Physics 129L: Problem Set 6

Problem Set Submission Guideline

A) Github Submissions We will use GitHub for problem set submissions. To access the problem set, please **fork** and **clone** the **forked** repository to your local virtual machine. **Please complete the problem set in this forked directory.** Submit **a pull request** for merging into the main branch before the problem set due date.

B) .tar.gz File compression and submission on Github For each problem set, you are asked to submit the compressed version of the problem set to GitHub via git operation. Here is a step-by-step guideline:

1. Use the **tar** command to compress the problem set directory into a **single** ".tar.gz" file.
2. Obtain the sha256sum by running "sha256sum P2.tar.gz".
3. Echo the **full sha256sum** to a text file named "sha25sum_problem_set.txt".
4. Initialize a git repository named "Archive_P# (#: problem set number) on your local machine, and move both the "tar.gz" file and the "sha25sum_problem_set.txt" file to the repository.
5. Create an empty **public** directory under the **same name** in **your own GitHub account**.
6. **Push** this local repository to the remote repository.

Imports and built-in functions

In [1]:
```
import numpy as np
#Import in-built functions for different integration techniques
#For reference: https://docs.scipy.org/doc/scipy/reference/integrate.html
from scipy.integrate import quad, fixed_quad, romberg, dblquad
#For plotting
import matplotlib.pyplot as plt
%matplotlib inline
```

Monte Carlo Method v.s. Deterministic Quadrature

In this problem, you will be looking at the difference between various deterministic and non-deterministic methods. Let's consider an ellipsoid parametrized by,

$$\frac{x^2+y^2}{\beta^2}+\frac{z^2}{c^2}=1$$

where $-1 \leq x \leq 1$ and $-1 \leq y \leq 1$. Let's consider the surface element,

$$A=\int_{\partial V} 1 dA$$

A)

Write down the explicit integration formula in LaTeX for surface area. Remember to express the integration boundaries,

-------------------------write your expression below---

-----------------------------write your expression above-------------------------------------

hint: You should be able to express it using a single variable.

B)

Use the above formula approximate the area by using **two** deterministic quadrature techniques: 1) the midpoint rule and 2) Gaussian quadrature (you can use **your code from previous problem set**.) Verify the calculated values with the formulas below,

$$A = 2\pi\beta^2\left(1+\frac{c}{ae}\sin^{-1}(e)\right),\quad e = 1-\frac{\beta^2}{c^2}$$

You may realize that I did not provide the value of a and c. Plot the error as a heatmap with various β, c values ranging from [0.001,1000]. Does the value of a and c has any influence on the error?

```
In [ ]: '''--------------------Write your code below this line------------------------------'''


        '''--------------------Write your code above this line------------------------------'''
```

C)

Let's first consider the surface with non-deterministic quadrature techniques (Monte Carlo). As discussed in class, the following simple Monte Carlo simulation can be used to approximate a 2d integral,

$$\int_a^b f(x)dx = \lim_{N\to\infty}\frac{b-a}{N}\sum_{i=0}^{N} f(x_i),\quad X \sim U(a,b)$$

and we set $2\beta = c = 1$. For each sampling size, N=[10,100,1000,10000,100000], calculate the error, and plot them.

```
In [9]: '''--------------------Write your code below this line------------------------------'''


        '''--------------------Write your code above this line------------------------------'''

Out[9]: '--------------------Write your code above this line------------------------------'
```

Box–Muller transform

Let's take a closer look at a joint probability,

$$p(x,y) = \frac{1}{2\pi}e^{-(x^2+y^2)/2}$$

and changing from x,y to θ, R, we have,

$$p(R,\theta) = p(x,y)\frac{\partial|x,y|}{\partial|\theta,R|} = \frac{1}{2\pi}Re^{-(R^2)/2}$$

where $\frac{\partial|x,y|}{\partial|\theta,R|} = \mathrm{det|J|} = \mathrm{R}$. This is called the Rayleigh distribution, which coincides with the χ distribution with two degrees of freedom (DOF). Let's consider two random variables that follow a standard normal distribution, with X and Y denoted as $X, Y \sim \mathcal{N}(0,1)$, representing two components of a vector. The cumulative distribution function (CDF) of the joint probability is given,

$$P(\sqrt{X^2+Y^2} \leq R) = \int_0^{2\pi}\int_0^R r\frac{1}{2\pi}e^{-r^2/2}drd\theta.$$

It gives the same PDF as the probability density above.

D)

Using the Box–Muller transform, write a python function that generates Gaussian distributed samples (return a numpy array) with mean μ and standard deviation σ.

In [ ]:
```
'''--------------------Write your code below this line------------------------------''

'''--------------------Write your code above this line------------------------------''
```

E)

Do a simple Monte Carlo simulation, this time using different Gaussian-distributed samples as mentioned above,

$$\int_a^b f(x)dx = \lim_{N\to\infty} \frac{b-a}{N} \sum_{i=0}^{N} f(x_i), \quad X \sim N(\mu, \sigma)$$

and we set $2a = c = 1$. Let's first assume $\mu = 0$ and $\sigma = 1$. For each sampling size, N=[10,100,1000,10000,100000], calculate the error, and plot them.

Next, test various μ and σ with a fixed $N = 10000$, calculate the error, and plot them. How is it different from C)?

In [ ]:
```
'''--------------------Write your code below this line------------------------------''

'''--------------------Write your code above this line------------------------------''
```

Rejection Sampling

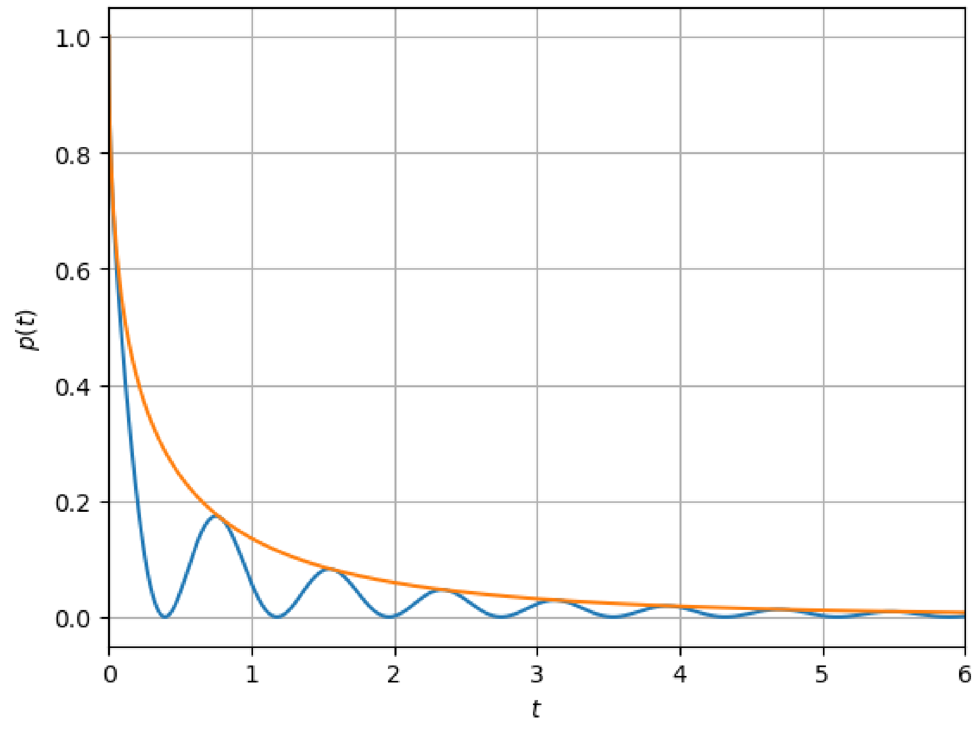
Rejection sampling is used to generate data points that follow a particular complicated distribution. Let's say that the probability of observing a particle decay event at time t follows the probability density function (PDF),

$$p(x) = e^{-bt} cos^2(at), \quad t \geq 0$$

.

See below for an example when $a = 4b = 4$ (If you are interested, you can refer to the lecture notes on the Lorentzian function for energy dissipation).

In [23]:
```python
# Example
t = np.linspace(0, 6, 100000)
y = np.exp(-2*np.sqrt(t))*np.cos(4*t)**2
y2 = np.exp(-2*np.sqrt(t))
plt.plot(t, y)
plt.plot(t, y2)
plt.grid(True)
plt.xlim(0, 6)
plt.xlabel('$t$')
plt.ylabel('$p(t)$')
plt.show()
```



A)

Write a rejection sampling function by using a **uniform proposal function**,

$$f(t) \sim U(0, t_f)$$

to sample the PDF discussed above, with $a = 4b = 4$.

It should return a N sample numpy array. How do you select the t_f? Plot the resulting sample histogram with N=[100,1000,10000].

In [ ]:
```python
'''--------------------Write your code below this line------------------------------'''


'''--------------------Write your code above this line------------------------------'''
```

B)

Write a rejection sampling function by using a **exponential proposal function**,

$$f(t) \sim Exp(1) = e^{-2t}$$

to sample the PDF discussed above, with the same $a = 4b = 4$.

It should return a N sample numpy array. Plot the resulting sample histogram with N=[100,1000,10000]. Make a comparison between the uniform proposal function and exponential proposal function at various sample sizes.

In [ ]:
```
'''--------------------Write your code below this line------------------------------''

'''--------------------Write your code above this line------------------------------''
```

Optimization

Gradient descent, Metropolis–Hastings algorithm, Simulated Annealing

Gradient descent is a deterministic method for optimization. It requires the function to be differentiable and convex. The general formula for gradient descent is as follows:

Let $H(\theta)$ be the cost or loss function, where θ represents the model parameters. The goal is to find the optimal θ that minimizes $H(\theta)$.

The process of gradient descent involves starting with an initial guess for θ_i, then iteratively applying the update rule until convergence. Convergence is typically determined by monitoring the change in the cost function or the norm of the gradient.

The update rule for gradient descent is as follows:

$$\theta_{i+1} = \theta_i - \alpha_i \cdot \nabla H(\theta_i)$$

where α_i The learning rate, which is a hyperparameter that determines the step size of each update. The choice of the learning rate (α) is crucial in gradient descent, as it can affect the algorithm's convergence and stability. It often requires experimentation to find an appropriate learning rate for a specific problem. The gradient (∇H) is a vector that contains the partial derivatives of the cost function with respect to each parameter in θ. It represents the rate of change of the cost function with respect to each parameter and guides the updates.

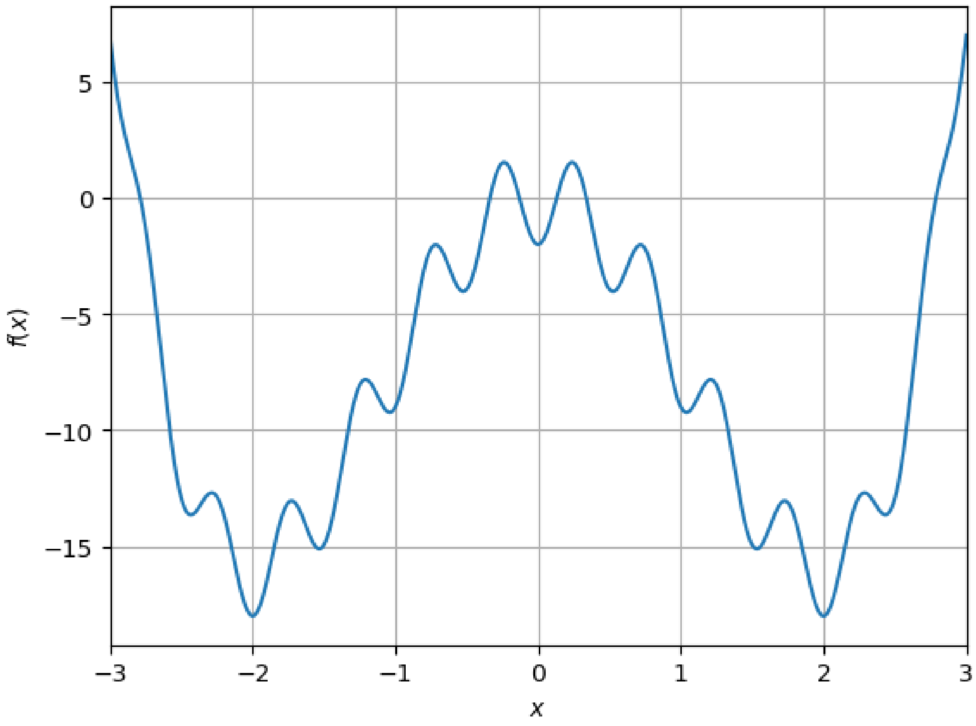
$\phi 4$ theory in 1D

Let's say you have a noisy $\phi 4$ theory in 1D, given by,

$$H = \theta^4 - 8\theta^2 - 2cos(4\pi\theta),$$

where θ is an order parameter. You want to find the ground state order parameter and energy (see the figure below).

In [50]:
```
x = np.linspace(-3, 3, 1000)
y = x**4-8*x**2 - 2*np.cos(4*np.pi*(x))
plt.plot(x, y)
plt.grid(True)
plt.xlim(-3, 3)
plt.xlabel('$x$')
plt.ylabel('$f(x)$')
plt.show()
```



A)

Using the gradient descent method, locate the global minimum starting with three initial guesses $\theta_0 = -1, 0.5, 3$. For each descent step, plot a **red dot** on the above plot and save it locally until it converges. Make a **video** by processing your saved figures (you may want to look into "cv2.VideoWriter"). You should tune the learning parameter at each step!

Do you get consistent results?

In [ ]:
```
'''--------------------Write your code below this line------------------------------'''


'''--------------------Write your code above this line------------------------------'''
```

Elementary Hamiltonian Monte Carlo: Metropolis–Hastings algorithm

As we discussed in class, Metropolis–Hastings algorithm is a Monte Carlo method that is used for optimization. In here, we will look at some basics. In Bayesian inference, the posterior can be expressed as Boltzmann factors,

$$P(\theta) = \frac{e^{-\beta H(\theta)}}{Z},$$

where H is Hamiltonian, $\beta = 1/kT$, and Z is the partition function. Let's start with an initial parameter guess θ_0. Let's randomly move from $\theta_1 \rightarrow \theta_0 + \Delta\theta$, where the step follows a Gaussian $\Delta\theta \sim \mathcal{N}(0, \sigma)$ (Markov process). Note, it must be symmetric. Then, the ratio,

$$r = \frac{e^{-\beta H(\theta^*)}}{e^{-\beta H(\theta)}} = e^{-\beta H(\theta^*)+\beta H(\theta)} = e^{-\beta \Delta H(\theta^*, \theta)}.$$

If $r > 1$, we accept it and set $\theta_1 \rightarrow \theta_0$. On the other hand, $r < 1$, we accept it with probability r and set $\theta_1 \rightarrow \theta_0$.

B)

Use the Metropolis–Hastings algorithm above to estimate the minimum of the noisy $\phi 4$ with initial guesses $\theta_0 = -1, 0.5, 3$. You should try different β.

In [10]:

```
'''--------------------Write your code below this line------------------------------''

'''--------------------Write your code above this line------------------------------''
```

Out[10]:

```
'--------------------Write your code above this line------------------------------'
```

Simulated Annealing

Simulated Annealing is a probabilistic optimization algorithm inspired by the annealing process in metal. The cooling schedule is usually defined as the following,

$$\beta_{i+1} = \beta_i + \delta_i,$$

where we update the inverse temperature each step. This update will change the Metropolis criterion,

$$r_i = e^{-\beta_i \Delta H(\theta^*, \theta)} > u_i$$

where $u_i \sim U(0, 1)$. The cooling schedule is a critical aspect of Simulated Annealing. It determines the rate at which the temperature decreases.

C)

Add a cooling schedule to the Metropolis–Hastings algorithm above. Then, estimate the minimum of the noisy $\phi 4$ with initial guesses $\theta_0 = -1, 0.5, 3$. You should try different cooling schedule e.g. δ_i. Make a graphical comparison of the convergence steps with cooling and without cooling.

In [40]:

```
'''--------------------Write your code below this line------------------------------''

'''--------------------Write your code above this line------------------------------''
```

Out[40]:

```
'--------------------Write your code above this line------------------------------'
```

In [ ]: