INTE 316 : NUMERICAL ANALYSIS AND PROGRAMMING

# Assignment 1:

## Chapter 1: Introduction to MATLAB

### Q1: For the following program: Compound interest:

```
balance = 1000;
rate = 0.09;
interest = rate * balance;
balance = balance + interest;
disp( 'New balance:' );
disp( balance );
```

**1.** Run the compound interest program as it stands.
**2.** Change the first statement in the program to read balance = 2000;
Make sure that you understand what happens when the program runs.
**3.** Leave out the line
balance = balance + interest;
and rerun. Can you explain what happens?
**4.** Rewrite the program so that the original value of balance is *not* lost.

**Q2:**
**2.1.** Give values to variables a and b on the command line—for example, a=3 and b=5. Write statements to find the sum, difference, product, and quotient of a and b.

**2.2.** Given a script for animating the Mexican hat problem.

```
[x y ] = meshgrid( -8 : 0.5 : 8);
r = sqrt(x.^2 + y.^2) + eps;
z = sin(r)./ r;
mesh(z);
```

Type this into the editor, save it, and execute it.

# INTE 316 : NUMERICAL ANALYSIS AND PROGRAMMING

**Q2:**

**2.1.** Evaluate the following expressions yourself (before you use MATLAB to check). The numerical answers are in parentheses.

**(a)** 2 / 2 * 3 (3)
**(b)** 2 / 3 ˆ 2 (2/9)
**(c)** (2 / 3) ˆ 2 (4/9)
**(d)** 2 + 3 * 4 – 4 (10)
**(e)** 2 ˆ 2 * 3 / 4 + 3 (6)
**(f)** 2 ˆ (2 * 3) / (4 + 3) (64/7)
**(g)** 2 * 3 + 4 (10)
**(h)** 2 ˆ 3 ˆ 2 (64)
**(i)** –4 ˆ 2 (–16; ˆ has higher precedence than –)

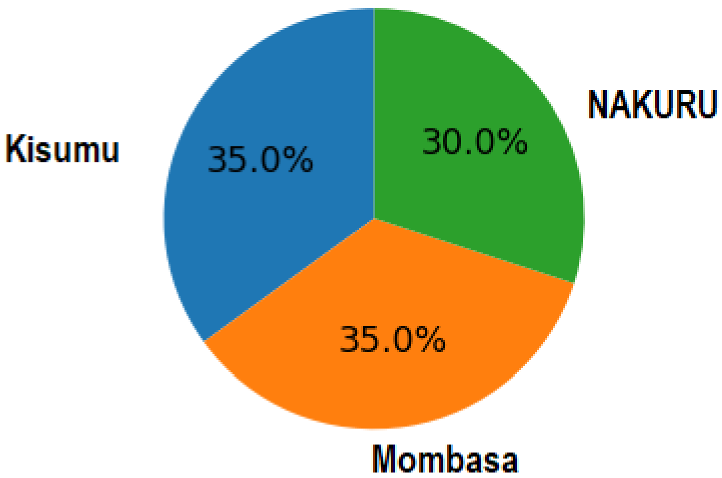
**2.2.** Use MATLAB to evaluate the following expressions. The answers are in parentheses.

**(a)** $\sqrt{2}$ (1.4142; use sqrt or ˆ0.5)
**(b)** $\frac{3+4}{5+6}$ (0.6364; use brackets)
**(c)** Find the sum of 5 and 3 divided by their product (0.5333)
**(d)** $2^{3^2}$ (512)
**(e)** Find the square of $2\pi$ (39.4784; use pi)
**(f)** $2\pi^2$ (19.7392)

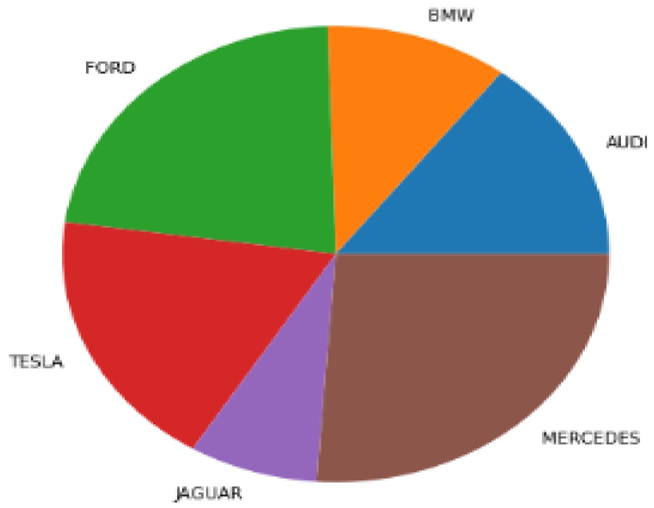


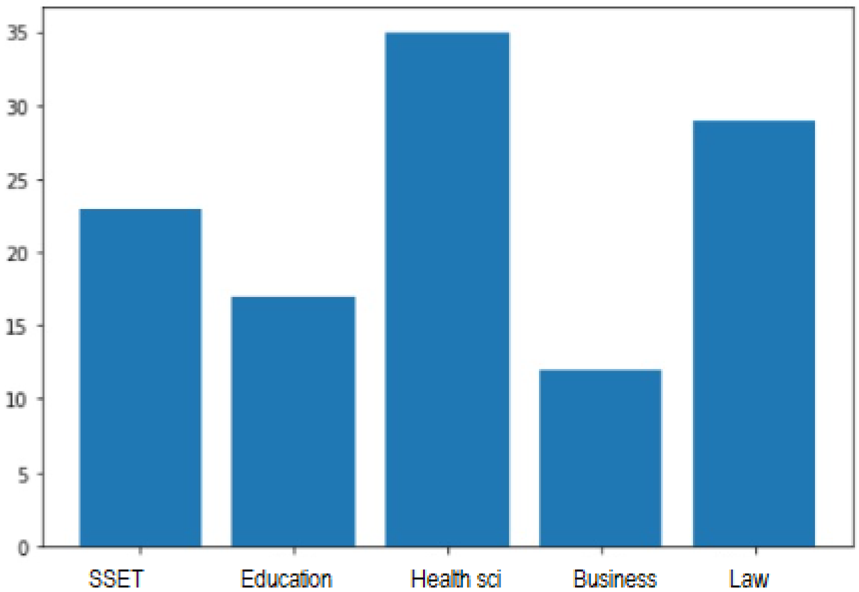
**PYTHON ASSIGNMENT**

a. Write a python program to out put the following shapes

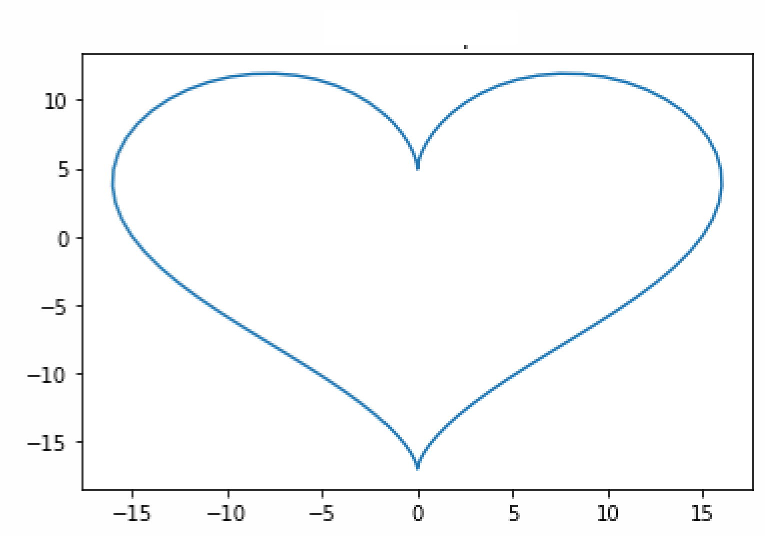


# INTE 316 : NUMERICAL ANALYSIS AND PROGRAMMING



# INTE 316 : NUMERICAL ANALYSIS AND PROGRAMMING




b. Write python programs to

- How to Remove rows in Numpy array that contains non-numeric values?
- Check whether a Numpy array contains a specified row
- Remove single-dimensional entries from the shape of an array
- Find the number of occurrences of a sequence in a NumPy array
- Find the most frequent value in a NumPy array
- Combining a one and a two-dimensional NumPy Array

c. Develop algorithms and Write a program using PYTHON to determine the roots of a quadratic equation using i)bisection method algorithm , ii) regular falsi iii)Newton Raphson method and iv) secant method. Determine the time complexity of your program

NB All data must be supplied by the user

- **Equation**
- **Lower/upper limit**
- **Error tolerance**
- **Iterations**

d)

One feature of NumPy that is powerful but tricky is the ability to perform broadcasting, which really just refers to repeatedly performing an operation over one or more dimensions. Start a new problem in your Colab notebook and when you're done with the entire assignment, follow the same procedure for appending a PDF and inserting the URL as you did in the previous assignment. The notebook URL is near the end of the assignment.

(A) The most basic kind of broadcast is with a scalar, in which you can perform a binary operation (e.g., add, multiply, ...) on an array and a scalar, the effect is to perform that operation with the scalar for every element of the array. To try this out, create a vector $1, 2, \ldots, 10$ by adding 1 to the result of the `arange` function.

(B) Now, create a $10 \times 10$ matrix $\boldsymbol{A}$ in which $A_{ij} = i + j$. You'll be able to do this using the vector you just created, and adding it to a reshaped version of itself.

(C) A very common use of broadcasting is to standardize data, i.e., to make it have zero mean and unit variance. First, create a fake "data set" with 50 examples, each with five dimensions.

```
import numpy.random as npr
data = np.exp(npr.randn(50,5))
```

You don't worry too much about what this code is doing at this stage of the course, but for completeness: it imports the NumPy random number generation library, then generates a $50 \times 5$ matrix of standard normal random variates and exponentiates them. The effect of this is to have a pretend data set of 50 independent and identically-distributed vectors from a log-normal distribution.

(D) Now, compute the mean and standard deviation of each column. This should result in two vectors of length 5. You'll need to think a little bit about how to use the `axis` argument to `mean` and `std`. Store these vectors into variables and print both of them.

(E) Now standardize the `data` matrix by 1) subtracting the mean off of each column, and 2) dividing each column by its standard deviation. Do this via broadcasting, and store the result in a matrix called `normalized`. To verify that you successfully did it, compute the mean and standard deviation of the columns of `normalized` and print them out.

e)

The core library for numerical computation in Python is called NumPy. NumPy provides useful abstractions to create and manipulate multidimensional arrays (e.g., vectors, matrices, tensors), and functions that are thin layers over high-performance C/C++/Fortran BLAS and LAPACK libraries. Conventionally you would start your numeric Python code with an `import numpy as np`. The most basic object is the NumPy array, which is a multi-dimensional array usually made up of type `double`, i.e., 64-bit floating point numbers. A vector is usually a one-dimensional array and a matrix is a two-dimensional array:

```
example_vector = np.array([50.0, 1.4, 0.4, 0.23, 1.04])
example_matrix = np.array([[ 3.4, 1.10,   0.8 ],
                           [ 4.2, 100.0, -1.4],
                           [ 1.1, 0.44, 9.4]])
```

Higher-dimensional arrays (tensors) are possible, but come up less often than vectors and matrices. The tuple made up of the size each dimension is called the *shape* and can be accessed as an attribute, so:

```
print(example_vector.shape, example_matrix.shape)
# Output: (5,) (3,3)
```

Create a Colab notebook following the same pattern as the previous assignment, and do the following:

(A) Use `arange` to create a variable named `foo` that stores an array of numbers from 0 to 29, inclusive. Print `foo` and its shape.

(B) Use the `reshape` function to change `foo` to a validly-shaped two-dimensional matrix and store it in a new variable called `bar`. Print `bar` and its shape.

(C) Create a third variable, `baz` that reshapes it into a valid three-dimensional shape. Print `baz` and its shape.

(D) There are several different ways to index into NumPy arrays. Use two-dimensional array indexing to set the first value in the second row of `bar` to -1. Now look at `foo` and `baz`. Did they change? Explain what's going on. (Hint: does `reshape` return a view or a copy?)

(E) Another thing that comes up a lot with array shapes is thinking about how to aggregate over specific dimensions. Figure out how the NumPy `sum` function works (and the `axis` argument in particular) and do the following:

   (i) Sum `baz` over its second dimension and print the result.

   (ii) Sum `baz` over its third dimension and print the result.

   (iii) Sum `baz` over **both** its first and third dimensions and print the result.

(F) Along with shaping and indexing, we also do a lot of slicing which is where you index with ranges to get subvectors and sometimes submatrices. Write code to do the following:

   (i) Slice out the second row of `bar` and print it.

   (ii) Slice out the last column of `bar` using the -1 notation and print it.

   (iii) Slice out the top right $2 \times 2$ submatrix of `bar` and print it.