numerical_stability

November 5, 2018

1 Numerical Stability

1.1 Paul Fanto

1.1.1 11/4/2018

2 Some caveats

1. I am *not* an expert on numerical stability.
2. This talk was written somewhat quickly.
   - My goal is to talk about my experience of stability issues from my research in computational quantum physics, share some tips and tricks that I've found helpful, and maybe prompt a discussion.

3 What is numerical stability?

- Numerical stability refers to the **sensitivity of the outputs of a numerical method to small errors in its input.**
- At a more basic level, numerical stability is the statement that floating point numbers do not behave exactly like real numbers.
- A **stable** or **well-conditioned** numerical method will consistently yield an input closet to the right answer even if a small error exists in the input.
- An **ill-conditioned** method will cause significant errors. Ultimately, these errors will destroy entirely your result and you will be sad.

4 Some definitions

- A numerical method is **backward stable** if its output is the exactly right answer to a perturbed input, i.e. $\hat{f}(x) = f(x + \delta x)$, $\delta(x) \sim O(\epsilon_{\text{mach}})$
- A numerical method has **mixed stability** if its output is *nearly* the right answer to a perturbed input, i.e. $\delta f = \hat{f}(x) - f(x + \delta x) \sim O(\epsilon_{\text{mach}})$, where $\delta(x) \sim O(\epsilon_{\text{mach}})$.

I will not discuss these in detail. For references, see MIT Lecture notes of Eric Liu at http://web.mit.edu/ehliu/Public/Yelp/conditioning_and_precision.pdf

5 Basic approach of this talk

- People who write standard-use numerical methods packages (e.g. LAPACK, Numpy) must worry about the numerical stability of their built-in methods, e.g. numpy.linalg.eig.
- In this talk, I will assume that you have access to one of these libraries.
- The basic problem is how to use them to develop methods for solving physics problems of interest that are stable.

6 Linear algebra

- Most of what we do as physicists.
- The basic idea of stability in linear algebra is that **you have to be careful about diverging scales**.
- The scale range of a matrix A is measured by the **condition number** $c(A)$, given by the ratio of the largest to the smallest *singular value* of the matrix.
  - Given an $n \times n$-dimensional matrix A, one may always stably compute $A = U\Sigma V^\dagger$, where $\Sigma = \text{diag}(\sigma_1, ..., \sigma_n)$ in descending order and U, V are unitary matrices.
  - $c(A) = \sigma_1/\sigma_n$

7 Examples

- Linear equations: if you want to solve $Ax = b$ given b, A, then $|\frac{\delta x}{x}| = c(A)|\frac{\delta b}{b}|$
  - So if $c(A)$ is sufficiently large, you will destroy all accuracy of x. Effectively, A is singular for the purposes of solving the linear system.
- Example from quantum many-body
  - One can express the partition function of a fermionic system as $Z \propto \int d\sigma G_\sigma \det[1 + U_\sigma]$ where $U_\sigma = U_\sigma(N)U_\sigma(N-1)...U_\sigma(1)$ for large N. Each matrix U has a fairly wide set of scales.
  - If you do not try to stabilize the integrand, then any method for computing the integral (e.g. quantum Monte Carlo) will fail.

8 Stable decompositions.

- Any matrix can be stably decomposed into the SVD from $A = UDV^\dagger$, D diagonal, U, V unitary.
- Another useful decomposition is the QDR decomposition: $A = QDR$, where D is diagonal, Q is orthogonal, and R is upper triangular with 1's on the diagonal
- Similarly, there are QL and LQ decompositions.
- The benefit is to separate the scales and allow methods to be implemented stably.

9 A painful example from my past (1st year)

- Computation of a partition function for a system of independent fermions, projected onto a specific particle number.

$Z_N = \mathrm{Tr}\hat{P}_N e^{-\beta\hat{H}} = \frac{1}{N_s}\sum_m e^{-i\phi_m N}\mathrm{Tr}e^{i\phi_m\hat{N}}e^{-\beta\hat{H}}$.

- Pairing correlations: $\hat{H}$ does not commute with $\hat{N}$.
- Basically, this means that the trace on the far right-hand side is given by the following determinant: $\mathrm{Tr}e^{i\phi_m\hat{N}}e^{-\beta\hat{H}} = \det[\mathbf{1} + \mathbf{W}^\dagger e^{i\phi_m\mathbf{N}}\mathbf{W}e^{-\beta\mathbf{H}}]$
- **Major scale difference**

- First stabilization step is to separate the scales. Use $\det[AB] = \det[A]\det[B]$

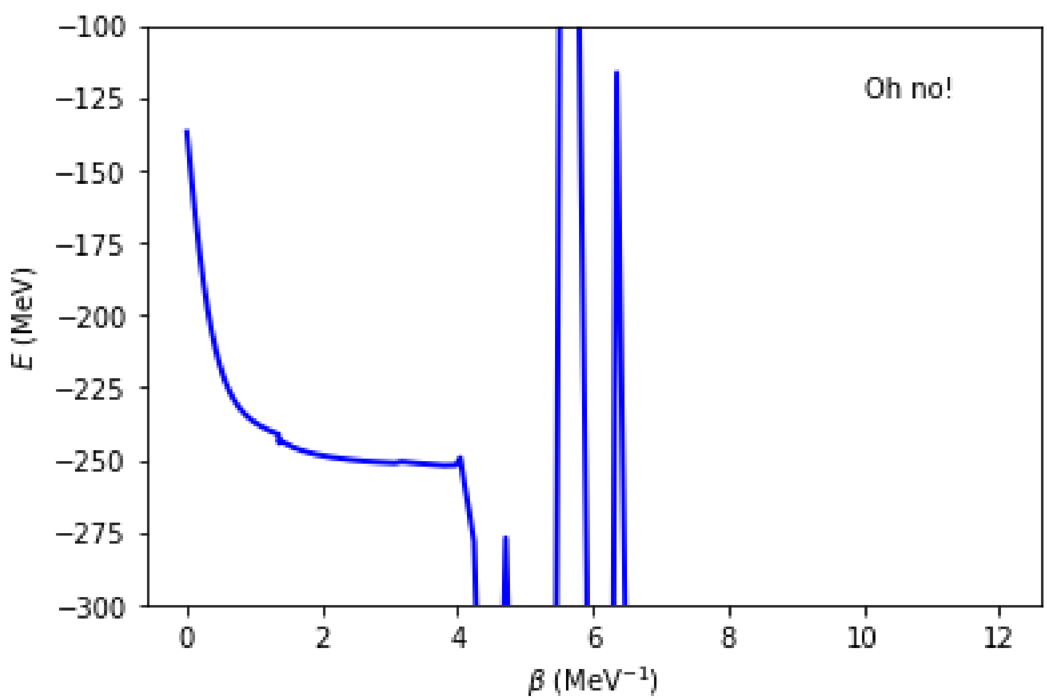
$$\det[\mathbf{1} + \mathbf{W}^\dagger e^{i\phi_m\mathbf{N}}\mathbf{W}e^{-\beta\mathbf{H}}] = \det[\mathbf{W}^\dagger e^{-i\phi_m\mathbf{N}}\mathbf{W} + e^{-\beta\mathbf{H}}]$$

- Is this enough?

In [4]:
```
import numpy as np
import matplotlib.pyplot as plt

data_unstabilized = np.loadtxt("Sm150_hfbprojection_unstab.dat")
beta = data_unstabilized[:,0]
E_N = data_unstabilized[:,1]
```

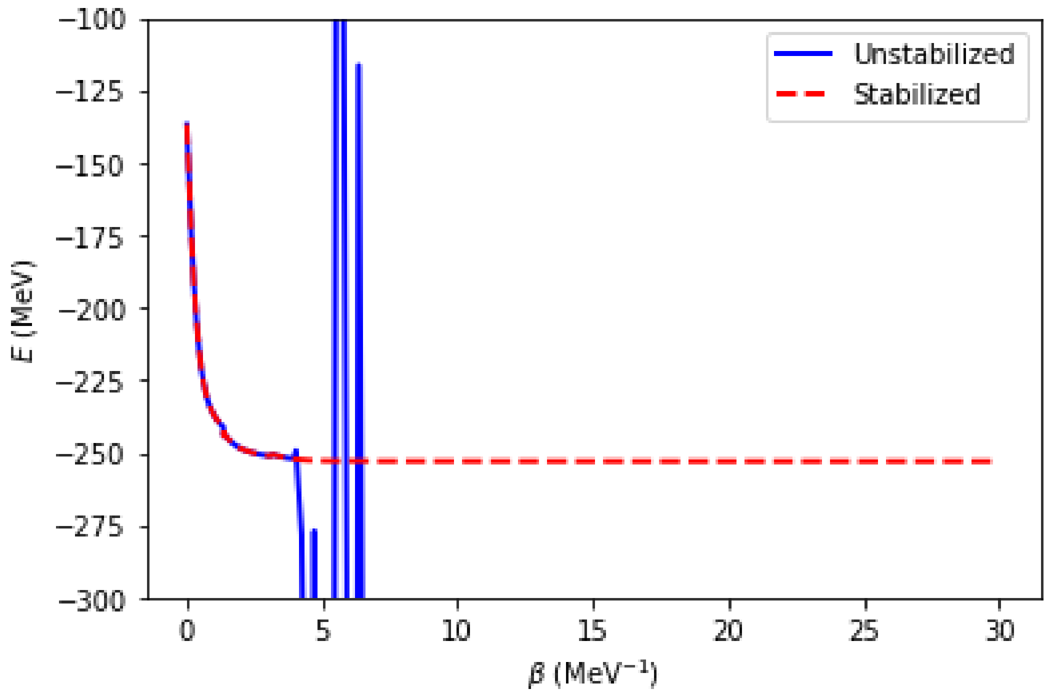
In [5]:
```
plt.figure(1)
plt.xlabel(r"$\beta$ (MeV$^{-1}$)")
plt.ylabel("$E$ (MeV)")
plt.plot(beta,E_N,'b-',lw=2,label='Unstabilized')
#plt.legend(loc='best')
plt.text(x=10.,y=-125,s="Oh no!")
plt.ylim([-300,-100])
plt.show()
```



- The competing scales in the central matrix destroy the accuracy. What can be done?
- Use a QDR decomposition. Then $\det A = \det Q \det D \det R$. Q, R are well-conditioned, so their determinants can be computed stably.
- Does this work?

In [6]:
```
E0_N = -2.52862892E+02
data_stabilized = np.loadtxt("Sm150_hfbprojection.dat")
beta = data_stabilized[:,0]
E_N_stab = data_stabilized[:,1]
E_N_stab += E0_N

plt.figure(1)
plt.xlabel(r"$\beta$ (MeV$^{-1}$)")
plt.ylabel("$E$ (MeV)")
plt.plot(beta,E_N,'b-',lw=2,label='Unstabilized')
plt.plot(beta,E_N_stab,'r--',lw=2,label='Stabilized')
plt.legend(loc='best')
plt.ylim([-300,-100])
plt.show()
```



Success!

10 Two other topics

- Regularization of sums
- Bound states of central potentials