Activated rates and the saddle-node transition

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

This addresses parts (i) and (j) of this exercise, where you 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.

Define functions to compute the exact scaling function $\mathcal{T}(\alpha)$ and escape time τ. For simplicity, in the numerical portion of the exercise we shall assume η = g = a = 1 throughout.

```
𝒯Exact[α_] := 2^(1/3) Pi^2 (AiryAi[-2^(2/3) α]^2 + AiryBi[-2^(2/3) α]^2)
τExact[α_, g_: 1] := 1/g^(2/3) 𝒯Exact[α]
```

Construct the Hamiltonian for numerical calculations. Fill in the cubic potential and the expression for the effective quantum potential. Again, we assume η = g = a = 1. Notice η cancels in the expression for Veff(x)

```
V[ϵ0_, a_: 1, η_: 1][x_] := ...
Veff[x_, ϵ0_, a_, g_] := ...
H[σ_, ϵ0_, a_, g_] := -1/2 g^2 σ''[x] + Veff[x, ϵ0, a, g] σ[x]
```

Compute ground state eigenvalue and eigenfunction using NDEigenststem function

+/-x0 - finite boundary conditions for numerical evaluation

dx - discrete grid size

How should σ behave at the boundary?

```
eigenSystem[ϵ0_, g_, a_: 1, x0_: 5, dx_: 0.01, Nstates_: 1] :=
 NDEigensystem[{H[...], DirichletCondition[σ[x] == ..., True]},
  σ[x], {x, ...}, Nstates, Method → {"SpatialDiscretization" ->
     {"FiniteElement", {"MeshOptions" → {MaxCellMeasure → dx}}}}]
```

(i) For the cubic potential, numerically compute the eigenstate of the transformed diffusion equation with smallest eigenvaluefor α = -2. What does the eigenvalue predict for the lifetime? How nearly does it agree with τ from eqn (11)? Using the corresponding eigenstate, plot the slowest decaying mode ρ(x) = $(\rho^*)^{1/2}$ σ(x), normalized to one, along with the Boltzmann distribution ρ^*(x)/Z. How well can you match the two inside the well, by varying Z?

Choose ϵ_0 so that α = -2 (since we have set a=g=1 the relationship between ϵ_0 and α is very simple).

```
ϵ0 = ...;
α = ...;
Print["α = ", α]
{val, vec} = eigenSystem[...] // Flatten
```

Plot the potential, notice where the well and barrier are located

```
Plot[..., {x, ...}, PlotRange → {...}, AxesLabel → {"x", "V(x)"}]
```

Compare escape time from eigenvalue to analytical result. We compute the percent difference between numerical and analytical result.

```
τNumerical = 1 / (...)
τAnalytical = τExact[...] // N
diff = 100 Abs[(...)] / τAnalytical;
Print[diff, "% difference"]
```

(i) ... What does the eigenvalue predict for the lifetime? How nearly does it agree with τ from eqn (11)? ...

(i) ... Using the numerical eigenfunction, compute the slowest decaying mode, normalize the result.

Due to numerical errors your slowest decaying mode may blow up at one of the boundaries (the numerical eigenstate doesn't exactly cancel the blow up from the Boltzmann distribution). If this occurs, use cutoff to restrict the normalization calculation to [-xLim, xLim]

```
boltz = Exp[...];
mode = Sqrt[...] vec;
xLim = ...;
Plot[mode, {x, -xLim, xLim}, PlotRange → All,
 PlotLegends → {"ρ₀"}, AxesLabel → {"x", "Density"}]
norm = NIntegrate[..., {x, -xLim, xLim}];
```

Adjust Z manually so that the Boltzmann distribution best matches the slowest decaying mode inside the well or approximate Z by normalizing over a region near the potential well. If you choose the later option, you will want to restrict the normalization to the domain [xMin, xMax] to integrate over the well (see plot of the potential above)

```
xMin = ...;
xMax = ...;
Z = NIntegrate[..., {x, ..., ...}];
```

```
Plot[{mode / norm, boltz / Z}, {x, -xLim, xLim}, PlotRange → {0, 1},
 PlotLegends → {"ρ₀", "ρ*"}, AxesLabel → {"x", "Density"}]
```

(i) ... How well can you match the two inside the well, by varying Z?

(j) Repeating the above steps, compute the slowest decaying mode for α=0. Does the eigenvalue give τ? Why or why not? Where is the slowest decaying mode peaked? Why do the particles that are slowest to escape sit there?

Choose ϵ0SN so that αSN = 0.

```
ϵ0SN = ...;
αSN = ...;
Print["α = ", αSN]
{valSN, vecSN} = eigenSystem[...] // Flatten
```

Plot the potential at the saddle node bifurcation. Notice where the slope is minimum

```
Plot[..., {x, ...}, PlotRange → {...}, AxesLabel → {"x", "V(x)"}]
```

Compare escape time from eigenvalue to analytical result. We compute the percent difference between numerical and analytical result.

```
τNumericalSN = 1 / (...)
τAnalyticalSN = τExact[...] // N
diffSN = 100 Abs[(...)] / τAnalyticalSN;
Print[diffSN, "% difference"]
```

(j) ... Does the eigenvalue give τ? Why or why not? Where is the slowest decaying mode peaked? ...

(j) ... Using the numerical eigenfunction, compute the slowest decaying mode, normalize the result. ...

Due to numerical errors your slowest decaying mode may blow up at one of the boundaries (the numerical eigenstate doesn' t exactly cancel the blow up from the Boltzmann distribution). If this occurs, use cutoff to restrict the normalization calculation to [-xLim, xLim]

```
boltzSN = Exp[...];
modeSN = Sqrt[...] * ...;
xLim = ...;
Plot[modeSN, {x, -xLim, xLim}, PlotRange → All]
normSN = NIntegrate[..., {x, -xLim, xLim}];

Plot[{modeSN / normSN}, {x, -xLim, xLim}, PlotRange → {0, 1},
 PlotLegends → {"ρ₀"}, AxesLabel → {"x", "Density"}]
```

(j) ... Where is the slowest decaying mode peaked? Why do the particles that are slowest to escape sit there?

Optional: For a range of positive and negative α, plot the eigenvalue approximation to the scaling function $\mathcal{T}(\alpha)$ = (g *a*)$^{2/3}$ τ = τ (since g = a = 1) and compare to the exact formula Eq. (4).

```
eigApprox = Table[{..., ... / eigenSystem[ϵ][[1, 1]]}, {ϵ, ..., ..., 0.1}];;

theoryPlot = LogPlot[... [α], {α, ...},
    PlotStyle → Black, PlotLegends → LineLegend[{"Analytical"}]];

Show[theoryPlot,
 ListLogPlot[{eigApprox}, PlotLegends → LineLegend[{"Eigenvalues"}]],
 PlotRange → All, AxesLabel → {"α", "𝒯(α)"}]
```