Name and student number: _____

PSCB57. FINAL EXAM 6 DEC 2019 - PROBLEMS WITH **SOLUTIONS**

Points in the square brackets give the idea of relative weight of problems. This part is worth **19% of the total course score**. Please use left (blank) pages of the booklet as scratchpad and mark it as such. Write legibly a well commented Python program on the right, lined, pages. If the assignment asks you to correct a code, you can skip the comments in the corrected code. Otherwise, in program header please explain the general outline of your code, i.e., what your algorithm is doing step by step. You can describe the algorithm separately from the code if you wish, but the lines of code still must have brief comments on what they are supposed to do. Even if you get stuck somewhere, you may get a partial credit. We value more a clear and correct algorithm than correct Python syntax or style.

1 [4 pts.] Find 6 coding errors

They could be conceptual or syntax mistakes. Circle and number them, In booklet write the correct code doing what it claims to do, skipping comments. In case you need to adjust indentation of a block of k lines by a constant amount, that counts as one error, not k errors. The lack of calling program and 'import numpy as np' statement is not an error.

```
'''
    Find a root of equation x^4 - x - sqrt(x) == 0
        in interval [a,b] by the secant method
'''
def f(z):
                sq_root = np.sqrt(z)  #[7th error, unintended indent problem in edit
    f(z) =  z^4 - z - sq_root
    return()

def secant(f,a,b):
    xp = a      # previous x point
    yp = f(xp)
    x  = b      # current x point
    y  = f(x)
    count = 0
    while( b-a > 1e-15 and y /= yp and count > 100):
        inv_slope = (x-xp)/(y-yp)
        xp, yp = x, y              # store as previous values
        x  = x - y*inv_slope   # like Newton but with slope /= f'(x)
        y  = f(x)
        count += 1
        if (abs(y) == 0.):
        return((x,-count))     # negative count marks y==0  result
        break
    return(x,count)
```

1.1 Solution

There were 9 errors, 3 are forgiven if not marked, missing any of the other 6 errors or marking a correct statement as error normally results in -0.6 pt out of 4 pt.

```
def f(z):
    sq_root = np.sqrt(z)
    f =  z**4 - z - sq_root
    return(f)

def secant(f,a,b):
    xp = a
    yp = f(xp)
    x  = b
    y  = f(x)
    count = 0
    while( x-xp > 1e-15 and y != yp and count < 100):
        inv_slope = (x-xp)/(y-yp)
        xp, yp = x, y
        x  = x - y*inv_slope
        y  = f(x)
        count += 1
        if (abs(y) == 0.):
            return(x,-count)
    return(x,count)
```

2 [5 pts.] (Unidentified) Flying Object

An object with cross-sectional area A = 15 m^2 and mass $M = 400$ kg flies tumbling through the air with an average drag coefficient $Cd = 0.55$ Initial horizontal position is $x = 0$ and the height is $z = 6$ m. Initial speed components are $v_x = 70$ m/s and $v_z = 20$ m/s. Gravitational acceleration is a constant = g = -9.81 m/s^2, along z axis. Write a program that simulates the motion of the object with time step dt=0.001 s, taking air drag into account. Use simple Euler scheme or leapfrog integrator (it's up to you). Call the 4 integrated variables x,z,vx,vz in the program, and write down the first-order equations for them before you start coding, for instance $d(vx)/dt = ...$, $dx/dt = vx$, etc.

Air density is ρ=1.25 kg m^{-3}, and the acceleration components including air drag contributions are:

$$a_x = -v_x \, v \, D$$

$$a_z = -g - v_z \, v \, D$$

where $D = (Cd/2)\ (A/M)\rho$, and $v = \sqrt{v_x^2 + v_z^2}$.

The program should perfom each time step by calling a function step(t,vx,vz,x,z) which updates and returns all 5 variables in this way:

t,vx,vz,x,z = step(t,vx,vz,x,z).

The program should silently integrate the trajectory (x(t),z(t)) until the object collides with a gently rising terrain, which has altitude $H(x) = 0.02x$. It should print the final position and velocity components of the object, as well as the total flight time (preceding each value by the name of variable being printed).

2.1 Solution

```
'''
    one time step of simulation; 2nd order leapfrog scheme
'''
def step(t,vx,vz,x,z):
    drag = D*(vx*vx+vz*vz)**0.5
    ax =       -vx*drag
    az = -9.81 -vz*drag
    vx = vx + ax*dt*q # in 1st step vx becomes midpoint vx due to q=0.5
    vz = vz + az*dt*q
    x = x + vx*dt     #  new position x computed using midpoint velocity vx
    z = z + vz*dt
    q = 1
    t += dt
    return(t,vx,vz,x,z)

'''
        Main program ufo.py
    Simulation of flight of a tumbling object in the air
    drag acceleration vector = -Cd/2 v|v| rho A/M
    Init. conditions: (x0,z0) = (0,6) m, (vx0,vz0) = (70,20) m/s
    Method: leapfrog
'''
Cd = 0.55          # drag coefficient
A, M = 15., 400    # m^2, kg
rho = 1.25         # kg/m^3  density of air
D = Cd/2*A/M *rho # drag constant, for efficiency evaluated only once here
t, dt = 0, 0.001   # s
vx,vz = 70, 20     # m/s
x,z = 0, 6         # m
q = 0.5            # will be reset to q=1 after the first step
while(z > 0.02*x):
    t,vx,vz,x,z = step(t,vx,vz,x,z)
print('Total time of flight ',t,'  position x,z ',x,z,'  vx,vz ',vx,vz)
```

3 [6 pts.] Prove order

Demonstrate that the popular Leapfrog Method for integrating an ordinary differential equation often found in dynamics

$$\frac{d^2x}{dt^2} = f(x)$$

as an initial value problem on interval $0 < t < 1$ divided into subintervals of width $h = const.$, is a second order method. In other word, the difference between the numerical solution and exact solution $x(t)$, at the end of the global interval [0,1], is $O(h^2)$.

One step of leapfrog can be written as:

$$v_{1/2} = v_{-1/2} + hf_0$$

$$x_1 = x_0 + hv_{1/2}$$

where subscript n refers to time $t = t_0 + nh$. E.g., $n = 0$ stands for the beginning of the first time step.

Hints: Single time step of length h adds a truncation error in x equal $O(h^p)$. Derive p, then consider composite scheme on the whole time interval, in order to show that it has error $O(h^2)$.

Use Taylor expansions, and be clear in your designations: quantities describing exact solution should have explicit dependence on time shown, for instance $x(t_0 + h)$. Numerically computed values should have subscripts, as shown above. At the start of integration, $v_{-1/2}$ and x_0 are assumed to be exact.

3.1 Solution

We will use subscripts for values in the numerical algorithm and explicit argument notation $x(t)$ for exact solution.

Consider the first time step from t_0 to $t_1 = t_0 + h$. Values x_0 and $v_{-1/2}$ are exact: $x_0 = x(t_0)$ and $v_{-1/2} = v(t - h/2)$.

Taylor expansion of the **exact** solution $x(t_1) = x(t_0 + h)$ at the end of the subinterval reads:

$$x(t_0 + h) = x(t_0) + hv(t_0) + (1/2!)h^2 f(t_0) + O(h^3)$$

where $v = dx/dt$, and $f = dv/dt$. We need the $v(t_0)$ which is not the same as time-lagged quantity $v_{-1/2}$ we know in the leapfrog scheme. So we use Taylor expansion around t_0 again:

$$v(t_0 - h/2) = v(t_0) - (h/2)f(t_0) + O(h^2)$$

and substitute $v(t_0)$ from this equation into the equation for $x(t_1)$ above. After simplification,

$$x(t_0 + h) = x(t_0) + hv(t_0 - h/2) + h^2 f(t_0) + O(h^3).$$

Let's now use $x_0 = x(t_0)$, $v_{-1/2} = v(t - h/2)$, $f_0 = f(t_0)$, and the equations specifying one leapfrog step from the formulation of the problem:

$$x(t_0 + h) = x_0 + hv_{-1/2} + h^2 f_0 + O(h^3) = x_0 + h[v_{-1/2} + hf_0] + O(h^3)$$

or

$$x(t_1) = x_0 + hv_{1/2} + O(h^3) = x_1 + O(h^3).$$

This proves that one step of leapfrog adds truncation error $|x_1 - x(t_1)| = O(h^3)$, and a composite scheme made of $N = 1/h$ such steps adds of order $O(Nh^3) = O(h^2)$ error. Thus the leapfrog scheme is what we call a second order method.

4 [4 pts.] Long and short of it

Write a code that simulates one million sticks of length = 1.8, broken in 2 (uniformly) random points x1 and x2. Compute and print the mean length of the longest and the shortest piece, together with their standard errors (standard deviations of the mean, not of values around the average).

Hint: Use x = np.random.random() to generate a uniformly distributed number x. Alternatively, generate all N random points x using x = np.random.random(N). In the second approach, you can do without the Python loops. To produce minimum of two numbers OR two arrays, use: x_min = np.minimum(x1,x2); similar for maximum.

4.1 Solution

Two solutions are given below. One generates two breakpoints in each passage of a python loop, and computes std error from the accumulated (mean of squared values)-(square of mean values). The second program generates 2N breakpoints and stores them in 2 numpy arrays, and has no python loops. It sorts the points using: array_of_min_values = np.minimum(array1,array2) construct. It computes the std error by using mean_value = array.mean() method for finding the average, and error_of_the_mean = array.std()/N**0.5 .

I have added the computation of statistics of middle piece as well, it was not required.

Although they work at understandably different speed (33x different) and different RAM usage, they both can run with N = 10M. This allows a rather precise characterization of the mean length (subject to slight stochastic variation, always within the error estimate):

longest piece = 1.099881 +- 8.06e-5

intermed. piece = 0.500023 +- 5.92e-5

shortest piece = 0.200096 +- 4.47e-5

In other words, the pieces are surely in the integer ratio 2:5:11, though we haven't shown it analytically. The two shorter pieces together are shorter than the longest piece, on average!

```
'''
    Long and short of it.          long-n-short-p.py
    Generate N sticks of length 1.8m, and break each at 2
    uniformly random points. Print the mean and std of
    the length of the longest and the shortest piece.
'''
import numpy as np
N = 1000000
s,s2 = 0.,0.  # shortest piece, average and average of squares
l,l2 = 0.,0.  # longest  piece, average and average of squares
m,m2 = 0.,0.
for i in range(N):          # loop over all sticks
    x1,x2 = np.random.random(),np.random.random() # two breakpoints
    L1 = min(x1,x2)         # leftmost piece
    L2 = abs(x1-x2)                 # middle piece
    L3 = 1. - max(x1,x2)    # |--L1--|-L2-|---L3---|  L3=rightmost piece
# now we can find the min and max lengths
    Lmin = min(L1,L2,L3)
```

```
    Lmax = max(L1,L2,L3)
    Lmid = 1.-Lmin-Lmax
# variance = mean squared length - square of mean length
# std of the mean = sqrt(variance/(N-1))
    s += Lmin
    s2 += Lmin*Lmin
    l += Lmax
    l2 += Lmax*Lmax
    m += Lmid
    m2 += Lmid*Lmid
s /= N; s2 /= N; l /= N; l2 /= N; m /= N; m2 /= N
sigma_s = ((s2 -s*s)/(N-1))**0.5
sigma_l = ((l2 -l*l)/(N-1))**0.5
sigma_m = ((m2 -m*m)/(N-1))**0.5
print(' longest  piece: ',1.8*l,'+-',1.8*sigma_l)
print(' middle   piece: ',1.8*m,'+-',1.8*sigma_m)
print(' shortest piece: ',1.8*s,'+-',1.8*sigma_s)


'''
    Long and short of it.              long-n-short-v.py
    Generate N sticks of length 1.8m and break each at 2
    uniformly random points. Print the mean and std of
    the length of 3 pieces.
'''
import numpy as np
N = 1000000
x1 = np.random.random(N) # 1st break point
x2 = np.random.random(N) # 2nd break point
# lengths of left, middle and right pieces of the stick
# rescaling to actual length of the stick will be done later
# during printing. Until then, that length = 1.
L1 = np.minimum(x1,x2)
L2 = abs(x1-x2)
L3 = 1.-L1-L2       # |--L1--|-L2-|---L3---|
Lmin = np.minimum(np.minimum(L1,L2),L3)
Lmax = np.maximum(np.maximum(L1,L2),L3)
Lmid = 1.-Lmin-Lmax
s = Lmin.mean()
l = Lmax.mean()
m = Lmid.mean()
# std of the mean = standard deviation of data /sqrt(N-1)
sigs = Lmin.std()/(N-1)**0.5
sigm = Lmid.std()/(N-1)**0.5
sigl = Lmax.std()/(N-1)**0.5
print(' longest  piece: ',1.8*l,'+-',1.8*sigl)
print(' middle   piece: ',1.8*m,'+-',1.8*sigm)
print(' shortest piece: ',1.8*s,'+-',1.8*sigs)
```

PSCB57 FINAL EXAM 2019. QUIZ. CIRCLE Y[ES] OR N[O] AND SUBMIT THIS PAGE.

Quiz is worth up to **19%** of the total course score. Statements are sometimes tricky. A single word or number may be incorrect. **Any "[N]" answer circled MUST have at least one wrong word circled for credit.** Please disregard typos. Raise you hand - we will try to answer your 1 or 2 questions during exam, but be respectful, since this distracts your colleagues. Programming questions refer to Python ver.3. Assume all modules needed have been imported. To account for unintended ambiguity of questions 6 points will be added to your result. Good luck!

[N] The butterfly effect **refers to the two lobes** of the Lorenz attractor that may look like two wings of a butterfly

[N] The **secant method is Newton's method** of root finding applied to the math function y = sec(x)

[N] print(np.log(100.)) will output **2.0**

[Y] Two last 2 values in 1-D numpy array A are A[-2] and A[-1]

[Y] Python's tuples are immutable, i.e., their components cannot be modified

[Y] All Numpy array elements have the same type (e.g., int, float)

[N] All list elements **must have** the same type (e.g., int, float)

[N] Tridiagonal matrix problems can be solved in O(N) steps, faster than the general matrices that **need** $O(N^4)$ operations, where N = number of unknowns

[N] Suppose x = np.linspace(0,30,7). Then the loop for i in x: **will return error** message that type of x is float instead of the required int

[Y] Python evaluates expression 'fi'+'nal' as 'final'

[Y] In 800 MB of RAM, the maximum number of Python floats is 100 M

[Y] Computer capable of 100 GFLOPS, and bandwidth (supply rate of data to CPU) 40 GB/s can add two numpy float arrays, each filling 1 GB of RAM, in no less than 1/20 s.

[N] The smallest non-zero positive floating point number in Python is approximately 5e-324. It is known as **machine epsilon**.

[N] Sometimes the **truncation error of differentiation or integration algorithm vanishes, because all values on x axis are integers**, so there is no roundoff error.

[N] Suppose that the optical thickness of an object is $\tau = 0.7$. Then **30% of photons** get through it without absorption or scattering.

[N] NumPy module allows you to declare a 4-byte integer. Such a number can assume **65536** different values

[Y] One of two composite Simpson integration rules reads: $I = (f_0 + 4f_1 + 2f_2 + 4f_3 + 2f_4 + ... + 4f_{n-1} + f_n)h/3$

[Y] Variance of random variable X is the mean squared deviation of X from its arithmetic average $< X >$

[N] A 1-D random walker on integer axis N, with two absorbing boundaries at N=0 and N=100, starts from N==20 and walks K random steps of length 1. As K $\to \infty$, Gambler's Ruin (N=0) happens with probability **1/5**

[Y] Images recorded by Hubble Space Telescope are convolutions of true images of a given part of sky it is observing, convolved with its Point-Spread Function.

[Y] Roundoff error of the formula $f''(x) = [f(x-h) - 2f(x) + f(x+h)]/h^2$ is equal$O(h^{-2})$.

[N] Error of the root in bisection, as a function of the number of steps, has a constant slope = -2 on a **log-log plot**.

[Y] The midpoint integration method (composite rule) has error $\sim h^2$, where h = (b-a)/N is the step size.

[Y] Steepest descent method of multivariate optimization of a real function E(x,y,...) has a possible problematic behavior called zigzagging.

[Y] There exist sets of N linear equations with N unknowns, which cannot be solved by Gauss elimination.

[Y] There are 60 students in class, and their 3-digit endings of student id number can be taken as random. The chance that two or more endings coincide is greater than 0.5. Similarly, there is a chance larger than 0.5 that one or more birthdays of these students fall on the same calendar day.

[Y] Euler method for ODEs is first order, i.e., the error of the value(s) obtained after a fixed time depends on the time step dt as dt^1.

[N] Nelder-Mead method **requires** calculation of **gradient** of the minimized function but not the second derivatives

[N] If you decrease step size in a RK4 algorithm from h=0.01 to h = 0.002, then the truncation error in the value(s) obtained at the end of a fixed time interval will decrease **16** times.

[Y] Positions of planets in the Solar System become unpredictable in detail after several dozen million years

[N] Truncation error of a **3rd** order method applied to second-order ODE is $\sim h^6$

[Y] The general problem of 3 and more gravitating bodies has no analytical solution, and requires numerical solution.

[N] Trapezoidal integration formula has the roundoff error **is equal** $O(\varepsilon h^2)$, where ε is machine accuracy

[Y] Composite integration methods have truncation error of the 1-interval scheme, divided by step size h

[Y] Composite integration methods have truncation error order of the underlying scheme, multiplied by N (number of subintervals)

[N] Instructions m = 2*[4.12,.311,0.2]; print(m) **will print [8.24, .622, 0.4]**

[N] Monte Carlo methods using N random numbers converge to precise results with **error od order** $O(\sqrt{N})$, i.e. very slowly compared with most methods.

[Y] Python is a dynamically typed language; not only the value but also type of a variable may change in the program

[Y] Splines minimize the bending of an interpolating curve, and pass exactly through all data points.

[Y] This code will print 'True': x = 128; x = (x//128 == 1); if x: print(x)

[Y] It is hard to predict the logical value of expression (3/4-1/6)/7 == 1/12

[Y] Integration by Simpson rule or any other 2nd order rule, finds the area under a parabola exactly

[Y] Gauss quadrature is a method in which both the weights of samples of the integrated function and their positions on the x-axis are optimized.

[Y] In the body of a function, return() can be found in 0, 1, or more places

[Y] The first compiler was Fortran compiler from 1957. Compilers are programs that check and translate users program into elementary instructions for the processor (assembly language), and later translate the program into binary form.

[Y] The fist electro-mechanical computer was Bomba, built by Rejewski, Zygalski and Rozycki in 1938 to help break the German ENIGMA code. It was significantly expanded by Turing during WWII.

[Y] If we could halve the operating voltages inside the microprocessor, its power consumption would drop to 1/4 of the initial.

[Y] The wave equation $\partial^2 y/\partial t^2 = c^2\, \partial^2 y/\partial x^2$ is a linear, partial differential eq. Because of superposition principle, wave-trains can pass through one another and emerge unchanged

[N] RAM mean Random Access Memory. The word 'random' comes from **early memory systems using vacuum tubes**, which were very unreliable ('random').

[N] The Moore's law is ending, **because** multi-core CPUs have replaced single-core CPUs.

[Y] According to reports in ancient Roman literature, Antikythera mechanism correctly computed eclipses of the Sun and Moon

[Y] All top 500 supercomputers today use Linux operating system, none Windows or MacOS (which is also a kind of Linux)

[N] The first general-purpose quantum computer **was built** by IBM in 2015

[Y] If a differentiation stencil has truncation error = $h^2/4$, and roundoff error = 4e-16/h, then the smallest possible total error is achieved for h = 2e-5, and equals 1e-10.

[N] Fast Fourier transform algorithm is a method of quickly computing **the values of n-th high-order polynomial** from a list of its n+1 coefficients

[Y] Newton's method uses iteration $X_{n+1} = X_n - f(X_n)/f'(X_n)$

[N] **Nasdaq was a compute** that NASA had built for its lunar missions in 1970s

[Y] We can simulate N-body problems with local interactions of particles with $N \sim 10^{10}$

[Y] The behavior of numerical error as a function of integration step size is only following the theoretical relationships if the integrated function has finite derivatives everywhere

[Y] Bolshoi cosmological simulation computed about 14 billion years of evolution of dark and ordinary matter, including gas component, in region of space that grew to be about 1 billion light years across.

[Y] I have signed the booklet and any detached sheets containing my solutions with my name and student number