Battista, Jude

MA-350-1

Rempe

HW 10

4.1.C5

a)

A = [[3 -1 2];[4 1 0];[-3 2 1];[1 1 5];[-2 0 3]]

b = [10; 10; -5; 15; 0]

x = A\b

x = [2.5246; 0.6616; 2.0934]

r = b – Ax = [-1.0990; -0.7601; -0.8428; 1.3468; -1.2310]

Two norm of r can be obtained with norm(r) = 2.4135

b)

A = [[4 2 3 0]; [-2 3 -1 1];[1 3 -4 2];[1 0 1 -1];[3 1 3 -2]]

b = [10;0;2;0;5]

x = A\b

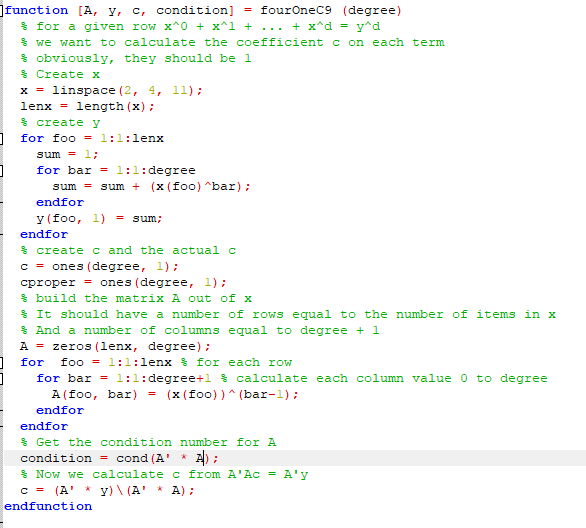
x = [1.2739; 0.6885; 1.2124; 1.7497]

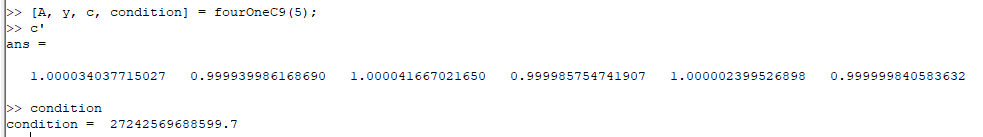
r = b – Ax = [-0.1099; -0.0550; 0.0110; -0.7367; 0.3518]

Two norm of r = 0.8256

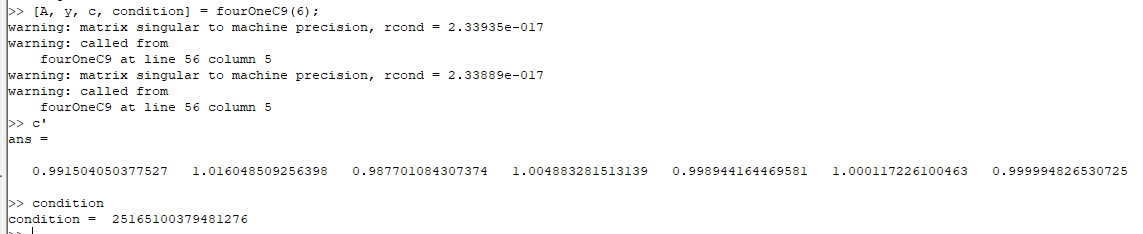
4.1.C9

All three parts to this question can be solved using the following function to calculate the coefficient vector:

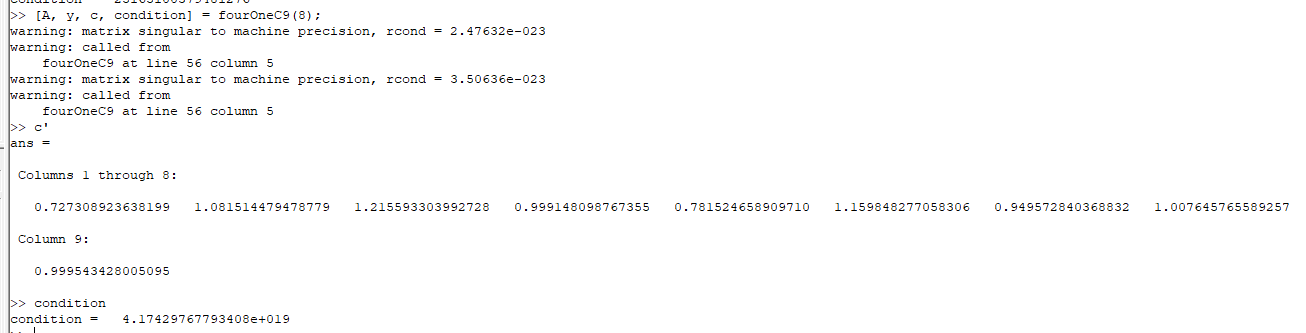
a) d = 5 yields:

Thus our coefficient calculation is correct to four decimal places.

b) d = 6 yields:

Thus our coefficient calculation is now only correct to one decimal place.

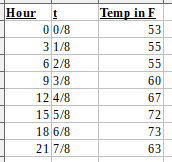
c) d = 8 yields:

Thus our coefficient calculation is now correct to zero decimal places.

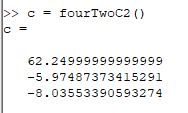
Overall, the decreasing accuracy of the calculated coefficients can be traced to the increasingly large condition values for the A-transpose \* A matrix.

4.2.C2:

Using the following weather data for Spokane on 2 May 2018 from weather underground:



And a c1 + c2 \* cos(2\*Pi\*t) + c3 \* sin(2\*Pi\*t) model, we get the following:



4.2.C8:

Using Vim to replace the ^M characters in the scrippsy.txt file gives us our b vector. After that, we can use a c1 \* exp(c2 \* t) = b model. This implies that ln (c1 \* exp(c2 \* t)) = ln b.

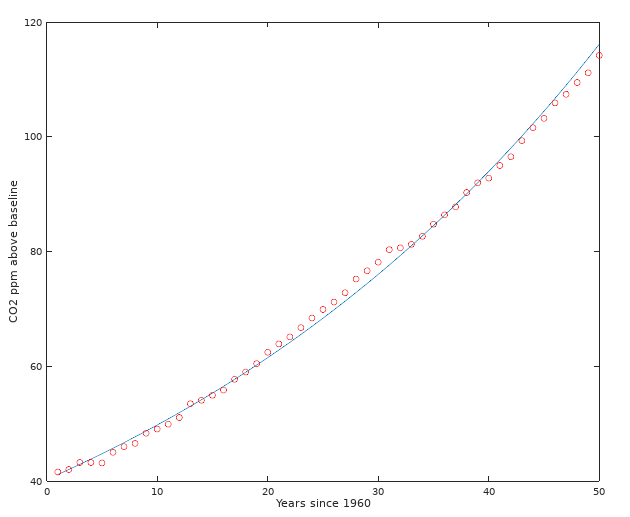
Therefore, ln(b) = ln(c1) + c2 \* t

Solving this using c = (A’\*A) \ (A’ b) we get the following set of coefficient values:

c = [3.6955332683400259; 0.0211885664261296]

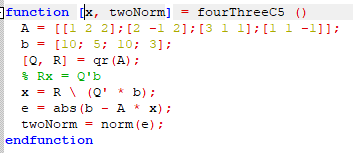
Thus c1 = exp(3.6955332683400259) = 40.2670400010962

And our model is 40.2670400010962 \* exp(0.0211885664261296 \* t) which gives us a plot of

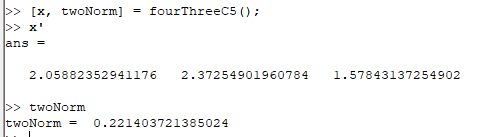


4.3.C5b:

We can use Octave’s qr function to factor A, then solve the equation Rx = Q’b for x:



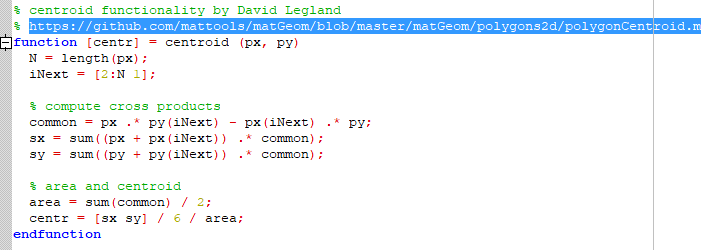
Which gives us the following x vector and error two norm:



13.1.C8:

There are a number of possible criteria for tolerance. I chose to work with the idea that the algorithm should stop when the polygon’s center moves less than a certain distance between steps. I used code from David Legland at <https://github.com/mattools/matGeom/blob/master/matGeom/polygons2d/polygonCentroid.m>

to calculate the centroid, rather than including the OctaveForge polygon library in order to maintain portability to Matlab. The helper function to do so is:

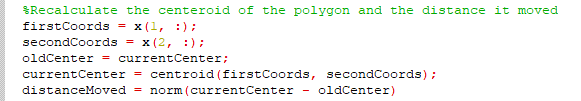


I update size at each iteration of the while loop as follows:



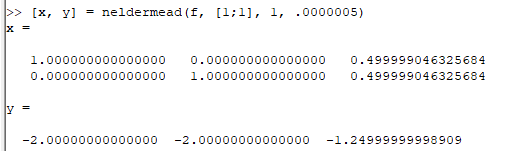
while this could have been done in a single step, it was easier to debug as is.

The update to the while loop criteria is then:

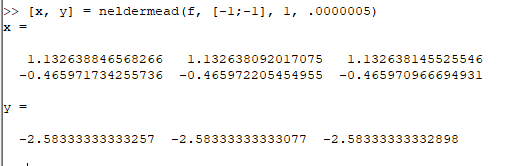


I have chosen not to reprint the full code since the vast majority is from the book. It is available at [\\CS1\2020\jbattista20\MA-350-1\MATLAB\neldermead.m](file://CS1/2020/jbattista20/MA-350-1/MATLAB/neldermead.m).

a) f = @(x) x(1)^4 + x(2)^4 + 2\*x(1)^2\*x(2)^2 + 6\*x(1)\*x(2) – 4\*x(1) – 4\*x(2) +1 gives us:

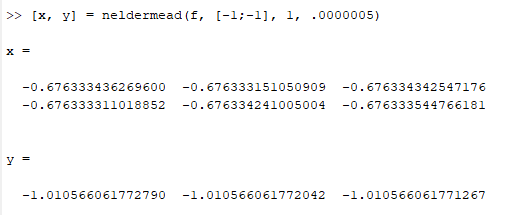


and

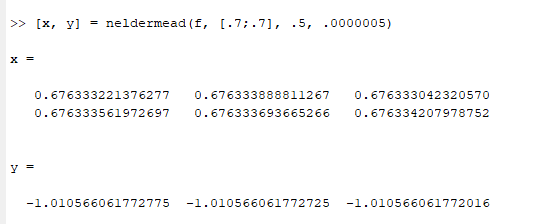


b) f = @(x) x(1)^6 + x(2)^6 + 3\*x(1)^2\*x(2)^2 – x(1)^2 – x(2)^2 – 2\*x(1)\*x(2)

If we plot the function, we can see that the minima should be around (-1, -1) and (0.7, 0.7), giving us:



and



Well, that’s odd. Further, varying the radius does not seem to change this result, nor does jostling the starting guess.

13.2.C3a:

f(x, y) = 100(y – x^2)^2 + (x-1)^2 starting at (2, 2)

f = (@x,y) 100\*x^4 - 200\*x^2\*y + x^2 – 2\*x + 100 \* y + 1

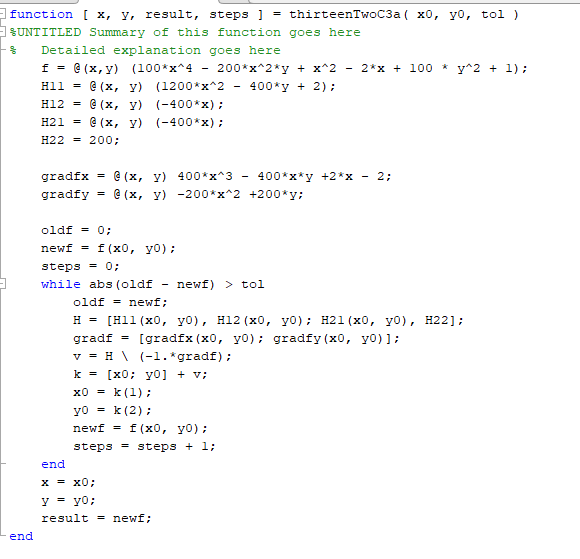
H = 1200\*x^2 - 400\*y -400

-800\*x 200

Gradf = [400\*x^3 - 400\*x\*y +2\*x - 2, -200\*x^2 +200\*y]

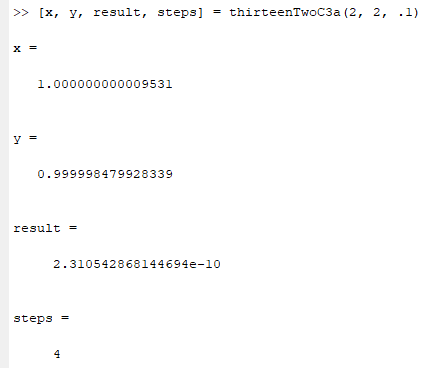
v = H(xk, yk) \ -Gradf( xk, yk)

We can automate this with the following function:



Our initial x and y values are fixed, but we can vary our tolerance to see when the solution remains relatively static.

Testing different values for the tolerance, we find that the algorithm rapidly converges, requiring only four steps to get within five decimal places of the correct value of (1, 1).



Using a much more stringent tolerance, we find that we can get within machine precision of the correct value of (1, 1) within only six steps:

