Finding the lowest eigenvalue of a Hamiltonian Using Variation Quantum Eigensolver (VQE)

Definitions

Let's start by defining some definitions that we will use later in this document:

- **Hamiltonian**: A Hamiltonian is an operator that describes the sum of kinetic and potential energies in a given system. Here is an expression of a the Hamiltonian:

$\hat{H} = \hat{T} + \hat{V}$

where $\hat{T}$ represent the operator of the kinetic energy and $\hat{V}$ represent the operator of the potential energy.

- **Eigenstate** An Eigenstate represent a possible energy state of the system. For example, if we have a Hydrogen atom, a possible state is to have the electron in the first orbital, and because this is the lowest energy state this atom can have, we call this state the "ground state". we can represent anystate using the bra-ket notation $|\psi\rangle$ which can be any possible energy state of the system. what make Eignestates different from other states is that it we can represent any state using combination of the Eignenstates.

- **Eigenvalue** Each Eigenvalue represents the quantity or the weight of contribution the corresponding Eigenstate to the current state of the system. λ represents the Eigenvalue.

Problem Statement

In this task we are given a Hamiltonian of a system and we are asked to find the ground state of that system

Filter tests and results

Identity filter

Figure 1: *Left:* My result was spectacular. *Right:* Curious.

Generation of Hybrid Image

In this part we are trying to add a low-pass filtered image and then add it to a high-pass filtered image of the same shape. This was implemented using 2 ways of filtering the first used the spatial convolution of the filter and the latter was done in frequency domain. The good thing of the frequency domain thing is that the convolution becomes a normal multiplication.

The kernel used is a Gaussian kernel with size 15x15 and $\sigma = 10$. This is how the low-pass filtered image was obtained and the high-filtered images was obtained by subtracting the low frequencies from the original image and this kept only high frequencies.

Here is a code snippet of the function implemented 2 times one with spatial convolution and the second using FFT-based convolution

```
1
2   def gen_hybrid_image(image1, image2, cutoff_frequency):
3
4   assert image1.shape == image2.shape
5
6   ksize = 15
7   sigma = cutoff_frequency
8
9
10  # Here I do outer product of 2 gaussian vectors to get 2D kernel
11  x = np.arange(-ksize//2, ksize//2+1)
12  gx = np.exp(-(x)**2/(2*sigma**2))
13  g = np.outer(gx, gx)
14  g /= np.sum(g)
15  kernel = g
16
17
18
19  # Your code here:
20  # looping over the channels of the image to apply the gaussian
      kernel
21  low_frequencies = np.zeros(image1.shape, dtype=np.float32)
22
23  for i in range(image1.shape[2]):
24  low_frequencies[:,:,i] = correlate2d(image1[:,:,i], kernel, 'same')
      # Replace with your implementation
25
26  # (2) Remove the low frequencies from image2. The easiest way to do
      this is to
27  #     subtract a blurred version of image2 from the original version
       of image2.
28  #     This will give you an image centered at zero with negative
      values.
29
30  low_frequencies2 = np.zeros(image2.shape, dtype=np.float32)
31
32  for i in range(image1.shape[2]):
33  low_frequencies2[:,:,i] = correlate2d(image2[:,:,i], kernel,'same')
34
35
```

```
36
37
38
39  high_frequencies = image2 - low_frequencies2 # Replace with your
      implementation
40
41
42  # print(np.sum(high_frequencies<0))
43
44  # (3) Combine the high frequencies and low frequencies
45
46  hybrid_image = low_frequencies/2 + high_frequencies/2 # Replace with
       your implementation
47
48  high_frequencies = np.clip(high_frequencies,-1.0,1.0)
49  hybrid_image = np.clip(hybrid_image,0,1)
50
51  # np.clip(low_frequencies,0,1)
52  # np.clip(high_frequencies,0,1)
53  # np.clip(hybrid_image,0,1)
54  return low_frequencies, high_frequencies, hybrid_image
55
56
57
```

```
def gen_hybrid_image_fft(image1, image2, cutoff_frequency):
assert image1.shape == image2.shape

ksize = 15
sigma = cutoff_frequency

x = np.arange(-ksize//2, ksize//2+1)
gx = np.exp(-(x)**2/(2*sigma**2))
g = np.outer(gx, gx)
g /= np.sum(g)
kernel = g


# Applyingthe fft convolution on each channel
low_freqs = np.zeros(image1.shape)
for i in range(image1.shape[2]):
low_freqs[:,:,i] = fft_convolve(image1[:,:,i], kernel)



low_freqs2 = np.zeros(image2.shape)
for i in range(image1.shape[2]):
low_freqs2[:,:,i] = fft_convolve(image2[:,:,i], kernel)


# getting only high freqs of image2
high_freqs = image2 - low_freqs2


# combining the low freqs and high freqs
hybrid_image = low_freqs/2 + high_freqs/2 # Replace with your
  implementation

high_freqs = np.clip(high_freqs,-1.0,0.5)
hybrid_image = np.clip(hybrid_image,0,1)

return low_freqs, high_freqs, hybrid_image
```

1 Results

The code worked perfectly in both spatial and FFT-based convolution.

Here are the 2 images I used:

low Frequencies