Problem 1

The BaseConverter function takes three inputs from the user: n, num, and m.
n represents the current base of the input number (num). It should be an integer between 2 and 10.
num is a string representing the number that needs to be converted. It is entered as a string to handle cases where digits are greater than 9.
m represents the desired target/base for converting into. It should also be an integer between 2 and 10.
The function first checks if both inputs (n and m) are within valid range ([2,10]). If any of them is not within this range, it throws an error.

Next, it performs two conversion steps:
Conversion from base-n to decimal:
The function iterates through each digit in the input number (num) using a loop. For each digit at position i, it multiplies it by (n^(length(num)-i)), where (length(num)-i) represents its positional value based on its index in reverse order (rightmost digit has index length(num)). The results are accumulated in variable 'decNum'.

Conversion from decimal to base-m:
Using a while loop, starting with 'decNum', it repeatedly divides 'decNum' by 'm' using modulo operation ('mod') and floor division ('floor'). The remainder at each step represents a digit in target/base-m representation which is then appended at front of variable 'num_m'.

Finally, after completing all iterations of while loop until decNum becomes zero, the resulting converted number ('num_m') is displayed using fprintf().

Note that there are no direct outputs returned by this function; instead, the result is displayed on screen through fprintf().

Problem 2

The GaussElim function solves a system of linear equations using Gaussian elimination method.
It takes two inputs: A and b.
A: The coefficient matrix of the linear system. It must be a square matrix.
b: The right-hand side vector of the linear system. Its length must be equal to the number of rows in matrix A.
The function performs several checks to ensure that input conditions are met:
It checks whether A is a square matrix by comparing its number of rows with its number of columns using 'size' function. It verifies whether the dimensions between A and b are compatible by comparing their lengths/rows using 'size' function again. Finally, it checks whether A is nonsingular (i.e., its determinant is nonzero) by calculating its determinant using 'det' function. If any condition fails, an appropriate error message will be displayed.
Next, Gaussian elimination is performed on augmented matrix [A|b]. This process involves iterating over each row (i) except for the last row (size(Ab, 1)-1). For each iteration, another loop (j) iterates over subsequent rows (i+1) up to last row (size(Ab, 1)). In this loop, a factor value is calculated based on division between current row's element at column index 'i' and pivot element at (i,i) position. Then this factor multiplied with current row elements subtracted from corresponding elements in subsequent rows to eliminate variables below pivot position. After performing Gaussian elimination steps on augmented matrix [A|b], solution vector 'x' initialized as zeros array with same size as 'b'. Next, the last element ('x(end)') can directly computed as division between last entry in augmented column ('Ab(end,end)') and second-to-last entry ('Ab(end,end-1)').
Finally, backward substitution starts from second-to-last element down to first element ('for' loop). At each iteration step (index 'i'), variable 'sum_val' accumulates multiplication between elements at current row ('i') from column index (j>i+2) and corresponding entries from solution vector ('x'). Then subtracting this accumulated value from augmented column entry at current row and dividing by diagonal element at (i,i) gives updated value for variable 'x(i)'.

Problem 3

(a) The MontePi function uses a Monte Carlo simulation to estimate the value of pi.
The input, n, is used to determine the number of random points that will be generated within a unit square. Inside a loop, each point is generated with x and y coordinates as random numbers between 0 and 1. These points are then checked to see if they fall within an inscribed quarter circle by testing if x^2 + y^2 < 1. If this condition holds true, then the point lies inside the quarter circle, and is counted towards insideCircleCount. The coordinates are also stored in insideCirclePoints. If a point does not satisfy this condition, it falls outside or on the boundary of quarter circle. These points' coordinates are stored in outsideCirclePoints.
After all points have been processed, an approximation of pi (piApprox) is calculated using the ratio of points that fell inside the quarter circle to total number of points generated (n). This ratio is multiplied by 4 because we used only a quarter circle for our simulation. Next, absolute error (absError) and relative error (relError) are computed between estimated value (piApprox) and actual value of pi.
Finally, zero rows from both point matrices (those initially allocated but not filled with any point coordinates) are removed before returning these matrices along with estimated value of pi and errors.

(b) The MontePiPlots function is designed to visualize how performance (execution time) and accuracy (absolute error) vary with increasing number of random points (n) used in Monte Carlo simulation.
In this function, nValues is an array holding different values of n that you want to use for simulations. A loop runs over each value in nValues. For each iteration:
The current value (n) is passed to the previously defined MontePi function which returns estimated pi (piApprox), absolute error (absError) along with coordinates for points falling inside or outside quarter circle (insidePts, outsidePts). The execution time taken by this process is recorded using tic-toc functions. Execution times are stored in an array (executionTimes) corresponding to current index. Absolute errors are also stored similarly into an array (absErrors). After running simulations for all values in nValues, two separate plots are generated: one plotting execution times against respective 'n' values, another plotting absolute errors against respective 'n' values. Additionally, if we're at last iteration step i.e., we're processing last element in 'nValues', a scatter plot is created displaying randomly generated points within unit square; those falling inside quarter circle are shown with one color ('b') while those falling outside or on boundary with another color ('r').
Therefore, by running this script/function you get visual understanding about how increasing number of random points impacts computation time as well as accuracy in estimation process.