Notes from February 2 – Tuesday

- The major difference between a calculator and computer was explained. The user of a calculator is required to make decisions to aid the calculation process whereas a computer program has decision making capabilities to allow the program to execute independently.

- A decision can be made based on a Boolean operation. The result is either true or false, to be represented by 1 or 0, respectively, in Matlab. For numerical values, there are the relational operators
  - == to test if the two operands are equal
  - ~= to test if the two operands are not equal
  - > to test if the left operand is greater than the right operand
  - < to test if the left operand is less than the right operand
  - >= to test if the left operand is greater than or equal to the right operand.
  - <= to test if the left operand is less than or equal to the right operand.

For Boolean variables, there are 3 basic logical operators
  - && the result is true if the left operand AND the right operand are both true
  - || the result is true if the left operand is true OR the right operand is true
  - ~ the NOT operator, change the Boolean variable to the other state

- An example was given using an interior number line bounded between x=0 and x=3. The value “x” is in the interval if “(x>=0)&&(x<=3)”

- Another example was given using an exterior number line to left of x=-3 and to the right of x=2 but not including 2. The value “x” is in the specified interval if “(x<=-3)||(x>2)”

- The concept of a function was introduced. A separate file must be created for each function. This is a necessity because Matlab is an interpreter. The first line of a function is in the form:
  
  ```matlab
  function answer=name(x,y,z)
  in which “answer” is to be returned to the calling program as the results desired. “name” is the name of the function and it should also be the name of the function file. “x,y,z” are the arguments provided for the calculation. A function can also return more than one answers, such as “function [a,b,c]=name(x,y,z)”```
A function etox was created to calculate e to the x using 10 term of the mathematical series:

```matlab
function answer=etox(x)
 term=1;
 sum=term;
 for i=1:10
   term=term*x/i;
   sum=sum+term;
 end
 answer=sum;
```

Note that before the end of the function the desired results should be placed in the return variable on the function line. The internal variables such as `term`, `sum` and `i` are different from those used in the calling program, they are known as local variables in most programming languages.

An improved function etox was created to calculate e to the x using to a preset accuracy:

```matlab
function answer=etox(x)
 term=1;
 sum=term;
 i=0;
 error=1;
 while error>1e-14
   i=i+1;
   term=term*x/i;
   sum=sum+term;
   error=abs(term/sum);
 end
 answer=sum;
```

Note that the while loop no longer provides a counter for `i` like the "for" loop, therefore, an increment statement must be provided within the loop to update the term counter "i". Also, care must be taken to be sure that the value of "error" be updated within the loop; otherwise, the loop would go on forever, forever, forever. The error limit 1e-14 was used because double precision calculation provides 15 digits of accuracy in Matlab.

To use the function “etox”, it is similar to the standard canned functions such as `exp`, `sin` and `cos`, i.e., “y=etox(1.45);”
The Newton-Raphson Method was described in class. For details, see Support Material for its derivation. It is an iterative method which requires an initial estimate, or guess, and then upon each iteration, the new value approaches the final solution, the root if a transcendental equation, \( f(x) = 0 \).

Let \( x_n \) be the initial estimate of the root for \( f(x) = 0 \) and the next value \( x_{n+1} \) could be obtained by \( x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} \). \( f' \) would be the derivative of \( f(x) \) with respect to \( x \). As an example, calculate the square root of 13. The root of \( x^2 - 13 = 0 \) would be \( \sqrt{13} \). The function in this case would be \( f(x) = x^2 - 13 \). Its derivative \( f'(x) = 2x \). In matlab, the iterative formula could be written as

\[
\begin{align*}
\text{xn} &= 13/2; \\
\text{xnp1} &= \text{xn} - (\text{xn}^2 - 13) / (2 * \text{xn});
\end{align*}
\]

The very rough estimate was used as \( 13/2 \) but this method converges after 4 or 5 iterations to a solution with more than 10 digits of accuracy for this particular simple function. The results of the iteration is \( x_n = 13/2 = 6.5 \) to 8 digits, using the formula \( x_{n+1} = 4.25 \) to 8 digits. Since the old \( x_n \) is no longer useful, erase it using \( x_n = x_{n+1} \), now \( x_n \) is 4.25. The next application of the formula will yield 3.65441176, then 3.60587791, then 3.60555129. Finally the last iteration yields the exact solution of 3.60555128.

There are situations in which the Newton-Raphson Method would not have a converging solution as \( f'(x) \) may be 0 for some values.

An introduction to the bisection method was also started near the end of the class. It converges by narrowing an interval and the convergence rate is quite a bit slower. But the method does not require the calculation of \( f'(x) \) and a real root can always be extracted.