C: Python Implementation

Authors: Ethan Rajkumar

Last Modified: Sunday April 21st, 2024 by Ethan Rajkumar

In [ ]:
```
#importing the necessary libraries
import numpy as np
from lib import *
import pandas as pd
import pennylane as qml
from lib.cr2dataset import hartree, get_pot_cr2
from lib.cr2dataset import cr2_params
from lib.qpe import QPE
import jax
from jax import random
from lib.phase_estimation import *
import phayes
import jax.numpy as jnp
import random as r
import seaborn as sns
import matplotlib.pyplot as plt
from scipy.interpolate import make_interp_spline
from mpl_toolkits.mplot3d import Axes3D
import numpy as np
```

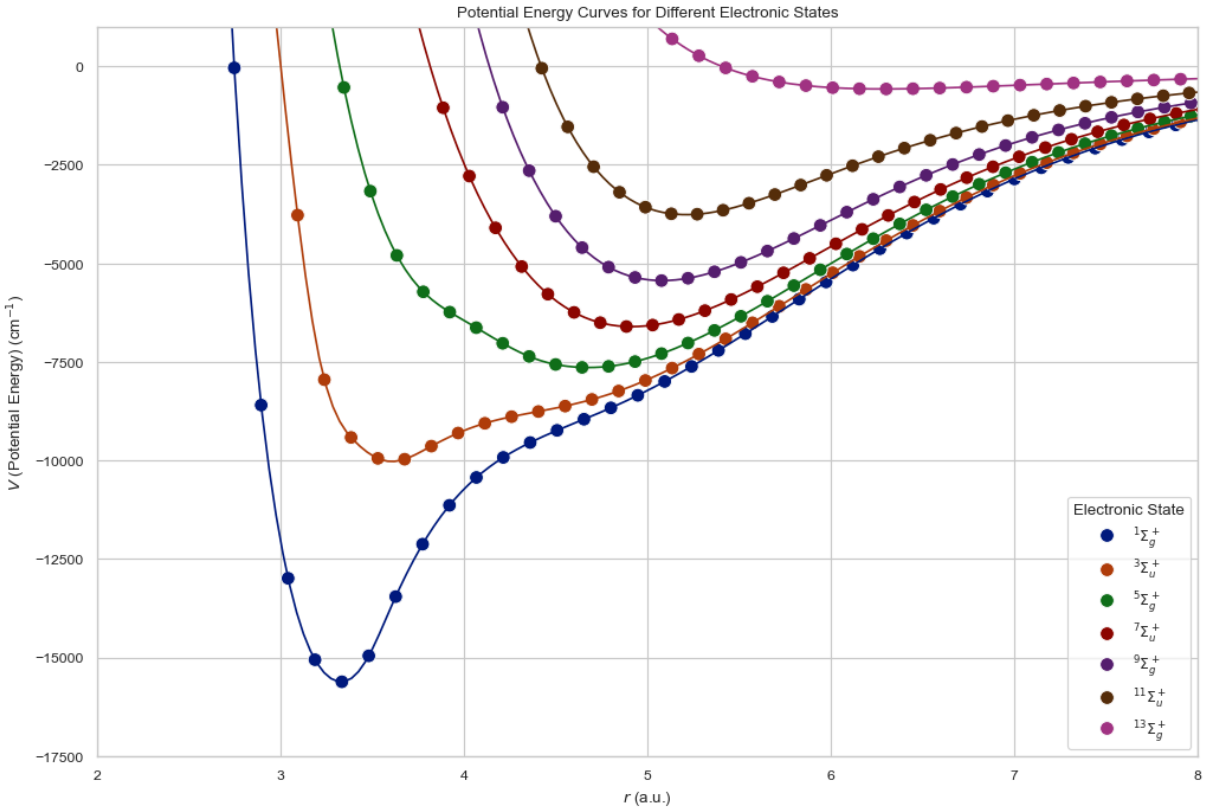
```
/var/folders/k9/jxn4k2_s1v50xzyqggp1kft40000gn/T/ipykernel_85851/2955796516.
py:4: DeprecationWarning:
Pyarrow will become a required dependency of pandas in the next major releas
e of pandas (pandas 3.0),
(to allow more performant data types, such as the Arrow string type, and bet
ter interoperability with other libraries)
but was not found to be installed on your system.
If this would cause problems for you,
please provide us feedback at https://github.com/pandas-dev/pandas/issues/54
466

  import pandas as pd
```

Part 1 Hamiltonian Generation

1.1 : Interpotential Curve Point Selection

In [ ]:
```
# Set the Seaborn theme and color palette
sns.set_theme(context='paper', style='whitegrid')
dark_palette = sns.color_palette("dark", n_colors=7)  # Using a darker palet

# Initialize DataFrame to store the potential data
data_32 = pd.DataFrame()
data_1000 = pd.DataFrame()
```

```
states = [1, 3, 5, 7, 9, 11, 13]
labels = ['${}^1\Sigma_g^+$', '${}^3\Sigma_u^+$', '${}^5\Sigma_g^+$', '${}^7
          '${}^9\Sigma_g^+$', '${}^{11}\Sigma_u^+$', '${}^{13}\Sigma_g^+$']

# Fetch potential data for each state and accumulate it in the DataFrames
for i, state in enumerate(states):
    pot, lims = get_pot_cr2(state)
    rs_1000 = np.linspace(lims[0], lims[1], 1000)
    rs_32 = np.linspace(start=lims[0], stop=lims[1], num=256)
    potentials_32 = pot(rs_32) * hartree
    potentials_1000 = pot(rs_1000) * hartree
    temp_df_32 = pd.DataFrame({'r (a.u.)': rs_32, 'V (cm^-1)': potentials_32
    temp_df_1000 = pd.DataFrame({'r (a.u.)': rs_1000, 'V (cm^-1)': potential
    data_32 = pd.concat([data_32, temp_df_32], ignore_index=True)
    data_1000 = pd.concat([data_1000, temp_df_1000], ignore_index=True)


plt.figure(figsize=(12, 8))
sns.lineplot(
    data=data_1000,
    x='r (a.u.)',
    y='V (cm^-1)',
    hue='State',
    palette=dark_palette,
    legend=None
)


sns.scatterplot(
    data=data_32,
    x='r (a.u.)',
    y='V (cm^-1)',
    hue='State',
    palette=dark_palette,
    s=72,  # Adjust the size of the scatter plot markers
    edgecolor="w",
    legend='full'
)

# Configure plot aesthetics
plt.ylim((-17500, 1000))
plt.xlim((2, 8))
plt.xlabel(r'$r$ (a.u.)')
plt.ylabel(r'$V$ (Potential Energy) (cm$^{-1}$)')
plt.title('Potential Energy Curves for Different Electronic States')
plt.legend(title='Electronic State', loc='lower right')

# Display the plot
plt.show()
```



1.2: Generating the Hamiltonians for all Spin States

In [ ]:
```
mol_params =  cr2_params
list_of_spins = [1, 3, 5, 7, 9, 11, 13]
list_of_params32 = [[2.8, 4], [3.2, 4.5], [4, 6.5], [4.3, 6.8], [4.5, 7], [4

hamiltonians = create_hamiltonians(list_of_spins, list_of_params32, mol_para

for i, spin in enumerate(list_of_spins):
    params = list_of_params32[i]
    variable_name = f"h_dvr_{spin}"
    decomposed_hamiltonian = qml.pauli_decompose(hamiltonians[(spin, tuple(p
    globals()[variable_name] = decomposed_hamiltonian
```

1.3 Conversion of the hamiltonians into a unitary matrix

In [ ]:
```
wire_order = [0, 1, 2, 3, 4]
n = 5
for spin in list_of_spins:
    variable_name = f"ufss{spin}"
    hamiltonian = globals()[f"h_dvr_{spin}"]
    unitary = convert_to_unitary(hamiltonian, n, wire_order)
    globals()[variable_name] = unitary
```

1.4 Find the true phases of the unitary matrices

In [ ]:
```
for spin in list_of_spins:
    unitary_matrix = globals()[f"ufss{spin}"]
    phase = find_true_phases(unitary_matrix)
    phase = phase/np.linalg.norm(phase)
    phase = np.abs(phase/(2*np.pi))
    globals()[f"phase{spin}"] = phase
```

In [ ]:
```
for spin in list_of_spins:
    phase = globals()[f"phase{spin}"]
    print(phase.shape)
    print(f"True phases for spin state {spin}: {phase}")
```

```
(32,)
True phases for spin state 1: [0.04096298 0.04450851 0.03373529 0.03641698
0.04883094 0.04707422
 0.04618917 0.04335729 0.04236002 0.04286075 0.03153212 0.02933257
 0.02824815 0.02153788 0.02175125 0.0180878  0.01859339 0.0256471
 0.0104862  0.00672655 0.00572107 0.00376777 0.00121464 0.00027203
 0.02241888 0.02029003 0.01805162 0.01533049 0.01363865 0.00616677
 0.00840116 0.00825458]
(32,)
True phases for spin state 3: [0.02901343 0.02373451 0.03744477 0.04043268
0.0427488  0.04435596
 0.04636712 0.04968681 0.04959865 0.04161587 0.04204605 0.03526964
 0.03264829 0.01655096 0.01354258 0.01365763 0.01005915 0.02640197
 0.00565433 0.00484661 0.00216334 0.02348992 0.02325409 0.02170404
 0.02199635 0.01558653 0.01279555 0.00834406 0.0092766  0.00902679
 0.00247604 0.00232195]
(32,)
True phases for spin state 5: [0.01227631 0.00621999 0.00261552 0.00198135
0.00339383 0.00250196
 0.00124663 0.00148712 0.02469586 0.02625868 0.02744886 0.01272442
 0.01362003 0.01481766 0.0317335  0.02972174 0.02082199 0.02099297
 0.02160365 0.02358665 0.02330392 0.03152144 0.03432793 0.03763654
 0.03866702 0.03931646 0.0414154  0.0411701  0.04617502 0.04629184
 0.04383206 0.04384084]
(32,)
True phases for spin state 7: [0.03247615 0.04127042 0.04582127 0.04682656
0.04935615 0.04803262
 0.04830488 0.04015521 0.03836336 0.03455383 0.02986265 0.02907656
 0.02588974 0.02264514 0.01966771 0.02723006 0.02580416 0.02644467
 0.01410045 0.01252247 0.00749492 0.00653779 0.00323278 0.00115246
 0.01668291 0.01541079 0.01466869 0.00436287 0.00975986 0.00909482
 0.00761252 0.0077013 ]
(32,)
True phases for spin state 9: [0.03644479 0.03410099 0.03235278 0.02972791
0.02487595 0.02248715
 0.02213779 0.01563024 0.04440149 0.04709256 0.04630769 0.04367538
 0.04300372 0.04214087 0.03814891 0.03733337 0.03145188 0.03176127
 0.03271724 0.01129369 0.0081831  0.01781276 0.01595272 0.01556459
 0.00298519 0.00221959 0.00064878 0.00018968 0.00709408 0.00770577
 0.00807828 0.00825916]
(32,)
True phases for spin state 11: [0.02388759 0.02166877 0.02048656 0.01556992
0.01169694 0.00879841
 0.00615253 0.00500117 0.00131725 0.00297208 0.00586808 0.00400173
 0.00446676 0.01710126 0.02665297 0.02037801 0.02125365 0.02352124
 0.02249257 0.03355426 0.03469166 0.03594763 0.04107166 0.04480059
 0.04601715 0.04757934 0.04908886 0.03792824 0.0332597  0.0341589
 0.03558577 0.035445  ]
(32,)
True phases for spin state 13: [0.02667305 0.02834227 0.03292579 0.03407857
0.03630076 0.03544317
 0.04465175 0.04655567 0.04831635 0.01516329 0.01289853 0.01196757
 0.00946927 0.04294896 0.03566375 0.03503588 0.03888334 0.03964266
 0.03955065 0.02637651 0.02455309 0.02127476 0.02116977 0.01578449
 0.01294445 0.00698745 0.01036515 0.00980925 0.00114507 0.00114902
 0.00204674 0.00204323]
```

Part 2 Quantum Circuit Generation for Phase Estimation in the Fourier Basis

2.1 Obtaining Prior Distribution and Setting Up for Bayesian Estimation

In [ ]:
```
unitaries = [ufss1, ufss3, ufss5, ufss7, ufss9, ufss11, ufss13]
J_max = 2000
num_experiments = 100
c = np.random.random_integers(low=7000, high=8000, size=None)
J_values = [c for i in range(num_experiments)]
n_shots_per_experiment = 1/(1-(1/(2**5)))
ks = []
betas = []
```

```
/var/folders/k9/jxn4k2_s1v50xzyqggp1kft40000gn/T/ipykernel_85851/2573652648.
py:4: DeprecationWarning: This function is deprecated. Please call randint(7
000, 8000 + 1) instead
  c = np.random.random_integers(low=7000, high=8000, size=None)
```

2.3 Obtaining the Bayesian Estimation

In [ ]:
```
results = []
for i in range(len(unitaries)):
    U = unitaries[i]
    unitary_results = []
    for l in range(len(U)):
        # Generate a new seed for each l, ensuring each unitary eigenvalue h
        eigenvalue_results = []
        phase_estimates = []
        true_eigvals, true_eigvecs = jnp.linalg.eig(U)
        phi_vec = true_eigvecs[:, l]
        n_qubits_U = int(np.log2(U.shape[0]))
        for j in range(num_experiments):
            random_keys = random.split(random.PRNGKey(0), num_experiments)
            prior = initial_fourier_state(J_values[j])
            k1= np.random.random_integers(low=20, high=8000, size=None)
            beta, k, new_state = phase_estimation_iteration(prior, k1, U, ph
            fourier_basis_probs = new_state/np.linalg.norm(new_state)
            # print(fourier_basis_probs)
            phase = np.log(np.abs(np.arccos(fourier_basis_probs) - k - beta)
            nphase = np.mean(phase/ (2*np.pi))
            # print(f"Unitary {i+1}, Eigenvalue {l}: Experiment {j + 1} of {
            average_phase = np.mean(nphase)
            phase_estimates.append(average_phase)
            eigenvalue_results.append((k, beta, new_state, fourier_basis_prc
        unitary_results.append(eigenvalue_results)
    results.append(unitary_results)
```

```
/var/folders/k9/jxn4k2_s1v50xzyqggp1kft40000gn/T/ipykernel_85851/1510552935.
py:15: DeprecationWarning: This function is deprecated. Please call randint
(20, 8000 + 1) instead
  k1= np.random.random_integers(low=20, high=8000, size=None)
/var/folders/k9/jxn4k2_s1v50xzyqggp1kft40000gn/T/ipykernel_85851/1510552935.
py:15: DeprecationWarning: This function is deprecated. Please call randint
(20, 8000 + 1) instead
  k1= np.random.random_integers(low=20, high=8000, size=None)
```

Part 2.4 Obtaining Synthetic Unitary Phase Matrices

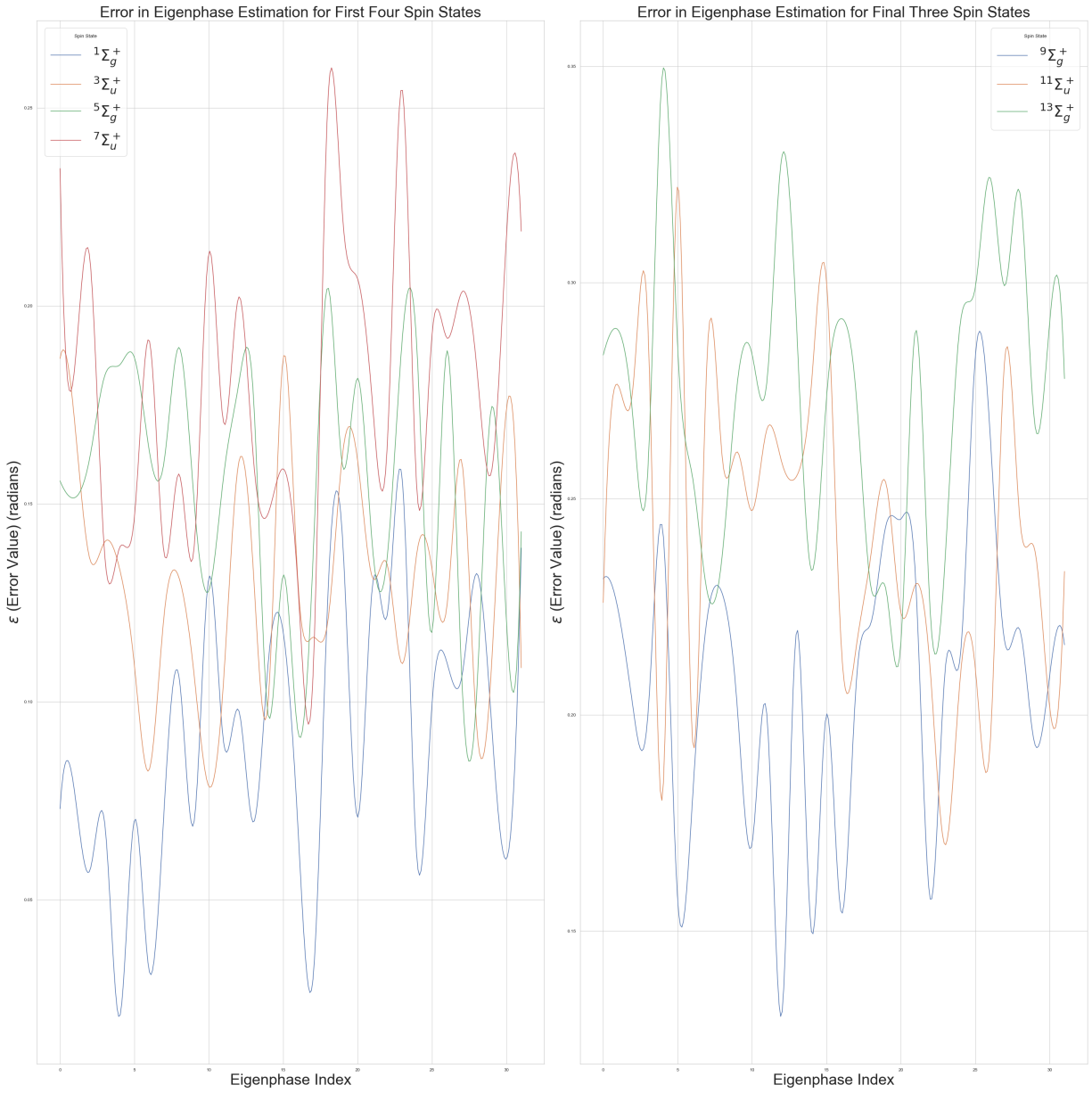
In [ ]:
```
num_unitaries = 7
num_eigenvalues = 32
average_phases = np.zeros((num_unitaries, num_eigenvalues))
for i in range(num_unitaries):
    for l in range(num_eigenvalues):
        all_phases = [experiment_data[-1] for experiment_data in results[i][
        average_phases[i, l] = np.mean(all_phases)
print(average_phases)
```

```
[[0.11395961 0.12111673 0.09104102 0.10592369 0.06922828 0.11720211
  0.07807143 0.11638102 0.14868714 0.11226954 0.16276877 0.11888868
  0.12617953 0.09108213 0.13188769 0.1357094  0.08626869 0.0554593
  0.137017   0.14726561 0.0765395  0.13311245 0.12331188 0.15654524
  0.08213437 0.11966629 0.12801765 0.12155099 0.14604159 0.10084592
  0.06871746 0.1470913 ]
 [0.18573934 0.16375986 0.14356634 0.15079993 0.14638789 0.12351263
  0.0992226  0.14173636 0.15090252 0.11885538 0.09070852 0.11103068
  0.16285193 0.11917759 0.08407069 0.17046222 0.10978495 0.11269349
  0.09596249 0.1377196  0.13314468 0.12483709 0.1278009  0.10131795
  0.13002306 0.11919911 0.10682631 0.13901824 0.07309473 0.08805895
  0.14624996 0.08091483]
 [0.10806192 0.09779248 0.10425775 0.12427528 0.128428   0.12945777
  0.10461319 0.10227016 0.15420115 0.11755738 0.09549206 0.10984612
  0.13364519 0.13193598 0.06819494 0.10165607 0.0530957  0.09676424
  0.16625066 0.12299077 0.14505373 0.11065858 0.11115564 0.16919513
  0.16693483 0.09699918 0.16998944 0.08743015 0.08789979 0.16059481
  0.10449243 0.12679575]
 [0.17716509 0.13619228 0.16797274 0.09405906 0.09825454 0.10463986
  0.14958811 0.08751243 0.10587692 0.08299973 0.15311566 0.10951629
  0.13801961 0.0939096  0.07733445 0.09607178 0.06458823 0.04262562
  0.17555495 0.14465526 0.1243597  0.09188099 0.07839951 0.16588652
  0.07927408 0.1180156  0.11658533 0.11781942 0.1045507  0.07716891
  0.13486521 0.13656639]
 [0.14793463 0.13836208 0.11404443 0.10957082 0.14761616 0.05982174
  0.08269269 0.11839959 0.15246695 0.12924878 0.09642825 0.12358081
  0.05365925 0.14161673 0.06788171 0.11734984 0.06567408 0.11884755
  0.13404033 0.13510743 0.13346457 0.12877452 0.0530991  0.10668735
  0.09660047 0.16468582 0.14577103 0.096764   0.10673943 0.08101795
  0.09704457 0.10447072]
 [0.09987066 0.14792761 0.14617535 0.15611042 0.04339428 0.18131077
  0.05039941 0.13925551 0.11078427 0.11381692 0.10308405 0.12007491
  0.11288551 0.12232678 0.15336508 0.17098401 0.08713941 0.08911204
  0.10972044 0.13716879 0.10840355 0.11615657 0.10048097 0.06469432
  0.10481496 0.10861614 0.09201349 0.17088619 0.12856022 0.1223587
  0.08747496 0.1186164 ]
 [0.1299143  0.13737421 0.12151149 0.11061744 0.20528017 0.14068857
  0.11967447 0.09478872 0.10526848 0.1115399  0.11636737 0.10901012
  0.15848495 0.15273784 0.0890598  0.12799093 0.15053999 0.13369177
  0.08890244 0.07540163 0.06099636 0.13024588 0.0637094  0.07304776
  0.12360635 0.12570602 0.15468851 0.12906601 0.14191705 0.08853123
  0.11261807 0.09982844]]
```

2.4 Graphing the Estimated Phase vs the True Phases

In [ ]:
```
true_phases = [phase1, phase3, phase5, phase7, phase9, phase11, phase13]

error_values = []
for syn_phase, true_phase in zip(average_phases, true_phases):
    error = np.abs(syn_phase - true_phase)
    error_values.append(error)

phase_indices = np.arange(len(error_values[0]))

labels = ['${}^1\Sigma_g^+$', '${}^3\Sigma_u^+$', '${}^5\Sigma_g^+$', '${}^7
```

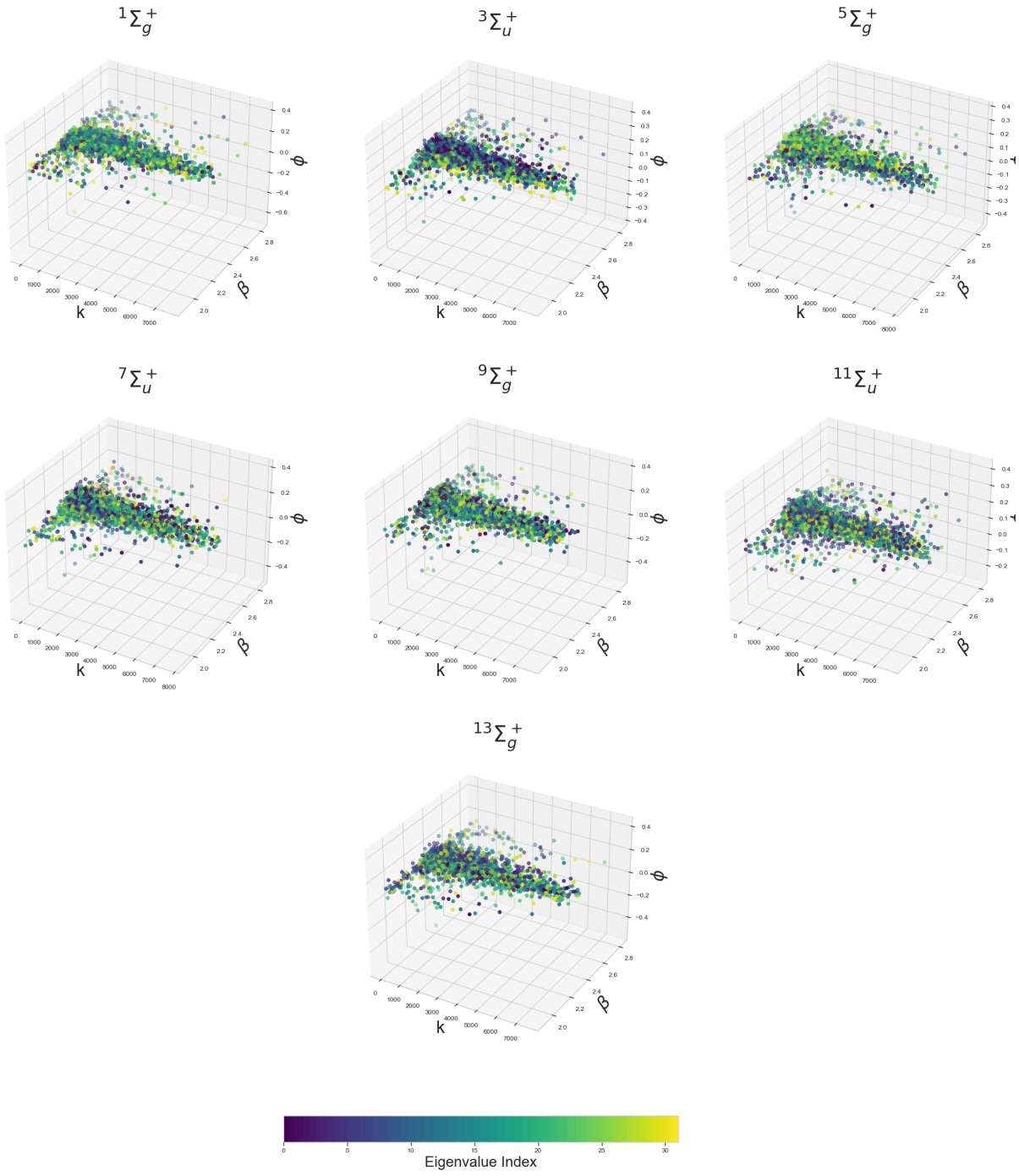
```
          '${}^9\Sigma_g^+$', '${}^{11}\Sigma_u^+$', '${}^{13}\Sigma_g^+$']

offset_increment = 0.03
n = 30
fig, axs = plt.subplots(1, 2, figsize=(n, n))
for index, errors in enumerate(error_values[:4]):
    spline = make_interp_spline(phase_indices, errors, k=3)
    smooth_indices = np.linspace(phase_indices.min(), phase_indices.max(), 3
    smooth_errors = spline(smooth_indices)
    offset_errors = smooth_errors + offset_increment * index
    sns.lineplot(x=smooth_indices, y=offset_errors, label=f'{labels[index]}'

axs[0].set_title('Error in Eigenphase Estimation for First Four Spin States'
axs[0].set_xlabel('Eigenphase Index', fontsize=n)
axs[0].set_ylabel(r'$\epsilon$ (Error Value) (radians)', fontsize=n)
axs[0].legend(title='Spin State', fontsize=n)

for index, errors in enumerate(error_values[4:7]):
    spline = make_interp_spline(phase_indices, errors, k=3)
    smooth_indices = np.linspace(phase_indices.min(), phase_indices.max(), 3
    smooth_errors = spline(smooth_indices)
    offset_errors = smooth_errors + offset_increment * (index + 4)
    sns.lineplot(x=smooth_indices, y=offset_errors, label=f'{labels[index +

axs[1].set_title('Error in Eigenphase Estimation for Final Three Spin States
axs[1].set_xlabel('Eigenphase Index', fontsize=n)
axs[1].set_ylabel(r'$\epsilon$ (Error Value) (radians)', fontsize=n)
axs[1].legend(title='Spin State', fontsize=n)

plt.tight_layout()
plt.show()
```



In [ ]:

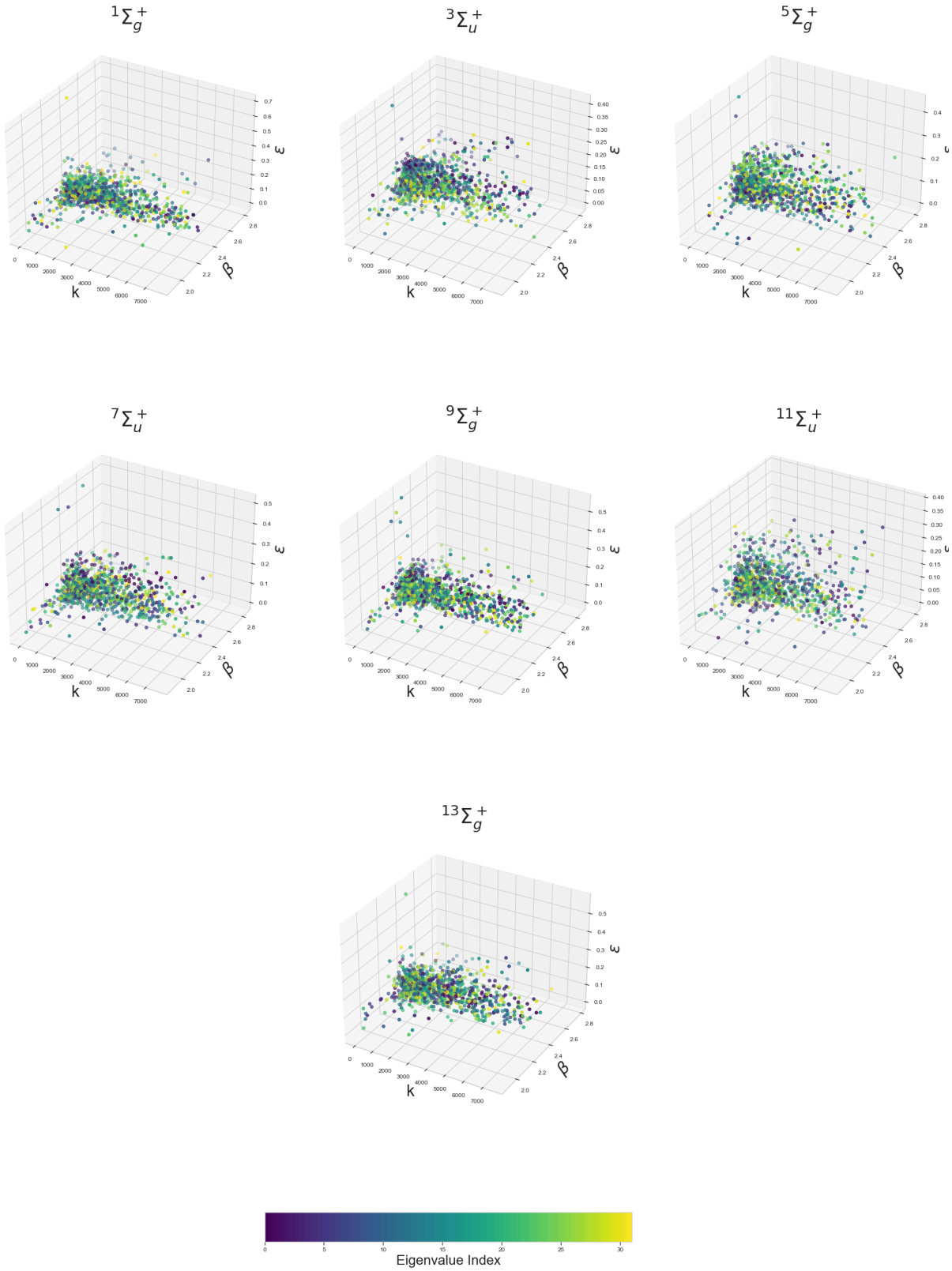
In [ ]:
```
fig = plt.figure(figsize=(25, 25))
labels = ['${}^1\Sigma_g^+$', '${}^3\Sigma_u^+$', '${}^5\Sigma_g^+$', '${}^7
          '${}^9\Sigma_g^+$', '${}^{11}\Sigma_u^+$', '${}^{13}\Sigma_g^+$']

for i in range(7):
    if i < 6:
        ax = fig.add_subplot(3, 3, i + 1, projection='3d')
    else:
        ax = fig.add_subplot(3, 3, 8, projection='3d')

    all_ks = []
    all_betas = []
    all_phases = []
    all_eigenvalue_indices = []

    unitary_results = results[i]
    for j, eigenvalue_results in enumerate(unitary_results):
```

```
        ks = []
        betas = []
        average_phases = []

        for experiment_result in eigenvalue_results:
            k, beta, _, _, average_phase = experiment_result
            ks.append(k)
            betas.append(beta)
            average_phases.append(average_phase)

        all_ks.extend(ks)
        all_betas.extend(betas)
        all_phases.extend(average_phases)
        all_eigenvalue_indices.extend([j] * len(ks))

    all_ks = np.array(all_ks)
    all_betas = np.array(all_betas)
    all_phases = np.array(all_phases)
    all_eigenvalue_indices = np.array(all_eigenvalue_indices)

    scatter = ax.scatter(all_ks, all_betas, all_phases, c=all_eigenvalue_inc
    ax.set_xlabel('k',  fontsize=24)
    ax.set_ylabel(r'$\beta$', fontsize=24)
    ax.set_zlabel(r'$\phi$', fontsize=24)
    ax.set_title(labels[i], fontsize=30)

cbar_ax = fig.add_axes([0.35, 0.04, 0.3, 0.02])
cbar = fig.colorbar(scatter, cax=cbar_ax, orientation='horizontal')
cbar.set_label('Eigenvalue Index', fontsize=20)


plt.show()
```



In [ ]:
```
# Assuming 'results' and 'true_phases' are defined somewhere before this coc

fig = plt.figure(figsize=(25, 30))
labels = ['${}^1\Sigma_g^+$', '${}^3\Sigma_u^+$', '${}^5\Sigma_g^+$', '${}^7
          '${}^9\Sigma_g^+$', '${}^{11}\Sigma_u^+$', '${}^{13}\Sigma_g^+$']

# Assuming 'results' and 'true_phases' are defined somewhere before this coc

for i in range(7):
    if i < 6:
        ax = fig.add_subplot(3, 3, i + 1, projection='3d')
    else:
        ax = fig.add_subplot(3, 3, 8, projection='3d')
```

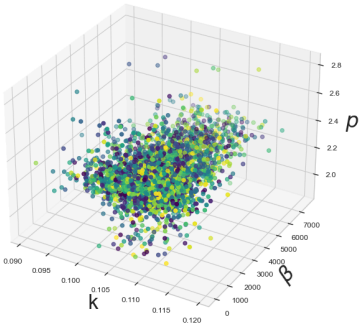
```
    all_ks = []
    all_betas = []
    all_errors = []
    all_eigenvalue_indices = []

    unitary_results = results[i]
    true_phase = true_phases[i]

    for j, eigenvalue_results in enumerate(unitary_results):
        ks = []
        betas = []
        errors = []
        for experiment_result, true_phase_value in zip(eigenvalue_results, t
            k, beta, _, _, average_phase = experiment_result
            ks.append(k)
            betas.append(beta)
            errors.append(np.abs(average_phase - true_phase_value))

        all_ks.extend(ks)
        all_betas.extend(betas)
        all_errors.extend(errors)
        all_eigenvalue_indices.extend([j] * len(ks))

    all_ks = np.array(all_ks)
    all_betas = np.array(all_betas)
    all_errors = np.array(all_errors)
    all_eigenvalue_indices = np.array(all_eigenvalue_indices)

    scatter = ax.scatter(all_ks, all_betas, all_errors, c=all_eigenvalue_inc
    ax.set_xlabel('k', fontsize=24)
    ax.set_ylabel(r'$\beta$', fontsize=24)
    ax.set_zlabel(r'$\epsilon$', fontsize=24)
    ax.set_title(labels[i], fontsize=30)

# Add color bar
cbar_ax = fig.add_axes([0.35, 0.04, 0.3, 0.02])
cbar = fig.colorbar(scatter, cax=cbar_ax, orientation='horizontal')
cbar.set_label('Eigenvalue Index', fontsize=20)

plt.show()
```



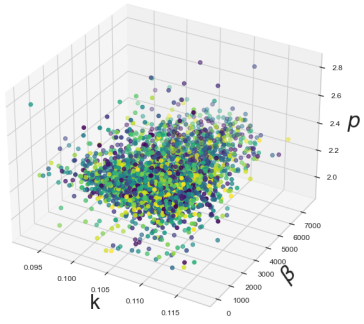
In [ ]:
```
fig = plt.figure(figsize=(25, 25))
labels = ['${}^1\Sigma_g^+$', '${}^3\Sigma_u^+$', '${}^5\Sigma_g^+$', '${}^7
          '${}^9\Sigma_g^+$', '${}^{11}\Sigma_u^+$', '${}^{13}\Sigma_g^+$']

for i in range(7):
    if i < 6:
        ax = fig.add_subplot(3, 3, i + 1, projection='3d')
    else:
```

```
        ax = fig.add_subplot(3, 3, 8, projection='3d')

    all_ks = []
    all_betas = []
    all_phases = []
    all_eigenvalue_indices = []

    unitary_results = results[i]
    for j, eigenvalue_results in enumerate(unitary_results):
        ks = []
        betas = []
        average_phases = []

        for experiment_result in eigenvalue_results:
            k, beta, _, fourier_basis_probs, _, = experiment_result
            ks.append(k)
            betas.append(beta)
            average_phases.append(np.mean(fourier_basis_probs))

        all_ks.extend(ks)
        all_betas.extend(betas)
        all_phases.extend(average_phases)
        all_eigenvalue_indices.extend([j] * len(ks))

    all_ks = np.array(all_ks)
    all_betas = np.array(all_betas)
    all_phases = np.array(all_phases)
    all_eigenvalue_indices = np.array(all_eigenvalue_indices)

    scatter = ax.scatter(all_phases, all_ks, all_betas, c=all_eigenvalue_inc
    ax.set_xlabel('k',  fontsize=24)
    ax.set_ylabel(r'$\beta$', fontsize=24)
    ax.set_zlabel(r'$p$', fontsize=24)
    ax.set_title(labels[i], fontsize=30)

cbar_ax = fig.add_axes([0.35, 0.04, 0.3, 0.02])
cbar = fig.colorbar(scatter, cax=cbar_ax, orientation='horizontal')
cbar.set_label('Eigenvalue Index', fontsize=20)


plt.show()
```

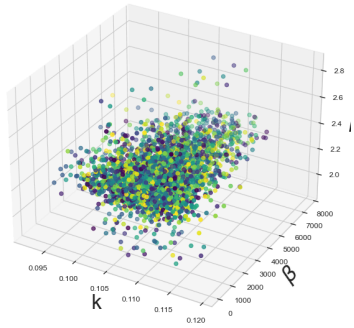
$^1\Sigma_g^+$



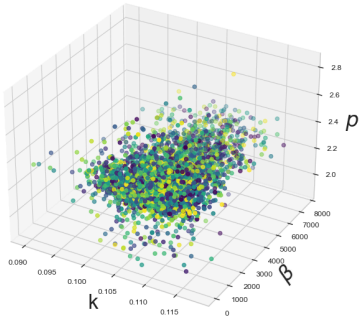
$^3\Sigma_u^+$



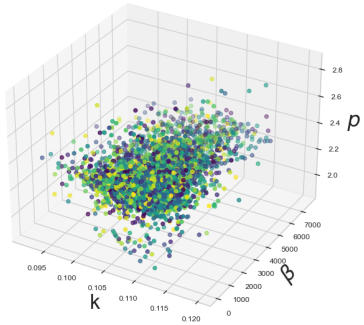
$^5\Sigma_g^+$



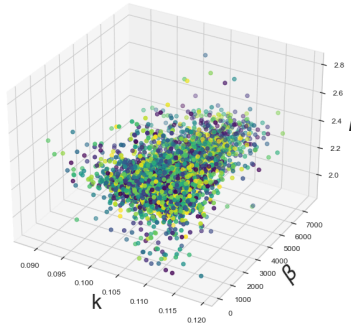
$^7\Sigma_u^+$



$^9\Sigma_g^+$



$^{11}\Sigma_u^+$



$^{13}\Sigma_g^+$