Metastability and Markov

(Sethna, "Entropy, Order Parameters, and Complexity", ex. 12.XXX)

The hints below will help you solve parts (e) and (f) of this question, where we will numerically evaluate the slowest decaying mode and barrier crossing time by computing eigenstates of the "quantum" Hamiltonian.

In the Mathematica hints, we shall solve the differential equations directly. In the Python hints, we shall instead construct a Hamiltonian matrix by discretizing space into segments of length dx, and then finding the lowest eigenvalue of that Hamiltonian.

Import packages

In [ ]:
```
# Sometimes gives interactive new windows
# Must show() after plot, figure() before new plot
# %matplotlib

# Adds static figures to notebook: good for printing
%matplotlib inline

# Interactive windows inside notebook! Must include plt.figure() between plots
# %matplotlib notebook

# Better than from numpy import *, but need np.sin(), np.array(), plt.plot(), e
import numpy as np
import matplotlib.pyplot as plt

from scipy.sparse import diags
from scipy.linalg import eig_banded
from scipy import special
```

Define functions to evaluate the potential $V(x)$ and effective quantum potential $V_{\mathrm{eff}}(\mathrm{x})$ and compute a discretized Hamiltonian matrix H. We assume $\eta = 1$ and $k_B T = 1/2$. We will use this matrix to compute the slowest decaying mode for parts (e) and (f).

In [ ]:
```
def V(x):
    """
    Returns the cubic potential -x^3/3 + x
    """
    return  ...


def Veff(x, eta=1, kbT=1/2):
    """
    Returns the effective quantum potential evaluated at x.
    Calculate Veff by pluging the cubic potential into the
    effective potential you found in part (g). You should find
    (x^2 - 1)^2/(4 kbT eta) + x/eta (notice this is independent of eta)
```

```
    """
    return ...


def hamiltonian(eta=1, kbT=1/2, x0=5, dx=0.001):
    """
    Returns a discretized hamiltonian matrix for numerical eigendecomposition
    +/-x0 - finite boundary conditions
    dx - discrete grid size
    """
    N = round(2*x0/dx) #total number of elements on grid (boundary at +/- x0)
    x = np.arange(-x0,x0,dx) #make discrete grid

    ## Finite difference Kinetic Energy
    #d^2/dx^2 is a matrix with -2 on the diagonal and 1 on the super/subdiagona
    #In the first list put the elements: -2, 1, and 1. In the second list put
    #0 = diagonal, 1 = superdiagonal, -1 = subdiagonal
    T = -kbT/eta*diags([..., ..., ...], [..., ..., ...], shape=(N, N))/(...)**

    ## Potential Energy
    #Effective quantum potential on discrete grid
    Veff_x = Veff(x, eta, kbT)
    #Matrix potential (Veff_x on diagonal)
    Veff_mat = diags([...], [0])

    H = T + Veff_mat

    return H.toarray()

def eigenSystem(eta=1, kbT=1/2, x0=5, dx=0.001, Nstates=1):
    """
    Returns the Nstates lowest energy eigenvalues and associated eigenstates f
    the quantum system corresponding to the Fokker-Plank equation with a cubic
    """
    H = hamiltonian(eta, kbT, x0, dx)

    banded = np.array([np.diagonal(H), np.append(np.diagonal(H,1),0)])
    eigs = eig_banded(banded, select="i", select_range=[0,0], lower=True)
    ##eigh_tridiagonal(diagonal(H), diagonal(H,1), select="i", select_range=[0,
    # If you have scipy version 1.0 or newer, eigh_tridiagonal (add from scipy
    # will give you slightly faster performance

    return eigs
```

(e) *For the cubic potential (eqn 7), numerically compute the eigenstate of the transformed diffusion equation with smallest eigenvalue. What does the eigenvalue predict for the lifetime? How nearly does it agree with the classic calculation of Kramers:*

$\lambda_0 \approx \sqrt{K\widetilde{K}}/(2\pi\eta)\exp(-E/k_BT)$

In [ ]:

```
# Compute eigenvalue and eigenstate
x0 = 5
dx=0.001

val, vec = ...
```

```
val = val[0]
vec = np.transpose(vec)[0]
```

In [ ]:
```
# Plot the potential, notice where the well and barrier are located
x = np.arange(-x0,x0,dx)

plt.plot(x, ...)
plt.xlabel("x",size=20)
plt.ylabel("V(x)",size=20)
plt.ylim([-10,10])

plt.show()
```

In [ ]:
```
# Compare eigenvalue approximation for escape time to analytical result

tauNumerical = 1/(...)
tauKramers = 1/(np.sqrt(... * ...)/(2 * np.pi * ...) * np.exp(-(...)/(...)))

# Percent difference between numerical and analytical result
diff = 100*abs(...)/tauNumerical

print(tauNumerical)
print(tauKramers)

print(str(round(diff,4))+"% difference")
```

(e) ... What does the eigenvalue predict for the lifetime? How nearly does it agree with τ from the Kramers calculation?

(f) Using the corresponding eigenstate ρ_0, plot the slowest decaying mode $\rho_0(x) = \sqrt{\rho^*}\sigma_0$, normalized to one, along with the Boltzmann distribution $\rho^*(x)/Z$ and the Boltzmann probability distribution in the approximation that the well is quadratic. Explain how ρ_0 differs from the quadratic approximation, and why this is physically sensible. Explain how ρ_0 differs from ρ^*, and how it resolves an important question about how to determine the metastable probability `in the well'.

In [ ]:
```
# Compute slowest decaying mode

# Boltzmann distribution for cubic potential
boltz = np.array(np.exp(...))

# Compute slowest decaying mode from eigenstate 'vec'
mode = (vec*np.sqrt(...))

# Plot the unnormalized mode
plt.plot(x,mode)
plt.legend([r"$\rho_0$"],prop={'size': 15})
plt.xlabel("x",size=20)
plt.ylabel("Density",size=20)

plt.show()
```

In [ ]:
```
# Now normalize your slowest decaying mode
# We only know the slowest decaying mode on a discrete grid, so the norm is gi
# adding up the points
```

```
# Due to numerical errors your slowest decaying mode may blow up at one of the
# numerical eigenstate doesn't exactly cancel the blow up from the Boltzmann di
# If this occurs, use cutoff to restrict the normalization calculation to [-xLi
xLim = ...
# boltz, x, and mode are lists. 'cutoff' gives the range of list elements of in
cutoff = round((x0-abs(xLim))/dx)+1
norm = sum(mode[cutoff:-cutoff])*dx

# Adjust Z manually so that the Boltzmann distribution best matches the slowest
# inside the well or approximate Z by normalizing over the inside of the potent
# If you choose the later option, you will want to restrict the normalization t
# Use your plot of the potential above to choose xMin and xMax
xMin = ...
xMax = ...

# Pick range corresponding to inside of the well
cutoff1 = round((x0-abs(xMin))/dx)
cutoff2 = round((x0-abs(xMax))/dx)

Z = sum(boltz[...:-...])*...

# Compute the Boltzmann distribution in the approximation that the well is quad
boltzQuadratic = np.exp(...)
ZQuadratic = sum(...)*...
```

In [ ]:

```
# Compare Boltzmann distribution to mode
plt.figure(1,[9,7])
plt.plot(x[cutoff:-cutoff],V(x[cutoff:-cutoff]))
plt.plot(x[cutoff:-cutoff],boltz[cutoff:-cutoff]/Z)
plt.plot(x[cutoff:-cutoff],mode[cutoff:-cutoff]/norm)
plt.plot(x[cutoff:-cutoff],boltzQuadratic[cutoff:-cutoff]/ZQuadratic,'--')
plt.ylim([-1,1])
plt.legend([r"$V(x)$",r"$\rho^*$",r"$\rho_0$","Quadratic Well"],prop={'size': 1
plt.xlabel("x",size=20)
plt.ylabel("Density",size=20)


plt.show()
```