SPS4050 Homework 6

Note: this homework set will be graded out of 25 points, with a maximum allowed score of 30 points. Remember to include your code alongside any output or by-hand derivations.

1. **(4 pts)** Consider the linear, inhomogeneous, 1st-order ODE below (some of you may recognize this as the equation for radiative transfer, where the spectral intensity I_ν is modified by both photon production j_ν and photon absorption α_ν):

$$\frac{dI_\nu}{ds} = j_\nu - \alpha_\nu I_\nu.$$

   Insert the above ODE into the backwards Euler method, and demonstrate that you can still compute the next step explicitly; in other words, show that it is still possible to isolate $I_{\nu,\mathrm{n+1}}$ on one side of the equation in terms of, e.g. $I_{\nu,n}$ and other quantities.

2. **(5 pts total)** The Lotka-Volterra model is a basic way to describe the population dynamics of a predator-prey system (and was used to explain anomalous fishery results after World War I disrupted large-scale fishing operations). It involves two time-dependent populations, the predator $f(t)$ and the prey $h(t)$. The equations,

$$\frac{dh}{dt} = \alpha h - \beta f h$$
$$\frac{df}{dt} = -\gamma f + \delta f h$$

   involve four parameters: α is the per-capita growth rate of the prey; β is the rate at which predator-prey interactions reduce the prey population; γ is the per-capita death rate of the predator; and δ is the rate at which predator-prey interactions allow the predator population to increase.

   (a) **(3 pts)** Consider an initial population in an area of 4,000 foxes and 35,000 hares. The dynamical parameters for this particular species are $\alpha = 0.2$ yr^{-1}, $\beta = 2\times10^{-5}$ yr^{-1} fox^{-1}, $\gamma = 0.15$ yr^{-1}, and $\delta = 2.5\times10^{-5}$ yr^{-1} hare^{-1}. Use an ODE solver of your choice to propagate the population forward in time. Plot the populations over at least two full cycles.

   (b) **(1 pt)** What is the cycle length; that is, the time between successive peaks (or troughs) in a given species' population?

   (c) **(1 pt)** What is the maximum hare population at any point in the cycle? Give your answer to at least three significant figures; this may mean re-simulating a smaller interval with finer time steps to ensure you capture the peak accurately.

3. **(8 pts total)** In this problem you will explore a variant of the SEIR model for epidemiology. The four equations of the model are

$$\begin{aligned}\frac{dS}{dt} &= -\frac{\beta I}{N}S \\ \frac{dE}{dt} &= \frac{\beta I}{N}S - \sigma E \\ \frac{dI}{dt} &= \sigma E - \gamma I \\ \frac{dR}{dt} &= \gamma I\end{aligned}$$

and the parameters are as follows: $\beta = 0.514$ day^{-1} is the transmission efficiency between infectious and susceptible individuals; $\sigma = 0.50$ day^{-1} is the rate at which exposed individuals become infectious; $\gamma = 0.20$ day^{-1} is the clearance rate at which infectious individuals recover and gain immunity; and $N = S + E + I + R$ is the total population.[1] (Note that immunity is lifelong, and there is no birth or death in the population. This is obviously a simplified model.)

(a) **(3 pts)** Consider an initial population with $S_0 = 6 \times 10^6 - 2 = 5999998$ susceptible individuals, and $I_0 = 2$ infectious individuals. Further, assume the "mean field" limit, where $S/E/I/R$ are allowed to take decimal values (corresponding to non-integer numbers of people—again, this is a simplified model). Using an ODE solver of your choice, plot the four population curves from $t = 0$ days to $t = 300$ days with a Δt of 1 day.

(b) **(2 pts)** At what day number (whole numbers are fine here) does the infectious population peak? What is the value of that peak?

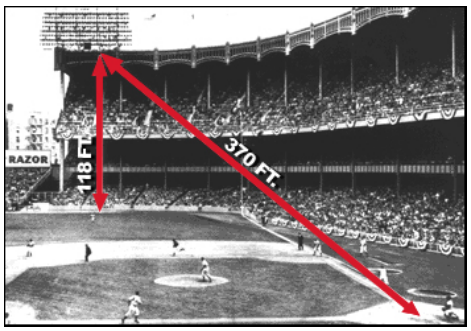
(c) **(2 pts)** In the early days of an epidemic, the number of infectious people can be approximated by an exponential function, $I(t) \approx I_0 e^{\kappa t}$ for some growth rate κ. Show that, if $I_1 = I(t_1)$ and $I_2 = I(t_2)$ at two consecutive days, the value of κ is given by

$$\kappa = \ln\left(\frac{I_2}{I_1}\right)(t_2 - t_1)^{-1}$$

(d) **(1 pt)** Plot κ as a function of day number for each day between 1 and 300, inclusive.

[1] These parameter values were taken from Asif et al. (2020), https://doi.org/10.1016/j.chaos.2020.110340

4. **(5 pts total)** (You were warned this was coming.) The photo at right shows what baseball player Mickey Mantle called "the hardest home run [he] ever hit". Starting from roughly 3 feet above home plate, the ball very nearly left Yankee Stadium over the stands in right field. The final position of the ball's trajectory is reliably known, making this a boundary value problem:



$$x(0) = 0 \qquad x(T) = 106.9 \text{ m}$$
$$y(0) = 0.9 \text{ m} \qquad y(T) = 36.0 \text{ m}$$

During its flight, the ball experienced a variety of forces: gravity, wind resistance, and the Magnus effect. These may be summarized in the following pair of second-order ODEs:[2]

$$\ddot{x} = -0.00258 \cdot C_d V \cdot \dot{x}$$
$$\ddot{y} = -0.00258 \cdot C_d V \cdot \dot{y} - g + 4.25 \times 10^{-4} V^2$$

Note that the $\ddot{x}$ equation contains a single negative term (for wind resistance), while the $\ddot{y}$ equation contains two negative terms (wind resistance and gravity) and one *positive* term (lift due to the Magnus effect). The quantity $V = \sqrt{\dot{x}^2 + \dot{y}^2}$ is the baseball's instantaneous speed, and the drag coefficient is $C_d = 0.29 + 0.22/[1 + \exp(V/5.2 - 6.23)]$. The acceleration due to gravity is $g = 9.81$ m/s^2, of course.

(a) **(3 pts)** Assume that the baseball had an initial launch angle of 30°, i.e. $\dot{x} = V \cos 30°$ and $\dot{y} = V \sin 30°$, and use the shooting method to find the minimum initial speed the baseball could have had to satisfy the boundary condition shown in the picture: **when** $x(T) = 106.9$**,** $y(T)$ **must be no less than** 36.0. Round your answer to the nearest 0.1 m/s. You do not need to implement the automatic bisection approach for credit (but you can if you wish); guess-and-check is acceptable as long as you document your guesses that get you to the right answer.

(b) **(2 pts)** For the same initial speed you found in part (a), extend the baseball's trajectory to find out how far from home plate it would have landed if it hadn't bonked off the structure in Yankee Stadium. Round your answer to the nearest 0.1 m.

5. **(5 pts total)** Consider the 1$^{\text{st}}$-order ODE with an initial condition $y(0) = 1$:

$$\frac{dy}{dx} = \frac{1}{2}(x - 1)(2 - y)$$

(a) **(2 pts)** Show that the function

$$y(x) = 2 - e^{(x/2) - (x/2)^2}$$

is the solution to the ODE above with the given initial condition.

(b) **(3 pts)** Evaluate the ODE from $x = 0$ to $x = 5$ using the RK4 and RKF4(5) methods. (Code for the latter method is provided at the end of this document.) For each method, determine the minimum number of function calls needed to reach a relative error of 10^{-6}. Give your two answers (one for each method) to just two significant figures.

[2] Formulae adapted from Escalera Santos et al. (2019), https://doi.org/10.3389/fams.2018.00066

6. **(8 pts total)** For a non-rotating star in hydrostatic equilibrium (i.e. no time dependent behavior), the four equations of stellar structure are as follows:

$$\frac{dr}{dM_r} = \frac{1}{4\pi r^2 \rho} \qquad \frac{dT}{dM_r} = -\frac{3\kappa L_r}{64\pi^2 acT^3 r^4}$$
$$\frac{dP}{dM_r} = -\frac{GM_r}{4\pi r^4} \qquad \frac{dL_r}{dM_r} = \epsilon$$

The key variables are the radial coordinate r; M_r, the mass interior to r; P, the local pressure; T, the local temperature; and L_r, the total energy production interior to r. Note that P and T are local, while M_r and L_r are cumulative. In addition to the key variables just listed, there are numerous auxiliary variables and coefficients used in the equations:

$$\rho = 9.91 \times 10^{-28} \frac{P - aT^4/3}{k_B T} \,[\mathrm{kg\,m^{-3}}]$$
$$\kappa = 0.035 + 6.44 \times 10^{18} \frac{\rho}{T^{3.5}} \,[\mathrm{m^2\,kg^{-1}}]$$
$$a = 7.565 \times 10^{-16} \,[\mathrm{J\,m^{-3}\,K^{-4}}]$$
$$\epsilon = 0.136\rho T_6^{-2/3} e^{-33.80\,T_6^{-1/3}} + 4.69 \times 10^{11} \rho^2 T_6^{-3} e^{-4403\,T_6^{-1}} \,[\mathrm{W\,kg^{-1}}]$$

The above formulae all use SI units, $T_6 \equiv T/(10^6\ \mathrm{K})$, and the constants c and k_B have their usual values. These have been calculated assuming a primordial star that is 75% hydrogen and 25% helium by mass, and which is fully ionized throughout.

This problem is another boundary value problem. Take a primordial Population-III (i.e. pure H/He) star whose mass is $100M_\odot$. At its surface you have $M_r = M_* = 100M_\odot$, and $P = 0$. At its core you have $L_r = r = 0$.

(a) **(3 pts)** Use the shooting method to estimate the core temperature and pressure of this star, to two significant figures each. I suggest you use RK4 as your ODE solver, with 100 steps between $M_r = 0$ and $M_r = M_*$. Your goal is to get P close to 0 at the surface without going negative (since pressure is a positive quantity). To guide your initial guess, the Sun's core temperature is 16×10^6 K, and its core pressure is 1.2×10^{16} Pa. Since this is a 2-D space, it is even trickier to automate than the baseball problem above (it's possible, just tricky).

(b) **(2 pts)** What are the radius and luminosity of the star, in units of solar radius ($R_\odot = 6.96 \times 10^8$ m) and solar luminosity ($L_\odot = 3.83 \times 10^{26}$ W)?

(c) **(3 pts)** Plot $r(M)$, $T(M)$, and $L(M)$ from 0 to M_*. (It will probably be easier to read if you use three different plots rather than one panel for all three curves.) Use logarithmic y axes and a linear x axis.

Code associated with problem 5

```
from math import sin, cos, log, exp, sqrt, pi
import numpy as np
import matplotlib.pyplot as plt

funcevals = 0

""" Define the derivative part of the ODE.  Since this is an ODE,
      we might need both x and y -- so pass them both regardless
      of whether or not they're used in the computation.
"""
def f(x,y):
    global funcevals
    funcevals += 1
    fy = 0.5*sin(x)-cos(pi*x)
    return fy


""" Define the derivative part of the ODE.  Since this is an ODE,
      we might need both t and S -- so pass them both regardless
      of whether or not they're used in the computation.
    In this function S is a state vector, so we need to split it
      into components at first and also return a vector of
      derivatives
"""
def fvec(t,S):
    global funcevals
    funcevals += 1

    # Unpack the state vector
    x = S[0]
    y = S[1]
    vx = S[2]
    vy = S[3]

    # Compute any additional quantities
    V = sqrt(vx**2 + vy**2)

    # Compute the derivatives
    fx = vx
    fy = vy
    fvx = -2*vx*V
    fvy = -2*vy*V
    return np.array([fx,fy,fvx,fvy],float)
```

```
""" Implementation of Runge-Kutta-Fehlberg method for solving 1D 1st-
      order ODEs.
    Input arguments:
      (1) f: function to compute derivative at given x, y
      (2) a: starting point of interval over which we want to solve
               the ODE
      (3) b: ending point of the interval
      (4) y0: intial value of y, i.e. y(a)
      (5) tol (optional): desired accuracy at each step
    Returns:
      (1) y: value of y after RKF45 method, i.e. y(b)
"""
def rkf45(f,a,b,y0,tol=1e-8):
    # Very crude initial guess for h
    h = (b-a)/float(100)
    x = a
    y = y0

    while True:
        # Assume we don't exit this step; change if needed
        break_now = False

        # Check whether current step size would carry past b, and
        #   adjust h if so
        if x+h > b:
            h = b-x
            break_now = True

        # Perform the RK step
        k0 = h*f(x, y)
        k1 = h*f(x+h/4, y+k0/4)
        k2 = h*f(x + 3*h/8, y + 3*k0/32 + 9*k1/32)
        k3 = h*f(x + 12*h/13, y + 1932*k0/2197 - 7200*k1/2197 + 7296*k2/2197)
        k4 = h*f(x+h, y + 439*k0/216 - 8*k1 + 3680*k2/513 - 845/4104*k3)
        k5 = h*f(x+h/2, y - 8*k0/27 + 2*k1 - 3544*k2/2565 \
                        + 1859*k3/4104 - 11*k4/40)

        ystar = y + 25*k0/216 + 1408*k2/2565 + 2197*k3/4104 - k4/5
        ynew  = y + 16*k0/135 + 6656*k2/12825 \
                  + 28561*k3/56430 - 9*k4/50 + 2*k5/55

        # Check the error; update x, y if allowed
        err = abs(ystar - ynew)
        if err < tol:
            x += h
```

```
            y = ystar

        # Exit if instructed to
        if break_now:
            break

        # Adjust step size
        if err != 0.0:
            h *= 0.8*(tol/err)**0.2
        else:
            h *= 5.0

    return y


""" Implementation of 4th-order Runge-Kutta method for solving
      simultaneous 1st-order ODEs.
    Input arguments:
      (1) f: function to compute derivative
      (2) a: starting point of interval over which we'll solve ODE
      (3) b: ending point of the interval
      (4) N: number of steps to take -- number of total points will be
               one higher
      (5) S0: intial value of state vector S, i.e. S(t=0)
    Returns:
      (1) ts: values of t at the N steps from a to b (including a and b)
      (2) Ss: values of S at the N steps from a to b (including a and b)
                Beware that Ss is a list of vectors, so make sure your
                call to rk4_vec is written to accept the right kind of
                output
"""
def rk4_vec(f,a,b,N,S0):
    h = (b-a)/float(N)
    ts = a + np.arange(N+1)*h
    # Make sure this has the right shape to hold the state vectors
    Ss = np.zeros((N+1,4))
    S = S0

    for i,t in enumerate(ts):
        Ss[i] = S
        k0 = h*f(t,S)
        k1 = h*f(t+0.5*h, S+0.5*k0)
        k2 = h*f(t+0.5*h, S+0.5*k1)
        k3 = h*f(t+h, S+k2)
        S += (k0 + 2*k1 + 2*k2 + k3)/6.0

    return ts, Ss
```

```
""" 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__':

    # Set initial quantities
    #TODO: set a, b, and y0 here

    # Compute the analytical solution
    ans = #TODO: if you know the answer, compute it here

    # Create the arrays to hold y values

    ys_rk45 = []
    fs_rk45 = []
    rk45_errs = []

    # Fill out the array of tolerances to use for RKF45
    #TODO: make an array of tolerances to check for accuracy

    for tol in tols:
        funcevals = 0
        ys_rk45.append( rkf45(f,a,b,y0,tol) )
        fs_rk45.append(funcevals)
        rk45_errs.append( abs(ys_rk45[-1]-ans)/abs(ys_rk45[-1]) )

    fig,ax = plt.subplots()
    plt.yscale('log')
    plt.xscale('log')
    plt.scatter(fs_rk45,rk45_errs)
    plt.scatter(fs_rk45,tols)
    plt.show()
```