bayesian_learning_applied

February 6, 2023

[ ]:
```
from gibbs.dataclass import GibbsResult
import matplotlib.pyplot as plt
import numpy as np
from gibbs.learning.bayesian_learning import BayesianLearning
from gibbs.learning.constraint_matrix import ConstraintMatrixFactory
from qiskit.quantum_info import Statevector
from gibbs.learning.klocal_pauli_basis import KLocalPauliBasis
from scipy.linalg import block_diag
from scipy.sparse import bmat
from qiskit.quantum_info import state_fidelity
from gibbs.utils import number_of_elements, simple_purify_hamiltonian,␣
 ↪spectral_dec
# plt.rcParams['text.usetex'] = True

%load_ext autoreload
%autoreload 2
```

```
The autoreload extension is already loaded. To reload it, use:
  %reload_ext autoreload
```

[ ]:
```
import os
gibbs_result_list = []
folder_path = "saved_simulations/turbo/allcontrols4heisenberg/"
for file in os.listdir(folder_path):
    if file.endswith(".npy"):
        path = os.path.join(folder_path, file)
        gibbs_result_list.append(GibbsResult.load(path))

gibbs_result_list = gibbs_result_list[:3]

c_original_prior = gibbs_result_list[0].coriginal
states = [r.state for r in gibbs_result_list]
control_fields = [r.coriginal - c_original_prior for r in gibbs_result_list]
```

[ ]:
```
#Faulty prep with control field - Original faulty prep - (Original H - Control␣
 ↪H)
control_error_vectors = [(r.cfaulties[-1]-gibbs_result_list[0].cfaulties[-1]) -␣
 ↪(r.coriginal - gibbs_result_list[0].coriginal) for r in gibbs_result_list]
```

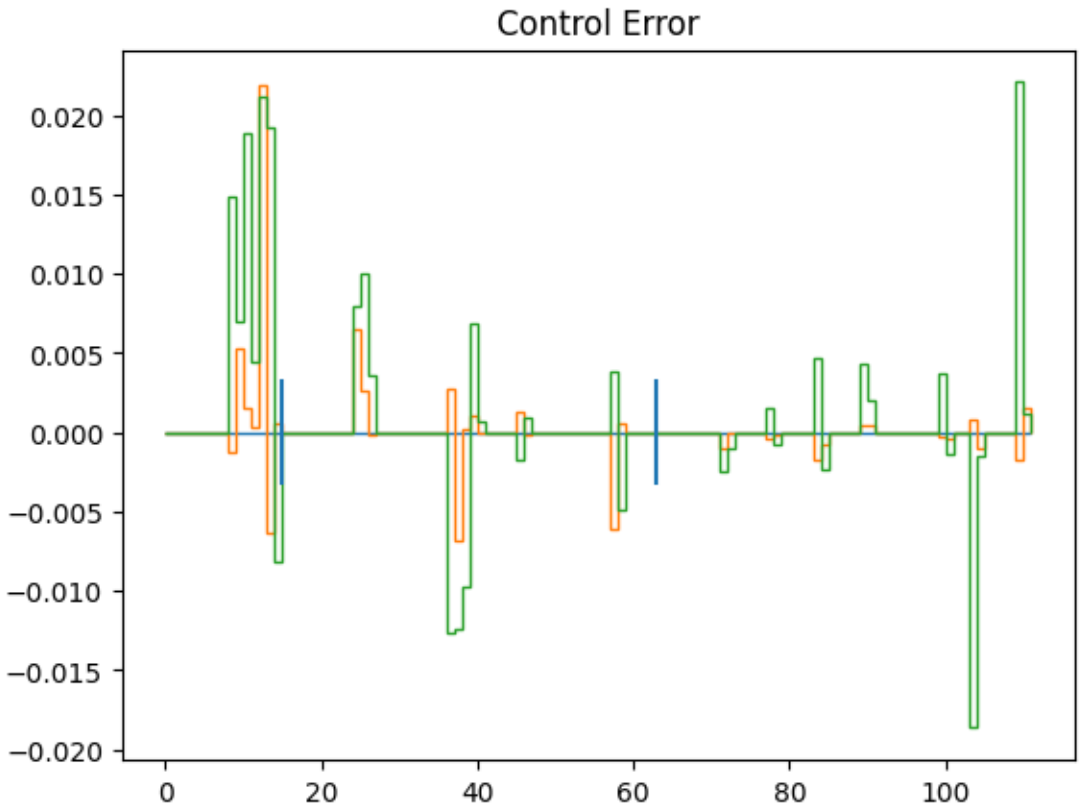
```
for err_vec in control_error_vectors:
    plt.stairs(err_vec)


control_noise = np.std(np.concatenate(control_error_vectors))
prep_noise = np.std(gibbs_result_list[0].cfaulties[-1]-c_original_prior)

plt.vlines([KLocalPauliBasis(4,2).size,KLocalPauliBasis(4,3).
  ↪size,],[-control_noise]*2,[control_noise]*2)
plt.title("Control Error")

print(f"The control error is: {control_noise:.2e}")
print(f"The preparation error is:{prep_noise:.2e}")
```

```
The control error is: 3.38e-03
The preparation error is:5.62e-02
```



[ ]:
```
shots = 1e10
initial_arguments = {
    "states":states,
    "control_fields": control_fields,
```

```
    "constraint_matrix_factory": ConstraintMatrixFactory(4,3,3),
    "prior_mean": c_original_prior,
    "prior_covariance": (prep_noise,control_noise),
    "sampling_std": 1/np.sqrt(shots),
    "shots": shots
    }
bl = BayesianLearning(**initial_arguments)
```

[ ]:
```
bl.constraint_matrix(0)
```

[ ]:
```
<111x111 sparse matrix of type '<class 'numpy.complex128'>'
        with 6072 stored elements in Compressed Sparse Row format>
```

[ ]:
```
update = bl.update_mean()
cov = bl.update_cov(update)
```

```
(333,) (333, 333)
The time it takes for minimize is: 380.8295521736145 for the rest:
0.03212690353393555
The cost function ends up with a value of:7.147792825871283, it started with a
value of 10.390187574337153
```

[ ]:
```
fig,ax = plt.subplots(1,3,figsize=(15,5))
ax[0].stairs(gibbs_result_list[0].cfaulties[-1],label="preparation")
ax[0].stairs(c_original_prior,label = "prior")
ax[0].stairs(update[:c_original_prior.size],label="posterior")
ax[0].legend(loc = "lower right")
width = 0.9
ax[1].bar(np.arange(bl.size),np.abs(gibbs_result_list[0].
 ↪cfaulties[-1]-c_original_prior),width,label="prior error",lw=2,fill=True)
ax[1].bar(np.arange(bl.size),np.abs(gibbs_result_list[0].cfaulties[-1]-update[:
 ↪c_original_prior.size]),0.6*width,label="posterior error",fill=True)
ax[1].stairs(cov.diagonal()[:c_original_prior.size],np.arange(bl.size+1)-1/
 ↪2,label="std",color="red")
ax[1].legend()
ax[2].stairs(update,label="posterior")
ax[2].stairs(cov.diagonal(),label="posterior std")
ax[2].vlines([bl.size],[-1],[0.4],color="black",linestyles="dotted")
ax[2].legend()

basisH = bl.constraint_matrix_factory.learning_basis
preparationH = gibbs_result_list[0].cfaulties[-1]
preparation_state = Statevector(gibbs_result_list[0].state)
print(f"We start with a hamiltonian error of:{np.linalg.
 ↪norm(c_original_prior-preparationH)} and end up with {np.linalg.norm(update[:
 ↪c_original_prior.size]-preparationH)} ")
```

```
print(f"The prior fidelity is: {state_fidelity(simple_purify_hamiltonian(basisH.
  ↪vector_to_pauli_op(c_original_prior)),preparation_state)} and the posterior␣
  ↪fidelity is: {state_fidelity(simple_purify_hamiltonian(basisH.
  ↪vector_to_pauli_op(update[:c_original_prior.size])),preparation_state)}")
```

```
We start with a hamiltonian error of:0.5993685831967875 and end up with
0.014040595341527635
The prior fidelity is: 0.9600228826006943 and the posterior fidelity is:
0.9945386618078459
```