SPS4050 Homework 5

Note: this homework set will be graded out of 24 points, with a maximum allowed score of 28 points. Remember to include your code alongside any output or by-hand derivations.

1. **(5 pts total)** For the fifth-degree polynomial below,

$$f(x) = x^5 - 0.832x^4 - 30.319x^3 + 23.954x^2 + 191.793x - 243.433$$

   (a) **(2 pts)** Write down the companion matrix for this polynomial

   (b) **(3 pts)** Use the QR method (either hand-coded or via `numpy`'s `linalg` package) to find the five real roots of the polynomial

2. **(4 pts)** Consider the function

$$f(x) = \frac{1}{1+x^2} - \frac{1}{2}$$

   Divide the interval $x \in [-3, 3]$ into 601 equally-spaced points (that is, steps of 0.01 between points). For each point, use Newton's method to find a root of $f(x)$. Plot $f(x)$ over the interval. Above the curve, mark whether each point converges to $x^* = +1$ or $x^* = -1$ by using plot markers (dots, crosses, etc.) of two different colors.

3. **(3 pts)** Given otherwise infinite precision, how many iterations of the bisection method are needed before a starter interval $[0, 1]$ bracketing a root has shrunk to at least 64-bit machine precision, $[x^*, x^* + 10^{-16}]$?

4. **(6 pts total)** Consider the function

$$f(x) = e^x - 2 - \frac{0.01}{x^2} + \frac{2 \times 10^{-6}}{x^3}$$

   and the intervals $[0.01, 3]$; $[0.01, 9]$; $[0.01, 27]$; and $[0.01, 81]$. For each interval, determine how many function evaluations it takes to find the root to an absolute accuracy of 1×10^{-6}, for (a) Bisection method, (b) Newton's method starting from the interval midpoint, and (c) Ridders' method. Code for Ridders' method is provided at the end of this document. Make a table showing your results so that the information is presented more compactly. (Scoring note: there are four intervals, and three methods per interval, worth 0.5 points each.)

5. **(8 pts total)** The power radiated per unit area, per unit wavelength, by a perfect blackbody follows the Planck radiation law

$$I(\lambda) = 2\pi hc^2 \frac{\lambda^{-5}}{e^{hc/\lambda k_B T} - 1}$$

where h, c, and k_B have their normal values, and the temperature T is measured in Kelvin.

   (a) **(3 pts)** For steps of 100 K between 2700K and 10000K, find the wavelength where the intensity $I(\lambda)$ is maximized. (Hint: plot $T = 2700$K and $T = 10000$K on the same plot to find bracketing values of λ covering the entire temperature range.) Plot wavelengths and temperatures together on a log-log plot to recover Wien's displacement law.

   (b) **(5 pts)** Some of the light emitted by the blackbody falls in the visible range, 390 nm $< \lambda <$ 750 nm, but not all of it. The efficiency of the blackbody—the fraction of the input energy that is radiated in visible light—can be calculated (after a bit of manipulation) to be

$$\eta = \frac{15}{\pi^4} \int_{hc/\lambda_2 k_B T}^{hc/\lambda_1 k_B T} \frac{x^3}{e^x - 1} dx$$

   where $\lambda_1 = 390$ nm and $\lambda_2 = 750$ nm are the limits of the visible spectrum. Determine the temperature where the Planck function achieves its maximum efficiency to the nearest Kelvin using the golden ratio method. (Hint: you'll probably have to do a numerical integration. Several, in fact.)

6. **(6 pts total)** In this question you will explore a 2-D function with multiple local minima. The function in question is

$$f(x_0, x_1) = x_0^2 - 2x_0 + x_1^4 - 2x_1^2 + x_1$$

   (a) **(2 pts)** Give the locations where the function's gradient is 0 (hint: there are three). You may use WolframAlpha or other math software as appropriate; don't stress about solving everything by hand.

   (b) **(1 pt)** Plot the 2-D function over the range $x_0 \in [0, 2]$, $x_1 \in [-1.5, 1.5]$.

   (c) **(3 pts)** Start at the locations $\mathbf{a} = (1.0, 0.26)$ and $\mathbf{b} = (1.0, 0.27)$ and apply gradient descent to find a minimum from each starter point. Ideally, plot the points encountered during the gradient descent on the same plot as part (b). If you choose not to do that, instead print a list of the points you encountered.

Code associated with problem 4

```
from math import exp, sqrt, sin, cos, log

""" Define the function that we'll find the roots of
"""
def f(x):
    #TODO: write the function


""" Implementation of Ridders' method.  The starting interval must
      bracket the root, but this is checked at the start of the
      function.
    Input arguments:
      (1) f: function that we want the root of
      (2) a: one end of bracketing interval
      (3) b: other end of bracketing interval
      (4) max_itrs (optional): maximum number of times to iterate before
           giving up
      (5) tol (optional): how far from the root are we willing to tolerate
    Output:
      (1) x_new: last guess of location of root
"""
def ridders(f,a,b,max_itrs=200,tol=1e-8):
    # Ensure that x0 < x1
    if a < b:
        x0 = a
        x1 = b
    else:
        x0 = b
        x1 = a
    # Compute the function at the two endpoints and make sure they bracket
    #   a root
    f0 = f(x0)
    f1 = f(x1)
    if f0 == 0.0:
        return x0
    elif f1 == 0.0:
        return x1
    if f0*f1 > 0.0:
        raise Exception("Starting points for Ridders' rule must bracket root")

    # Main Ridders' method loop
    for i in range(max_itrs):
        # Find the midpoint of the interval
```

```
        x2 = (x0+x1)/2.0
        f2 = f(x2)

        # Apply Ridders' formula to get x3
        x3 = x2 + (x1-x2)*(f2/f0)/sqrt((f2/f0)**2-(f1/f0))
        f3 = f(x3)
        print(i,x3)

        # Check to see if we can return: we took a very small step or
        #   our latest guess evaluates to 0
        if abs(x3-x2) < tol or f3 == 0.0:
            return x3

        # Bracket the root as tightly as possible
        if f3*f2 < 0.0:
            # The root lies between x2 and x3, so set that as the interval for
            #   the next iteration
            x0 = x2; f0 = f2
            x1 = x3; f1 = f3
        else:
            # f3 and f2 have same sign, so x3 and x2 don't bracket the root.
            #   Check whether f0 or f1 is the other endpoint
            if f3*f1 < 0.0:
                # x3 and x1 bracket the root, so replace x0 with x3
                x0 = x3; f0 = f3
            else:
                # x3 and x0 bracket the root, so replace x1 with x3
                x1 = x3; f1 = f3

    print("Exceeded iteration limit without solution")
    return None


""" By putting our main function inside this if statement, we can safely
        import the module from other scripts without having this code execute
        every time
"""
if __name__ == '__main__':
    #TODO: call Ridders' method appropriately for a specified interval.
```