Erin Obermayer
Computational Mathematics HW 5

Problem 1:
(1a) It took 32 iterations to achieve 1e-8 accuracy.
(1b) It appears to converge to (x,y) = (1.570894960828351, -0.000022243833968). The true minima is at (x,y) = (2,0). It does not look like 1e-8 accuracy, but if you plug the solution into the problem, it is 1e-8 accuracy close to the true minimum value, z = -2.
(1c) It takes 178 iterations to achieve 1e-8 accuracy after changing the second initial point. It located the same minima at (x,y) = (1.570810651779174, -0.000037829081218). I think the step size was probably too big, and might have put the second iteration further away from the minima on the first try.
(1d) The algorithm finds the minima at (x,y) = (-4.715185546875040, 0.006766764322948).

Problem 2:
(2a) It takes 6 iterations to achieve 1e-8 accuracy.
(2b) It now takes 736 iterations, probably because the initial guess is far away from the minimum.

Problem 3:
(3a) Newton's Method and Successive Parabolic Interpolation appear to converge faster to less accurate tolerances.
(3b) When you increase the accuracy, Successive Parabolic Interpolation does not do well. For the most accurate tolerances, Newton's Method does the best.
(3c) I think Successive Parabolic Interpolation takes the longest because it inverses a matrix that gets more and more singular the more iterations, so the inverses get less accurate. Golden search and Newton's Method do not have these same issues, which is why they converge better. For Newton's Method, computing a derivative does not take long and should only make the next guess more accurate.

Problem 4:
(4a) It took 5 iterations to find the minima at (x,y) = (-4.7124, 0).
(4b) It took 4 iterations to find the minima at (x,y) = (-4.7124, 0).

PSEUDOCODE for Nelder Mead:

```
- Initialize:
    - three points to start with (x1,x2,x3)
    - Error and tolerance
- While loop (run until error > tolerance)
    - Update iteration number
```

```
        - Sort initial points in order from least to greatest by their function value. I will
          create a vector to hold the x and y values of each point, and their function values.
          Sort the function vector, and then reorder x1,x2,x3 based on the new indices. (x1
          should be the best guess)
        - Compute the midpoint of the two best points (x1 and x2)
        - Create a reach vector that will move x3
        - If this new vector is better than x3
            - Make the move
        - Else
            - Find a different point, xc
            - If make sure xc is better
                - Make the change
            - Else
                - Redefine x2 and x3 to make the triangle smaller
            - End
        - End
        - Redefine the new function value
        - Compute the error
    - End
    - Compute the approximate minimum
```

PSEUDOCODE for Newton's Method1D:

```
    - Input tolerance
    - initialize
        - number of iterations
        - Guess
        - error
    - While loop until error < tolerance
        - calculate root of derivative using newtons method: xn = xp - (f''(xp))/(f'(xp))
        - calculate new error
        - Update xp
        - Update n
    - End
```

PSEUDOCODE for Multivariable Newton's Method:

```
    - Function Newtons_2D_Opt
    - Initialize
        - Iteration number
        - Initial guess
        - Error
    - While loop until error > tolerance
        - Calculate root of gradient using Newton's method: xn = xp - invH(xp)*gradx(xp)
          (see other functions for invH and gradx)
        - Calculate new error (use L2)
```

    - Update xp
    - Update iteration number
- End

- **Function invH:** calculate inverse Hessian matrix for input xp
- Redefine xp vector into 2 x y components
- Calculate second partial derivatives
- Create Hessian Matrix: H = [fxx fxy; fxy fyy]
- Output inverse of H

- **Function gradx:** calculate gradient vector of input xp
- Redefine xp vector into x y components
- Calculate first partial derivatives
- Output gradient vector