

Metastability and Markov

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

© 2019, James Sethna, all rights reserved.

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 [ ]: %pylab inline
        from scipy import *
        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 $V_{\text{eff}}(x)$ 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 plugging 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 = 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/subdiagonal divided by dx^2
    #In the first list put the elements: -2, 1, and 1. In the second list put the corresponding diagonal
    #0 = diagonal, 1 = superdiagonal, -1 = subdiagonal
    T = -kbT/eta*diags([..., ..., ...], [..., ..., ...], shape=(N, N))/(...)**2

    ## 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 for
    the quantum system corresponding to the Fokker-Plank equation with a cubic potential and noise g
    """
    H = hamiltonian(eta, kbT, x0, dx)

    banded = array([diagonal(H), append(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=[0,0])
    # If you have scipy version 1.0 or newer, eigh_tridiagonal (add from scipy.linalg
    g import eigh_tridiagonal)
    # 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:
 $\lambda_0 \approx \sqrt{K \tilde{K}} / (2\pi\eta) \exp(-E/k_B T)$

```
In [ ]: # Compute eigenvalue and eigenstate
x0 = 5
dx=0.001

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

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

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

show()
```

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

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

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

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

show()
```

```

In [ ]: # Now normalize your slowest decaying mode
# We only know the slowest decaying mode on a discrete grid, so the norm is given by
# adding up the points
# Due to numerical errors your slowest decaying mode may blow up at one of the bound
aries (the
# numerical eigenstate doesn't exactly cancel the blow up from the Boltzmann distrib
ution)
# If this occurs, use cutoff to restrict the normalization calculation to [-xLim, xL
im]
xLim = ...
# boltz, x, and mode are lists. 'cutoff' gives the range of list elements of interes
t
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 deca
ying mode
# inside the well or approximate Z by normalizing over the inside of the potential w
ell.
# If you choose the later option, you will want to restrict the normalization to the
domain [xMin, xMax]
# 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 quadratic
boltzQuadratic = exp(...)
ZQuadratic = sum(...)*(...)

```

```

In [ ]: # Compare Boltzmann distributions to mode
figure(1,[9,7])
plot(x[cutoff:-cutoff],V(x[cutoff:-cutoff]))
plot(x[cutoff:-cutoff],boltz[cutoff:-cutoff]/Z)
plot(x[cutoff:-cutoff],mode[cutoff:-cutoff]/norm)
plot(x[cutoff:-cutoff],boltzQuadratic[cutoff:-cutoff]/ZQuadratic,'--')
ylim([-1,1])
legend([r"$V(x)$",r"$\rho^*$",r"$\rho_0$","Quadratic Well"],prop={'size': 15})
xlabel("x",size=20)
ylabel("Density",size=20)

show()

```