```
"""
Andy Lu
SMU Mathematics
MATH 5315 Fall 2017

Pseudocode for Homework 3
- This script is my implementation of a root finder using interpolating
  polynomials to do the work. A choice of using the Newton form of interpolating
  was made because it is easier to recompute the values of the coefficients
  rather than constantly rebuild polynomials of degree 0 up to n. Helper
  functions have been written to improve source code readability.
"""

quadratic_sol(INPUTS):

Usage: x, its = quadratic_sol( Inputs )
Inputs:
             Ffun      Nonlinear function name/handle
             x         Initial guess at solution
             maxit     max allowed number of iterations
             Srtol     relative solution tolerance
             Satol     absolute solution tolerance
             Rrtol     relative residual tolerance
             Ratol     absolute residual tolerance
             output    Boolean to output iteration history
Outputs:
             x         Approximate solution
             its       Number of iterations used

# Check EACH input arguments
  if bad input:
    set to a default value
    output warning to the screen

# Initialize variables
x0 = x

# Set up the two other initial guesses
if x0 equal 0:
  x1 =  x + 1e-2
  x2 = x - 1e-2
else:
  x1 = x(1 + 1e-2)
  x2 = x(1 - 1e-2)

LOOP: from 1 to maxit:
  # Call utility functions to build a quadratic interpolating function
  # clist will be [a, b, c] for ax^2 + bx + c
  clist = newtwoncoeff(Ffun, x0, x1, x2)

  # Using a,b,c plug those into the quadratic question and return a root, and if
  #  the root is imaginary
  root, imag = quad_equation(clist)

  # if root is imaginary, print message and quit
  if (imag):
    print error message and quit
```

```
  # Shift guesses
  x2 = root
  x1 = x2
  x0 = x1

  # Check for convergence
  if (exit_condition_true):
          break
END LOOP
END QUADRATIC_SOL


quad_equation()
Usage: root, imag = quad_equation(clist)
Inputs:
            clist         A list of coefficients in the order a,b,c for
                          ax^2 +bx + c
Outputs:
            root      Solution to the quadratic
            imag          Boolean describing if the root returned is imaginary
root = -(clist[1]) + sqrt(c[1]^2 - 4*c[0]*c[2]) / 2*c[0]
if c[1]^2 - 4*c[0]*c[2] < 0
     return root and true
else
     return root and false

newtoncoeff()
Usage: clist = newtoncoeff(Ffun, x0, x1, x2)
Inputs:
            Ffun          Function handle
            x0        A point to interpolate
            x1        A point to interpolate
            x2        A point to interpolate
Outputs:
            clist         A list of the coefficients that describe the
                          interpolating polynomial

# calculate divided differences
# store in an array
arr[i] = Ffun(xi)                                           # for i = 0, 1, ...
arr[i] = arr[i] - arr[i-1] / xi - xi-1                      # for i = 1, 2, ...
arr[i] = arr[i] - arr[i-1] / (xi - xi-1)(xi - xi-2)         #for i = 2, 3, ...
# No need to do anything else because we don't need to evaluate
#   our interpolant, just need to find one so we can find a root
# return the array
return arr
```

MATH 5315 Homework 3 Andy Lu

3.) Proof:

Suppose $f \in \mathbb{P}_K$. Pick $n \in \mathbb{N}$ s.t. $n > K$. Then by theorem 4 in section 6.2 of the book $f[x_0, x_1, \dots x_n] = 0$, since the nth derivative of f equals zero. $\blacksquare$

4.) Proof: Given $A_i(x) = [1 - 2(x - x_i)\ell_i'(x_i)]\ell_i^2(x)$ $(0 \le i \le n)$

$B_i(x) = (x - x_i)\ell_i^2(x)$ $(0 \le i \le n)$

where $\ell_i(x) = \prod_{\substack{j=0 \\ j \ne i}}^{n} \dfrac{x - x_j}{x_i - x_j}$ $(0 \le i \le n)$

Note: $\ell_i(x_i) = \prod_{\substack{j=0 \\ j \ne i}}^{n} \dfrac{(x_i - x_j)}{(x_i - x_j)} = \prod_{\substack{j=0 \\ j \ne i}}^{n} 1 = 1$ (Eq. 1)

$\ell_i(x_j) = \prod_{\substack{j=0 \\ j \ne i}}^{n} \dfrac{(x_j - x_j)}{(x_i - x_j)} = \prod_{\substack{j=0 \\ j \ne i}}^{n} \dfrac{(0)}{(x_i - x_j)} = 0$ (Eq. 2)

i. $A_i(x) = \delta_{ij}$

Case I: $x = x_i$

$A_i(x_i) = [1 - 2(x_i - x_i)\ell_i'(x_i)]\ell_i^2(x_i)$

$= 1$ since $(x_i - x_i) = 0$ and $\ell_i(x_i) = 1$ from (Eq. 1)

Case II: $x = x_j$

$A_i(x_j) = [1 - 2(x_j - x_i)\ell_i'(x_i)]\ell_i^2(x_j)$

$= 0$ since the square brackets multiply $\ell_i^2(x_j) = (0)^2 = 0$ by (Eq. 2).

Thus $A_i(x) = \delta_{ij}$.

ii. $A_i'(x_j) = 0$

$$A_i'(x_j) = \frac{d}{dx}\left(\left[1 - 2(x_j - x_i)\, l_i'(x_i)\right] l_i^2(x)\right)$$

$$= \frac{d}{dx}\left(l_i^2(x) + 2x_i\, l_i'(x_i)\, l_i^2(x) - 2 l_i'(x_i)\, l_i^2(x)\, x\right) \quad \left[\begin{array}{c}\text{Distribute} \\ \text{terms}\end{array}\right]$$

$$= \frac{d}{dx}\left(l_i^2(x)\left[1 + 2x_i\, l_i'(x_i)\right] - 2 l_i'(x_i)\, l_i^2(x)\, x\right)$$

The first term is a constant times $l_i^2(x)$, thus only a chain rule is needed. The second term is a constant times $l_i^2(x)$ times x; thus, a product rule is needed.

$$= 2(l_i(x))\, l_i'(x)\left[1 + 2x_i\, l_i'(x_i)\right] - 2 l_i'(x_i)\left[x\, 2 l_i(x) l_i'(x) + l_i^2(x)\right]$$

Evaluating at $x = x_j$ and using Eq. 2:

$$= 2(0)\, l_i'(x_j)\left[1 + 2x_i\, l_i'(x_i)\right] - 2 l_i'(x_i)\left[x_j\, 2 \cancel{l_i(x_j)}^{0}\, l_i'(x_j) + \cancel{l_i^2(x_j)}^{0}\right]$$

$$= 0 - 0 = 0$$

Thus $A_i'(x_j) = 0$

iii. $B_i(x_j) = 0$

Using Eq. 2, $B_i(x_j) = (x - x_i)\, l_i^2(x_j) = (x_j - x_i)(0)^2 = 0$

Thus $B_i(x_j) = 0$

iv. $B_i'(x_j) = \delta_{ij}$

$$B_i'(x_j) = x(2)\, l_i(x)\, l_i'(x) + l_i^2(x) - x_i(2)\, l_i(x)\, l_i'(x)$$

Case I: $x = x_i$

Evaluating $B_i'(x)$ at $x = x_i$ and using Eq. 1:

$$B_i'(x_i) = 2x_i\, l_i'(x_i) + 1 - 2x_i\, l_i'(x_i) = 1$$

Case II: $x = x_j$

Evaluating $B_i'(x)$ at $x = x_j$, since every term has a $l_i(x_j)$, then by Eq. 2, $B_i'(x_j) = 0$

Thus $B_i'(x_j) = \delta_{ij}$

$\blacksquare$