**Report**

**Methods:**

def Bisection\_eval(exp, xl, xu, es, imax):  
 if exp.evalf(10,subs={x:xl})\*exp.evalf(10,subs={x:xu}) > 0:  
 print("no bisection")  
 return  
 for i in range(imax):  
 xr\_new = (xl + xu) / 2  
 if i != 0:  
 ea = abs((xr\_new-xr\_old/xr\_new))  
 xr\_old = xr\_new  
 if exp.evalf(10,subs={x:xl})\*exp.evalf(10,subs={x:xr\_new}) > 0:  
   
 xl = xr\_new

elif exp.evalf(10,subs={x:xl})\*exp.evalf(10,subs={x:xr\_new}) < 0:  
   
 xu = xr\_new  
  
 if ea is not None and ea > 1-es and ea < 1:  
   
 return result

def False\_Position(exp,xl,xu,es,imax):  
 if exp.evalf(10,subs={x:xl}) > 0 and exp.evalf(10,subs={x:xu}) < 0:  
 print("function has same sign at endpoints")  
 return  
 for i in range(imax):  
 y = xl \* exp.evalf(10,subs={x:xu}) - xu \* exp.evalf(10,subs={x:xl})  
 z = exp.evalf(10,subs={x:xu}) - exp.evalf(10,subs={x:xl})  
 xr\_new = y/z  
 if i != 0:  
 ea = abs((xr\_new-xr\_old/xr\_new))  
 xr\_old = xr\_new  
 if exp.evalf(10,subs={x:xr\_new}) > 0:  
 xu = xr\_new  
 elif exp.evalf(10,subs={x:xr\_new}) < 0:  
 xl = xr\_new  
 if ea is not None and ea >= 1-es and ea < 1:  
 return result

def Fixed\_Point(xr, expG, es, imax):  
 for i in range(imax):  
 xr\_old = xr  
 xr = expG.evalf(10, subs={x:xr})  
 if xr != 0:  
 ea = abs((xr - xr\_old) / xr)  
 if xr == 0 or ea >= 1-es and ea < 1:  
   
 return result

def Newton\_Raphson(expr, xr, es, imax):  
 diff\_expr = sym.diff(expr)  
 ea = None  
 for i in range(imax):  
 xr\_old = xr  
 xr = xr - (expr.evalf(10, subs={x:xr})) / (diff\_expr.evalf(10, subs={x:xr}))  
 ea = abs((xr\_old - xr)/xr)  
 if xr == 0 or ea < es:  
return result

def Secant\_Eval(expr, x0, x1, es, imax):  
 xr\_i1 = x1 # equivalent to x(i-1)  
 xr\_i0 = x0 # equivalent to x(i)  
 for i in range(imax):  
 xr = xr\_i0 - (expr.evalf(10, subs={x:xr\_i0}) \* (xr\_i1 - xr\_i0)) / (expr.evalf(10, subs={x:xr\_i1}) - expr.evalf(10, subs={x:xr\_i0}))  
 ea = abs((xr-xr\_i0)/xr)  
 xr\_i1 = xr\_i0 # make x(i-1) of next iteration = x(i) of current iteration  
 xr\_i0 = xr # make x(i) of next iteration = xr of current iteration  
 if xr == 0 or ea < es:  
  
 return result

I used Sympify’s built in data structure which was helpful in treating user input as mathemetical equation instead of standard string data type used in programming languages.