Phys 256 — Midterm 1

Student Name

We are now ready to do some physics with tools we have learned. This midterm will focus on finding the period $T(E)$ of oscillations of a particle of mass m in a one-dimensional potential $U(x)$ as a function of total energy E. In general, these calculations cannot be done analytically, and numerical methods are really a must.

Exam Structure

This exam consists of two parts:

1. Take-home assignment (AI allowed for coding only)

You will build a numerical package to calculate $T(E)$ and test it on several potentials $U(x)$. You may use AI and other resources to develop, test, and debug your code.

Deliverables:

- Code package for Problems 1–3 (ZIP file containing Python .py files).
- Report for Problem 7 (PDF; AI prohibited).
- User's Manual (1–2 pages, PDF; AI prohibited).

As in previous homeworks, submit your code and a short report on Problem 7 verifying that it works correctly. Problems 4–6 are optional practice for the in-class exam.

The User's Manual is new: Write a short guide *for yourself* explaining how to use, understand, and extend your code. Think of it as a "cheat sheet" for your own package that you may consult during the in-class exam.

2. In-class exam (AI prohibited)

You will apply your package to a new physical problem and extend it to new situations. You will be asked to:

- Analyze a new potential $U(x)$ using your package.
- Modify and generalize your code to handle new complications.

You will not have access to AI during this portion. Further details will be provided closer to the exam.

Your success on the in-class exam depends on understanding and being able to modify your own code.

- Keep your code simple, clear, and easy to modify.
- Use the User's Manual to consolidate your understanding and prepare a useful reference for the exam.
- Do not rely on AI-generated code or explanations that you do not understand. A working package alone will not prepare you for the in-class exam.

Instructions

Implement your own versions of central-difference differentiation, Newton-Raphson root finding, and Simpson's rule for numerical integration, following the requirements below.

Test your implementations against known results and benchmark their numerical errors.

You may use AI and external libraries to develop, test, and debug your code. **AI may not be used to write the report or User's Manual.**

Auxiliary functions

Problem 1:

Implement a central difference function to estimate the derivative of a function

$$f'_c(x) = \frac{f(x+h) - f(x-h)}{2h}$$

```
def cdiff(f, x, h):
    """
    Parameters
    ----------
    f : Function to differentiate.
    x : Point at which to evaluate the derivative.
    h : Step size.

    Returns
    -------
    dfdx : Approximate derivative at x.
    """
```

Listing 1: Central difference derivative prototype

Submit code to the autograder as `p1.py`. Code template also available on course web site.

Problem 2:

Implement the Newton–Raphson root-finding algorithm as discussed in class. This algorithm should find solution(s) to the equation

$$f(x) = 0$$

using the Newton-Raphson iterative update

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}.$$

```
def newton_raphson(f, xg, eps_x, eps_f):
    """
    Parameters
    ----------
    f : Function whose root is sought.
    xg : Initial guess.
    eps_x : Convergence tolerance for successive guesses.
    eps_f : Convergence tolerance for |f(x)|.

```

```
    Returns
    -------
    x0 : Approximate root.
    fx0 : Function value at x0.
    Neval : Number of evaluations of f.
    """
```

Listing 2: Newton-Raphson prototype

Note that this function should return three values x0, fx0, Neval.

Submit code to the autograder as `p2.py`. Code template also available on course web site.

Problem 3:

Implement Simpson's numerical integration algorithm to approximate

$$\int_a^b f(x)\,dx.$$

```
def mysimpson(f, a, b, Np):
    """
    Parameters
    ----------
    f : Function to integrate.
    a, b : Lower and upper integration limits.
    Np : Number of equally spaced points, including the endpoints a,b
       .

    Returns
    -------
    integral : Approximate integral.

    Note: you must return the approximate integral or raise the
       ValueError exception if something is wrong with the inputs.

    Here is an example of how to raise an exception:

    if z < 5:
        raise ValueError("z is below 5")

    It is your job to think of what could go wrong.
    """
```

Listing 3: Simpson integration prototype

Submit code to the autograder as `p3.py`. Code template also available on course web site.

Physics problems

Problem 4: Fit the data

You are provided with a data file named `dataUvsX.csv` containing two columns: x and U, where x is the independent variable and U is the measured value of the potential energy.

Which of the following 3 functions best describe or model the data and why so? Making plots, calculating the residuals, and calculating the χ^2 parameter would be good ideas.

1. $f(x) = ax^2 + bx + c$
2. $g(x) = a \exp\left[-\frac{(x-\mu)^2}{2\sigma^2}\right] + c$
3. $h(x) = a \cos\left[\omega(x - x_0)\right] + b(x - x_0) + c$

Calculate fit parameters for the best model. You will need them in the following problem.

Problem 5: Classical Turning Points

Interpret the best-fit function from the previous problem as the potential energy function $U(x)$. Assume that units are Joules (J). If a body with total energy E is moving in such a potential energy 'landscape' it cannot go beyond the region where $U(x) > E$.

The points where $U(x) = E$ are called turning points, since the velocity of an object is 0 at these points and this is where the object reverses its direction.

Within the x region of measured $U(x)$, use your root-finding algorithm from above to find all values of x satisfying

$$U(x) = E$$

where $E = -1$ J.

Generate a plot showing the modeled $U(x)$ as a function of x, a horizontal line for the total energy E, and dots for the turning points. Change the value of E and repeat this process.

Problem 6: Travel Time Between Turning Points

A particle of mass $m = 1$ kg moves in the best-fit potential $U(x)$ found above with total energy $E = -1$ J.

The particle's speed is

$$v(x) = \sqrt{\frac{2(E - U(x))}{m}}.$$

Therefore the travel time from the left turning point x_0 to the right turning point x_1 is

$$t(E) = \sqrt{\frac{m}{2}} \int_{x_0}^{x_1} \frac{dx}{\sqrt{E - U(x)}}.$$

Find the period $T(E) = 2t(E)$ of the round trip motion between the turning points.

Note: plugging the above formula straight into Simpson's method will lead to answers that diverge / blow up. Why is this? Use the debugger to step through your code and isolate the divergences. Then think about possible routes to avoid them.

Problem 7: Period of a Simple Pendulum

Using the formalism described above, analyze the period T of a simple pendulum as a function of the pendulum release height H normalized to the length of the pendulum L. Also normalize the period to $T_0 = 2\pi\sqrt{\frac{L}{g}}$. Assume that the pendulum's mass m is infinitely small, and neglect air drag.

Put a physicist's hat on: do you really need to know L and m to solve this problem numerically?