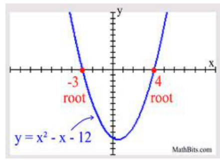
MA #4 Guide

Overall Goal: In this MA you will sharpen your understanding of data validation and while loops to complete a problem about polynomial functions. You will be determining at what x-value the polynomial crosses the x-axis, also known as the root of a function, using a calculation known as the Newton-Raphson method.



Third order polynomial example: $ax^3 + bx^2 + cx + d$ where a, b, c, d are numerical coefficients. In MATLAB, you will be referring to this polynomial as a vector of coefficients `[a b c d]`.

MA Requirements:

Algorithm: Submitted to Blackboard

MATLAB Code: Submitted to Zybooks (2 total submissions)

PROBLEM 1

Task 1: Main Script – Obtain and validate a user input for a coefficient vector

- Prompt the user to enter a vector that contains the coefficients for a polynomial.
- Perform data validation to check that the user-entered vector contains an even number of elements
  - ➢ HINT: Explore the `rem()` or `mod()` functions to help you. What is the remainder of an even number after dividing by 2? What is the remainder of an odd number after dividing by 2?
- Continue re-prompting the user to re-enter the vector ***until*** a valid entry is obtained. Limit the total number of tries to 5 attempts.
- If after the 5^{th} attempt, an invalid vector is still entered, display a warning, and remove the last entry so that you have an even number of elements in the vector:

  Example: `Invalid = [1 2 3 4 5]`   `Valid = [1 2 3 4]`
- Save the final vector as **MA4_Polynomial.csv**

Task 2: Main Script – Obtain and validate a user input for the root value initial guess and percent error

- Review the help documentation for the `polyval()` and `polyder()` functions. Practice in the command window to make sure you understand how each of these functions works and what the outputs mean! This is essential for the rest of the MA!
- Prompt the user to input an initial guess for V_o. V_o is the x-value at which the polynomial function they entered in Task 1 crosses the x-axis.
- Perform data validation to check that the derivative of the polynomial does not equal zero at the user-entered x-value. Continue prompting the user to re-enter an initial guess ***until*** a valid entry is obtained. Limit the total number of tries to 3 attempts.
  - ➢ HINT: You will need to use `polyval()` and `polyder()`
- After three failed attempts to enter an x-value where the derivative does not equal zero, issue an error and terminate the program.
- If a valid x-value is obtained, prompt the user to enter a percent error that will be used for the Newton-Raphson calculation. No additional validation is required.

Task 3: Function – Calculate the true x-value where the function crosses the x-axis using the Newton-Raphson method

- Create a new function named **FindZero.m**
  - Inputs: (3) Final vector of coefficients from Task 1, initial guess for V_o from Task 2, percent error from Task 2 as a whole number.
  - Outputs (2) Final value for V_o and number of iterations for the Newton-Raphson method
- Within your function, apply the Newton-Raphson method until the percent error between two subsequent iterations is less than the user-entered percent error.
- In the Newtonian-Raphson method you systematically guess different x-values for V_o until you fall below the user-specified error threshold.
  - For the first calculation, divide the value of the function for the given "initial guess" x-value by the value of the derivative when evaluated at the same "initial guess" x-value: $Output = \frac{f(guess)}{f'(guess)}$
    - Hint! `polyval()` and `polyder()` are very useful here!
  - For the next calculation, you use the $Output$ from the previous function as the "guess" and perform the same calculation.
- Continue this process until two subsequent outputs have a percent difference less than the user-specified threshold.
    - $Percent\ Diff = abs\left(\frac{Output - Previous\ Output}{Previous\ Output}\right) \cdot 100$
- If at any time the denominator in the calculation is equal to 0, issue an error and terminate the program.
- Add a counter to keep track of how many times you perform the Newton-Raphson method before converging on an x-value that falls within the error threshold.
  - Note: 1 iteration is comparing two subsequent calculations

Task 3: Main Script – Call your FindZero.m function

- Call your **FindZero.m** function to determine the final V_o value and number of Newton-Raphson iterations to reach that value.

Task 4: Main Script – Controls to repeat your program

- Use a menu to ask the user if they would like to repeat the program starting at Task 2.
  - If the user exits out of the menu without making selection, continue re-prompting them ***until*** a selection is made.
- If the program is repeated, each time the program is run, store the V_o value and number of Newton-Raphson iterations. You should have two separate vectors that you add onto each time the program is run:
  - Vector 1: Stores the V_o values from each run
  - Vector 2: Stores the number of Newton-Raphson iterations from each run
  - HINT: If you call your function 3 times, the vectors should each have 3 values
- Save the final vectors containing the V_o and iteration values as **MA4_Results.mat**.