## 382 Project 1 - Summary File

### Exercise 1

1. We wrote the coef\_vander.m function for an input matrix of data vectors (xdata, ydata), and looped through them to find the coefficients of our approximate polynomial using the Vandemonde matrix method. We then tested our functions for given data fitting the y=x2 function and obtained the following coeff vectors:

[1 0 0], for xdata= [ 0 1 2 ] ydata= [ 0 1 4 ]

[1 -2 3 -4 5 -6 7], for xdata= [ -3 -2 -1 0 1 2 3] ydata= [1636 247 28 7 4 31 412]

1. Then we checked our results against MATLAB’s built-in polyfit function, using the command

(polyfit([-3 -2 -1 0 1 2 3],[1636 247 28 7 4 31 412],6))'

and obtained parallel results with a 10+3 coefficient. The entirety of the code is available in each of our diary files.

### Exercise 2

1. We inputted the given roots to the poly function and obtained

cTrue = [ 1 -3 -5 15 4 -12]

and using these coefficients, taking them to be accurate, deduced that our polynomial takes the value of c0=-12 at x=0.

1. Then we used this as the sixth data point our coef\_vande requires to obtain the coefficients of our polynomial approximation of degree 5. The result was the same as the values in cTrue, except in 4 decimal points and as a column, precisely:

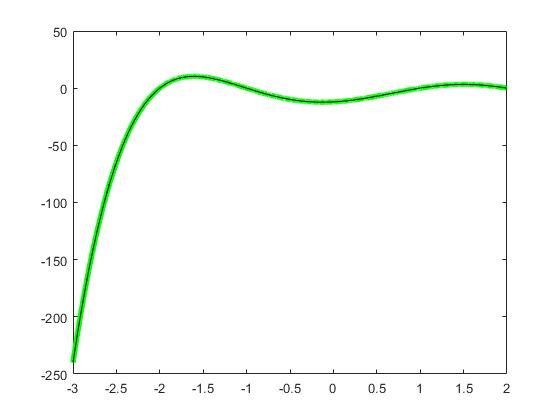
cVander’ = [ 1.0000 -3.0000 -5.0000 15.0000 4.0000 -12.0000 ]

1. Then we ran the coef\_vande function using only the five roots, and obtained a zero vector,

notcVander = coef\_vander([-2 -1 1 2 3], [0 0 0 0 0]) = [0 0 0 0 0]

which is reasonable since the only degree four polynomial passing through our five zero points would need to be the constant zero function.

1. Then we inputted the code provided to plot the polynomials on the interval [-3 , 2], where the green curve used the coefficients from the poly function and the black curve used the coefficients from the coeff\_vande function. In this interval, the plots seem to match perfectly.



### Exercise 3

1. The component functions fi(x) of our Lagrange polynomial lk(x) are given by   
   fi(x) = (x - xi)/(xk-xi)
2. Since their numerators are given by (x-xi) (and their denominators are nonzero constants), they take the value of zero at x=xi. As a product of these components for all i ≠ k, the Lagrange polynomial lk will be zero for all xi, ≠ xk. Furthermore at x=xk, each of the component functions fi will have equal numerator and denominator, so that fi(xk) = (xk - xi)/(xk - xi) = 1
3. We have constructed lagrangep function that “constructs” the kth Lagrange polynomial for a given set of x values (called xdata) and evaluates the polynomial at a given x value (called xval or xgiven). The code loops through values of i from the set {1, …, k-1, k+1, …,n}. At each step it evaluates the ith component function fi(x) at the given x value, and then multiplies it wth the result from previous steps.
4. We checked that our lagrangep function indeed gave 0 for all x = xi, ≠ x1 , and gave 1 for x= x1, for the arbitrary xdata= [0 1 2]. More precisely,

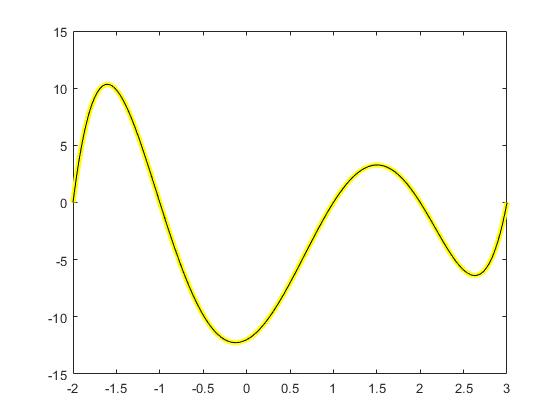
lagrangep(1, [0 1 2], 0) = 1  
lagrangep(1, [0 1 2], 1) = 0  
lagrangep(1, [0 1 2], 2) = 0

1. The confirmation in step 3. was simplified and generalized by taking xgiven to be the same vector as xdata, and checking that each k value returned a zero vector except for its kth component which was 1. More precisely, we obtained

lagrangep(1, [0 1 2], [0 1 2]) = [1 0 0]  
lagrangep(2, [0 1 2], [0 1 2]) = [0 1 0]  
lagrangep(3, [0 1 2], [0 1 2]) = [0 0 1],

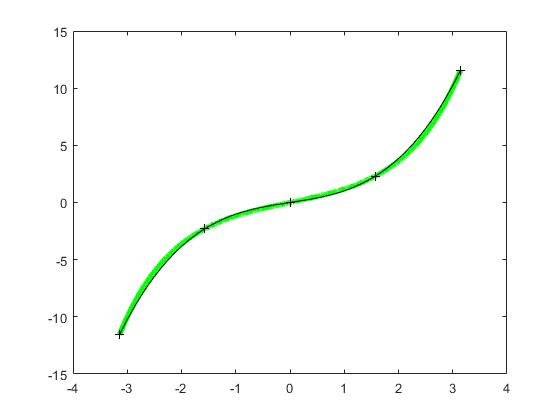
These results confirmed that our function indeed gave the right values for each xi, and satisfied the conditions of a Lagrange polynomial.

### Exercise 4

1. We produced a function for evaluating the lagrange polynomial interpolating a given set of points, and called it eval\_lag. Then we ran this function for the simple data points we used in previous steps, values of y=x2 at x = [0 1 2], and obtained yval= [0 1 4] as predicted. Similarly we ran the function for xdata = [ -3 -2 -1 0 1 2 3], ydata = [1636 247 28 7 4 31 412] and evaluated at the same x values. We again obtained the same y values as we were given, confirming that function satisfies interpolation conditions.
2. Then we repeated exercise 2 using eval\_lag instead of coef\_vander and got the following plot, demonstrating that our interpolation fits well in the interval [-3, 2]:

Yellow line depicts our interpolation using eval\_lag, and the black line depicts “true” function as interpolated by MATLAB’s polyval function.

### Exercise 5

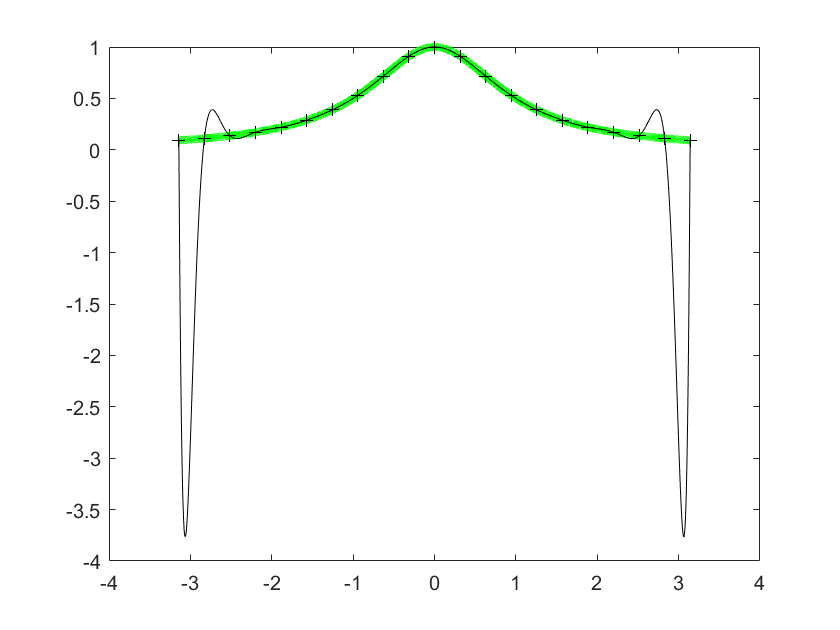
1. We have defined a new m file namely “exer5.m”, which uses the given script text in the exercise to construct the interpolating polynomial of on the interval by using many sample points. By default we let the to be . After completing the gaps in the script text, we have run the code and obtained the following figure: 

As we can see from the graph, the exponential function and its interpolant agree at the interpolation points.

1. By changing the value of (number of sample points for the interpolation) in the “exer5.m”, we have filled the following table. As we increase the sample points, our approximation error becomes smaller as we expected.

|  | Approximation Error = |
| --- | --- |
|  | Approximation Error = 6.6116e-06 |
|  | Approximation Error = 1.6550e-13 |

### Exercise 6

1. A new function m- file, namely runge.m, is created. This function takes input x, if it is a vector division and exponentiation is componentwise, and computes 1/(1+x^2) as y.
2. The code is directly copied to a new m.file named “exer6.m”. The sinh function in exer5.m is changed to the runge function defined before. ydata, the given y values of data points, takes its values from runge(xdata) similarly yvalTrue, the true test point values. The plot generated is as follows:
3. We can observe that the runge function and its interpolant agree at the interpolation points, but not necessarily between them as seen between the first and last pairs of interpolation points.

| N (number of data points) | Approximation Error |
| --- | --- |
| 5 | 0.313272549252914 |
| 11 | 0.584572917193051 |
| 21 | 3.860659576795580 |

We are not surprised to see that the approximation errors do not decrease with increasing number of data points, this is because the function we are trying to approximate is not close to a polynomial.

### Exercise 7

1. the code in the exercise 6 is copied and the line

yval=eval\_lag(xdata,ydata,xval);

is replaced with

yval=polyval(coef\_vander(xdata,ydata),xval);

to make the approximation using vandermonde matrix and not the lagrange interpolation.

1. The code is executed 3 times with increasing number of interpolation points, it’s observed that the approximation errors are the same as in exercise 6 upto 14 decimals.

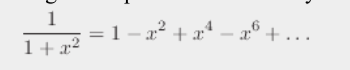
| N (number of data points) | Approximation Error |
| --- | --- |
| 5 | 0.313272549252913 |
| 11 | 0.584572917193054 |
| 21 | 3.860659576788524 |

1. Now, the nontrivial vandermonde coefficients of the interpolating polynomial are calculated as in the following table:

| N=5 | c(5)=1 | c(3)=-0.353867362739210 | c(1)=0.026532744753167 |  |
| --- | --- | --- | --- | --- |
| N=11 | c(11)=1 | c(9)=-0.893987576002104 | c(7)=0.500960268691744 | c(5)=-0.139839647261546 |
| N=21 | c(21)=1 | c(19)=-0.997652096371180 | c(17)=0.960784303783368 | c(15)=-0.801731880025198 |

1. Trivial coefficients (c\_k) are as follows:

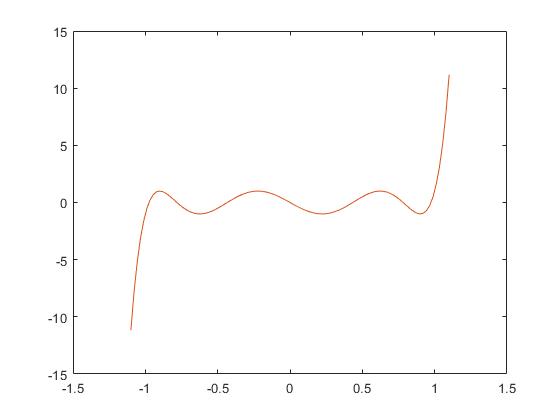
| N (number of data points) | c\_k |
| --- | --- |
| 5 | 5.153676567186254e-18 |
| 11 | 0 |
| 21 | 1.658970412809470e-14 |

The trivial coefficients are the c\_k’s where k is an odd integer and as we can see in the table the values are very close to, or even equal to zero corresponding to the terms with even powers. This means interpolating polynomial is quite similar to the taylor polynomial of runge function which is:

### Exercise 8

1. We have defined the “cheby\_trig” and “cheby\_recurs” functions such that one of them uses the direct definition of Chebyshev Polynomial with trigonometric functions and the other one uses recursive definition of Chebyshev Polynomial respectively. Defining “cheby\_trig” was quite easy as it just take one line code to finish it. On the other hand, defining “cheby\_trig” was not that easy. Since the recursive formula of the polynomial is designed for just real numbers, the main problem was that designing the function such that it should be able to take a row vector as an input of the recursive formula. By using for loop commands, we were able to solve that problem, and now our function is also takes a row matrix as an input, which will be so useful later on the exercise, while ploting some graphs.
2. After defining the functions, in order to check their consistency, we have defined a test row vector that is . After feeding the functions with and degree we have obtained following output,

Since the functions give same result and they are degree polynomials, we have that they are actually corresponding to the same polynomial. Thus, the defined functions are working well.

1. We have plotted the graph of both functions with degree on the interval . Both graphs lie on each other as we expected.

Exercise 9

1. We have defined our “cheby\_points.m” function, by using “for” loops. Although defining the function in such a techniqe is not an efficient way, it is really more clear to see the whole process with this definition.
2. In order to check that whether our function is working or not, we have tested it on the interval with degree . It was so nice to see that the output of the function is consistent with the result that we have found in the previous example (the roots of the polynomial that we have plotted is same with the output of the function).

1. We have looked at the result of the function on the interval with degree . The resulting points are not evenly spaced but symmetric about their center, which is (as we expected).