MATLAB to find the total number of infected people.

(b) Run and graph multiple simulations on the same plot. Use the same values for p, N and $I(1)$ as in the lab.

(c) Try different values for the number of people infected initially.

Be prepared to discuss (i) why do you get different graphs each time you run the stochastic model, (ii) the magnitude of the difference between multiple simulation of the stochastic model with the same parameters (iii) how the graph changes when you change the number of people initially infected.

2. Implement the deterministic model in equation (9.4) and plot the number of infected people vs time. Use the same values for p, N and $I(1)$ as you did for the stochastic model.

   Be prepared to discuss the complexity of the code and the amount of time it takes to compute the stochastic model vs the deterministic model.

3. Implement the continuous time model in equation (9.4) and plot the number of infected people vs time. Use the same values for p, N and $I(1)$ as you did for the deterministic and stochastic model. Your plot should look the same as those for the stochastic and deterministic models.

   Be prepared to discuss the differences between your deterministic model code and your code for the continuous time model.

Part 3

1. Follow the instructions in the first part of the homework and load the monthly AIDS diagnoses data for the Boston area into MATLAB. Here are some tips: Once you download the data from the CDC I suggest first importing it into an Excel file and deleting all rows and columns that don't contain the number of infected people. Then use the MATLAB functions *readtable* and *table2array* to put it in an array. Be prepared to discuss if these data satisfy our assumptions for the model as described at the beginning of Section 4.

2. Use the MATLAB function *cumsum* to form a data vector **I** that contains the number of infected people. Plot the data and be prepared to discuss how well it appears the epidemic models we have been discussing are a good choice for this data set.

3. Use the deterministic model (9.4) to fit a curve to the data in the following manner:

   (a) Identify values for the total population N and the initial number of infected people $I0$.

   (b) Follow the instructions in 4.4.1 to estimate p. Be prepared to interpret your value for p.

   (c) Use your estimate of p and values for N and $I0$ you identified in 3a to form the curve for the deterministic model. Plot the curve and identify how well your curve fits the data.

4. Use the continuous time model (9.6) and fit the curve to the data in the following manner:

   (a) Form a function for the model that depends on N, p, and $I0$.

   (b) Read the documentation for the MATLAB function *lsqcurvefit*. Identify the following inputs: initial estimates of N, p, and $I0$, data values for the independent variable t, data values for the number of infected people.

   (c) Use *lsqcurvefit* to find estimates for N, p, and $I0$.

   (d) Use your estimates of N, p, and $I0$ to form the continuous curve. Plot the curve and identify how well your curve fits the data.