Journal Report 7

10/15/19-10/22/19
Jialin Tso
Computer Systems Research Lab
Period 2, White

Daily Log

Tuesday October 15

Wrote try except statement so that multiple digit numbers are correctly converted to word form; similar to word to number try except statement, but trickier.

Thursday October 17

Converted var stats, linreg, and integral function (however, with e represented as long number) to text.

Monday October 21

Tested that matrix and all other existing functions are converted to text. Went back through all old journals to test those cases again and found some small errors.

Tuesday October 22

Converted 2.718.... to e with if statement. Added commas between array of coefficients (e.g. changed numpy array to regular one). Since num2words doesn't recognize "point" and would not enter necessary try statement, added num2words(arr[i] + "0"). This is so that program can correctly convert decimals. Because "x" and "e" can both appear as symbols and in a word, wrote if statement to determine which case they fit in for the answer to be converted to words. Ran test cases all over again to ensure everything works properly.

Timeline

Date	Goal	Met
Sept 23	Solve matrices and find 1-var stats for user-inputted list.	Common matrix functions like addition, reduced row echelon, and determinant work. 1-var stats outputted.
Sept 30	Find regression equations from user-inputted lists	Linreg, Polyreg, and Expreg can generate equations and graphs.
Oct 7	Convert answers from expression to words and resolve problem of word numbers separated by dashes	No dashes needed between numbers. Many non-numerical answers can be displayed in words.
Oct 14	Convert all calculator-recognized function answers to words	Success!
Oct 28	Experiment with Speech Recognition and make sure it can accurately transcribe what I say to text;	

Reflection

I've made lots of progress with my code to the point where I believe all functions my program recognizes can output the correct answer in words. Depending on the function, there can be too many spaces between characters that aren't in words, but I don't think this is a large issue because Speech Recognition will hopefully ignore these extra spaces. A challenge I faced was that as I added new features to my code, all the other functions shouldn't be affected by this change. Initially, the integral function returned e as a long number (length 16!), which made the text output very long. I wrote a segment of code in my integral function that would convert this number to its symbol. In addition, my program was originally written so that it would erroneously return 0.016 as "zero point sixteen". I then wrote some code so that each individual number after the decimal point would be outputted as a single digit number. Now that I think of it, it would be a good idea for the program to recognize a number such as "1234" as "one two three four" instead of "one thousand two hundred thirty four".

Example of new input and output of current program: Input: var-stats left parentheses left-bracket one comma two comma three right-bracket right parentheses

Expression output: varstats([1,2,3])

Answer: ('Mean:2', 'Sum:6', 'SumSquares:14', 'Sx:1.0', 'Popstd:0.816496580927726', 'SampVar:1', 'Popvar:0.816496580927726', 'Size:3', 'Min:1', 'Q1:1.5', 'Median:2', 'Q3:2.5', 'Max:3', 'Mode:None')
Output: left parentheses Mean is two comma Sum is six comma SumSquares is fourteen comma Sx is one comma Popstd is zero point eight one six four nine six five eight zero nine two seven seven two six comma SampVar is one comma Popvar is zero point eight one six four nine six five eight zero nine two seven seven two six comma Size is three comma Min is one comma Qone is one point five comma Median is two comma Qthree is two point five comma Max is three comma Mode is None right parentheses

Input: integrate left parentheses x to the power of two times e to the power of x times cosine of left parentheses x right parentheses comma x right parentheses

Expression output: integral(x**2*e**x*cos(x),x)

Answer: 0.5*e**x*x**2*sin(x) + 0.5*e**x*x**2*cos(x) - 1.0*e**x*x*sin(x) + 0.5*e**x*sin(x) - 0.5*e**x*cos(x)

Output: zero point five times e to the power of x times x to the power of two times sin left

parentheses x right parentheses plus zero point five times e to the power of x times x to the power of two times cos left parentheses x right parentheses minus one point zero times e to the power of x times x times sin left parentheses x right parentheses plus zero point five times e to the power of x times sin left parentheses x right parentheses minus zero point five times e to the power of x times cos left parentheses x right parentheses

Input: reduced-row-echelon left parentheses matrix left parentheses left-bracket left-bracket one comma zero comma one comma three right-bracket comma left-bracket two comma three comma four comma seven right-bracket comma left-bracket negative one comma negative three comma negative three comma negative four right-bracket right-bracket right parentheses right parentheses

Expression output: rref(matrix([[1,0,1,3],[2,3,4,7],[-1,-3,-3,-4]]))

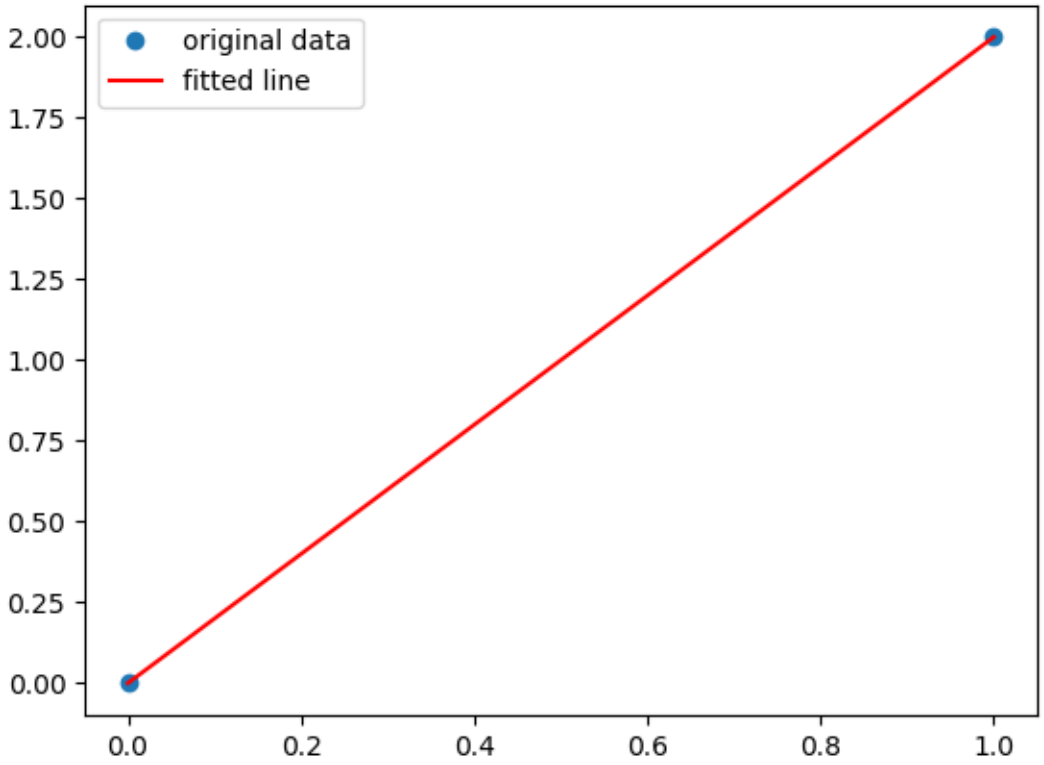
Answer: (Matrix([[1, 0, 1, 3],[0, 1, 2/3, 1/3],[0, 0, 0, 0]]), (0, 1))

Output: left parentheses Matrix left parentheses left bracket left bracket left bracket one comma zero comma one comma three right bracket comma comma left bracket zero comma one comma two divided by three comma one divided by three right bracket comma comma left bracket zero comma zero comma zero comma zero right bracket right bracket right parentheses comma left parentheses zero comma one right parentheses right parentheses

Input: linreg left parentheses left-bracket zero comma one right-bracket comma left-bracket zero comma two right-bracket right parentheses

Expression output: linreg([0,1],[0,2]) Answer: LinregressResult(slope=2.0, intercept=0.0, rvalue=1.0, pvalue=0.0, stderr=0.0) Output: LinregressResult left parentheses slope is two point zero comma intercept is zero point zero comma rvalue is one point zero comma pvalue is zero point zero comma stderr is zero point zero right parentheses

Graph:



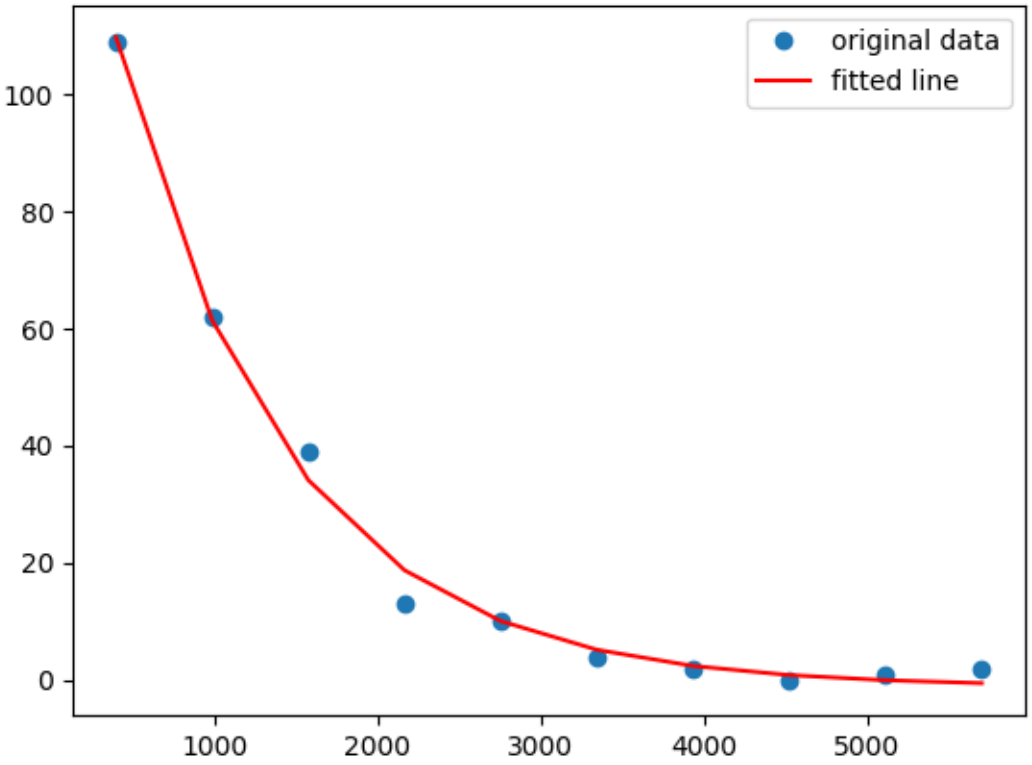
Input: exponential left left-bracket three hundred ninety nine point seven five comma nine hundred eighty nine point two five comma one thousand five hundred seventy eight point seven five comma two thousand one hundred sixty eight point two five comma two thousand seven hundred fifty seven point seven five comma three thousand three hundred forty seven point two five comma three thousand nine hundred thirty six point seven five comma four thousand five hundred twenty six point two five comma five thousand one hundred fifteen point seven five comma five thousand seven hundred five point two five right-bracket comma left-bracket one hundred nine comma sixty two comma thirty nine comma thirteen comma ten comma four comma two comma zero comma one comma two right-bracket right

Expression output: expreg([399.75,989.25,1578.75,2168.25,2757.75,3347.25,3936.75,4526.25, 5115.75,5705.25],[109,62,39,13,10,4,2,0,1,2])

Answer (coefficients of curve equation): [163.56165391511246, -0.00097114214889287072, -1.1685460560461085] (a*np.exp(b*x)+c format, different equation from calculator, but still pretty accurate)

Output: left bracket one hundred and sixty-three point five six one six five three nine one five one one two four five comma minus zero point zero zero zero nine seven one one four two one four eight eight nine two eight seven zero eight comma minus one point one six eight five four six zero five six zero four six one zero eight five right bracket

Graph:


original data
fitted line
100
80
60
40
20
0
1000
2000
3000
4000
5000