

Join GitHub today

GitHub is home to over 40 million developers working together to host and review code, manage projects, and build software together.

[Sign up](#)

Branch: master ▾

[Find file](#)[Copy path](#)

PH413 / k_cnot.py

**TejasAvinashShetty** Add files via upload

9d4810d Nov 3, 2019

[1 contributor](#)[Raw](#) [Blame](#) [History](#)

840 lines (713 sloc) | 24.1 KB

```
1  # # Optimization of a State-to-State Transfer in a Two-Level-System
2  from krotov.shapes import flattop
3  # from krotov import Objective
4  from krotov import optimize_pulses, gate_objectives
5  from krotov.propagators import expm
6  # from krotov.functionals import chis_re
7  from krotov.info_hooks import print_table
8  from krotov.convergence import check_monotonic_error, Or, value_below
9  # from krotov.functionals import J_T_re
10 from krotov.functionals import chis_ss
11 from krotov.functionals import J_T_ss
12 from krotov.structural_conversions import pulse_onto_tlist
13
14 # from qutip.control import grape_unitary,
15 from qutip.control import plot_grape_control_fields
16 from qutip.control import grape_unitary_adaptive
17 from qutip.ui.progressbar import TextProgressBar
18 # from cw import crude_writer
19 from qutip.operators import sigmax, sigmaz
20 from qutip.operators import sigmay, identity
21 from qutip.qip.gates import cnot
22 from qutip.tensor import tensor
23
24 from qutip.states import basis
25 from qutip import ket, ket2dm
26 from numpy import linspace, array
27 # from numpy import zeros
28 from numpy import pi
29 from numpy import savez
30 # from math import sqrt
31 from os import getcwd, mkdir, chdir
32 # from sys import stdout
33 from mcf import plot_compare
34 from numpy.random import random_sample
35 from tej_plotter import tej_plotter
36 from cf2u import cf2u
37 from matplotlib import use
38 from matplotlib.pyplot import figure, axes
39 from matplotlib.pyplot import subplots
40
41 from matplotlib import rcParams
42 from matplotlib.pyplot import style
```

```

43 from matplotlib.pyplot import cla, close
44 use("TkAgg")
45 pgf_with_rc_fonts = {"pgf.texsystem": "pdflatex"}
46 rcParams.update(pgf_with_rc_fonts)
47 style.use('seaborn-whitegrid')
48
49 print("Please input a run_no ")
50 print("Only integers allowed")
51 run_no = int(input())
52 run_dir_name = 'autorun' + str(run_no)
53 mkdir(run_dir_name)
54 chdir(run_dir_name)
55 print("Current directory")
56 print(getcwd())
57 f = open('workfile.txt', 'w')
58 # This first example illustrates the basic use of the `krotov`
59 # package by solving
60 # a simple canonical optimization problem: the transfer of
61 # population in a two
62 # level system.
63
64 # ## Two-level-Hamiltonian
65
66 # We consider the Hamiltonian  $\hat{H}_0 = -\frac{\omega}{2}$ 
67 #  $\hat{\sigma}_z$ , representing
68 # a simple qubit with energy level splitting  $\omega$  in the basis
69 #  $|\ket{0}\rangle, |\ket{1}\rangle$ . The control field  $\epsilon(t)$  is assumed to
70 # couple via
71 # the Hamiltonian  $\hat{H}_1(t) = \epsilon(t) \hat{\sigma}_x$  to the
72 # qubit,
73 # i.e., the control field effectively drives transitions between both
74 # qubit states.
75
76 # In[20]:
77 T = 1 # 1 # 5
78 tlist = linspace(0, T, 500)
79 # T = 1
80 # tlist = linspace(0, T, 100)
81
82
83 def hamiltonian(omega=1.0, ampl0=0.2):
84     """Two-level-system Hamiltonian
85
86     Args:
87         omega (float): energy separation of the qubit levels
88         ampl0 (float): constant amplitude of the driving field
89     """
90
91     H_ops = [tensor(sigmaz(), identity(2)),
92              tensor(sigmay(), identity(2)),
93              tensor(sigmaz(), identity(2)),
94              tensor(identity(2), sigmaz()),
95              tensor(identity(2), sigmay()),
96              tensor(identity(2), sigmaz()),
97              tensor(sigmaz(), sigmaz()) +
98              tensor(sigmay(), sigmay()) +
99              tensor(sigmaz(), sigmaz())]
100
101     H0 = 0 * pi * (tensor(sigmaz(), identity(2)) + tensor(identity(2),
102                      sigmaz()))
103
104     # H0 = -0.5 * omega * sigmaz()
105     # H0 = 1 * pi * sigmaz()
106     # H1 = sigmaz()
107
108     def guess_control(t, args):
109         return ampl0 * flattpot(
110             t, t_start=0, t_stop=T, t_rise=0.3*T/5, func="sinsq"

```

```

111         )
112
113     H_qutip_list = [H0]
114     for h in H_ops:
115         H_qutip_list.append([h, guess_control])
116
117     # [H0, [H1, guess_control]]
118
119     return H_qutip_list
120
121
122 # H = hamiltonian(ampl0=2 * pi) # pass no change
123 # H = hamiltonian() # fail
124 # H = hamiltonian(ampl0=1.0) # fail
125 # H = hamiltonian(ampl0=2.0) # fail
126 H = hamiltonian(ampl0=3.0)
127
128 U = cnot()
129 # U = sigmax()
130 # The control field here switches on from zero at $t=0$
131 # to it's maximum amplitude
132 # 0.2 within the time period 0.3 (the switch-on shape
133 # is half the first period of
134 # a $\sin^2$ function). It switches off again in the time period 0.3
135 # before the
136 # final time $T=5$). We use a time grid with 500 time steps between 0
137 # and $T$:
138
139
140 def plot_pulse(pulse, tlist, result_string='r', rounding_limit=5):
141     '''
142     plot of the pulse
143
144
145     Inputs:
146
147     rounding_limit : int
148         The no of digits to which the file no should be
149         fixed to. Default is 5
150
151     '''
152
153     # the random number
154     t_r_no = str(round(random_sample(), rounding_limit))
155     gp_fig = figure()
156     gp_ax = axes()
157     # gp_fig, gp_ax = subplots()
158     if callable(pulse):
159         pulse = array([pulse(t, args=None) for t in tlist])
160     gp_ax.plot(tlist, pulse)
161     gp_ax.set_xlabel('time')
162     gp_ax.set_ylabel('pulse amplitude')
163     gp_ax.set(title='Plot of ' + result_string + 'pulse')
164     fig_rand_name = result_string + 'pulse' + t_r_no + '.pdf'
165     # Could also use date time
166     # fig_path_name = path + fig_name
167
168     gp_fig.savefig(fig_rand_name)
169     cla()
170     close(gp_fig)
171     # plt.show(fig)
172     return gp_fig, gp_ax
173
174
175 gp_fig, gp_ax = plot_pulse(H[1][1], tlist, result_string='guess')
176
177
178 # ## Optimization target

```

```

179
180 # The `krotov` package requires the goal of the optimization to be
181 # described by a
182 # list of `Objective` instances. In this example, there is only a single
183 # objective: the state-to-state transfer from initial state
184 #  $|\Psi_{\text{init}}\rangle =$ 
185 #  $|\text{ket}\{0\}$  to the target state  $|\Psi_{\text{tgt}}\rangle = |\text{ket}\{1\}$ ,
186 # under the dynamics
187 # of the Hamiltonian  $\text{op}\{H\}(t)$ :
188
189
190 '''
191 objectives = [
192     Objective(
193         initial_state=ket("0"), target=ket("1"), H=H
194     )
195 ]
196 '''
197 2qubit_basis_states = [tensor(ket("0"), ket("0")),
198                        tensor(ket("0"), ket("1")),
199                        tensor(ket("1"), ket("0")),
200                        tensor(ket("1"), ket("1")), ]
201
202 objectives = gate_objectives(
203     basis_states=2qubit_basis_states,
204     gate=U,
205     H=H
206 )
207 print("objectives[0]\n", objectives[0])
208 print("objectives[1]\n", objectives[1])
209 # print("objectives[2]\n", objectives[2])
210
211 print("objectives\n", objectives, file=f)
212
213
214 # In addition, we would like to maintain the property of the control
215 # field to be
216 # zero at  $t=0$  and  $t=T$ , with a smooth switch-on and switch-off.
217 # We can define
218 # an "update shape"  $S(t) \in [0, 1]$  for this purpose: Krotov's method
219 # will
220 # update the field at each point in time proportionally to  $S(t)$ ;
221 # wherever
222 #  $S(t)$  is zero, the optimization will not change the value of the
223 # control from
224 # the original guess.
225
226 def S(t):
227     """Shape function for the field update"""
228     return flattop(
229         t, t_start=0, t_stop=T, t_rise=(0.3*T)/5,
230         t_fall=(0.3*T)/5, func='sinsq'
231     )
232
233
234 # Beyond the shape, Krotov's method uses a parameter  $\lambda_a$ 
235 # for each control
236 # field that determines the overall magnitude of the respective field
237 # in each
238 # iteration (the smaller  $\lambda_a$ , the larger the update;
239 # specifically, the
240 # update is proportional to  $\frac{S(t)}{\lambda_a}$ ). Both the
241 # update-shape
242 #  $S(t)$  and the  $\lambda_a$  parameter must be passed to the
243 # optimization routine
244 # as "pulse options":
245
246 '''

```

```

247 pulse_options = {
248     H[1][1]: dict(lambda_a=5, shape=S)
249 }
250
251 pulse_options = {
252     H[1][1]: dict(lambda_a=5, update_shape=S)
253 }
254 '''
255
256 pulse_options = {
257     H[i][1]: dict(lambda_a=5, update_shape=S)
258     for i in range(1, len(H_ops) + 1, 1)
259 }
260
261
262 # ## Simulate dynamics under the guess field
263
264 # Before running the optimization procedure, we first simulate the
265 # dynamics under the
266 # guess field  $\epsilon_0(t)$ . The following solves equation
267 # of motion for the
268 # defined objective, which contains the initial state
269 #  $|\Psi_{\text{init}}\rangle$  and
270 # the Hamiltonian  $H(t)$  defining its evolution.
271 # This delegates to QuTiP's
272 # usual `mesolve` function.
273
274 # We use the projectors  $P_0 = |0\rangle\langle 0|$ 
275 # and  $P_1 = |1\rangle\langle 1|$  for calculating the population:
276
277 proj0 = ket2dm((basis(4, 0)))
278 proj1 = ket2dm(U*(basis(4, 0)))
279
280 '''
281 proj0 = ket2dm((basis(2, 0) + basis(2, 1))/(1/sqrt(2)))
282 proj1 = ket2dm((basis(2, 0) - basis(2, 1))/(1/sqrt(2)))
283
284 proj0 = ket2dm(basis(2, 0) + 1j*basis(2, 1))
285 proj1 = ket2dm(basis(2, 0) - 1j*basis(2, 1))
286
287 Python 3.7.4 (default, Aug 13 2019, 20:35:49)
288 [GCC 7.3.0] :: Anaconda, Inc. on linux
289 Type "help", "copyright", "credits" or "license" for more information.
290 >>> import qutip
291 >>> basis(0, 2)
292 Traceback (most recent call last):
293   File "<stdin>", line 1, in <module>
294 NameError: name 'basis' is not defined
295 >>> from qutip.operators import basis
296 Traceback (most recent call last):
297   File "<stdin>", line 1, in <module>
298 ImportError: cannot import name 'basis' from 'qutip.operators'
299 (/home/tejas/anaconda3/envs/krotov/lib/python3.7/site-packages/qutip/
300 operators.py)
301 >>> doc
302 Traceback (most recent call last):
303   File "<stdin>", line 1, in <module>
304 NameError: name 'doc' is not defined
305 >>> doc(qutip)
306 Traceback (most recent call last):
307   File "<stdin>", line 1, in <module>
308 NameError: name 'doc' is not defined
309 >>> from qutip.states import basis
310 >>> basis(0, 2)
311 Traceback (most recent call last):
312   File "<stdin>", line 1, in <module>
313   File
314     "/home/tejas/anaconda3/envs/krotov/lib/python3.7/site-packages/qutip/

```

```

315     states.py", line 105, in basis
316         raise ValueError("basis vector index need to be in n <= N-1")
317 ValueError: basis vector index need to be in n <= N-1
318 >>> basis(2, 0)
319 Quantum object: dims = [[2], [1]], shape = (2, 1), type = ket
320 Qobj data =
321 [[1.]
322  [0.]]
323 >>> basis(2, 0) + 1j*basis(2, 1)
324 Quantum object: dims = [[2], [1]], shape = (2, 1), type = ket
325 Qobj data =
326 [[1.+0.j]
327  [0.+1.j]]
328 >>> from qutip.operators import sigmay
329 >>> sigmay()*(basis(2, 0) + 1j*basis(2, 1))
330 Quantum object: dims = [[2], [1]], shape = (2, 1), type = ket
331 Qobj data =
332 [[1.+0.j]
333  [0.+1.j]]
334 >>>
335
336
337 proj0 = ket2dm(ket("0"))
338 proj1 = ket2dm(ket("1"))
339 '''
340 '''
341 Python 3.7.4 (default, Aug 13 2019, 20:35:49)
342 [GCC 7.3.0] :: Anaconda, Inc. on linux
343 Type "help", "copyright", "credits" or "license" for more information.
344 >>> import qutip
345 >>> proj0 = qutip.ket2dm(qutip.ket("0"))
346 >>> pr
347 Traceback (most recent call last):
348   File "<stdin>", line 1, in <module>
349 NameError: name 'pr' is not defined
350 >>> proj0
351 Quantum object: dims = [[2], [2]], shape = (2, 2), type = oper, isherm = True
352 Qobj data =
353 [[1. 0.]
354  [0. 0.]]
355 >>> qutip.ket("0")
356 Quantum object: dims = [[2], [1]], shape = (2, 1), type = ket
357 Qobj data =
358 [[1.]
359  [0.]]
360
361 '''
362
363 guess_dynamics = objectives[0].mesolve(tlist, e_ops=[proj0, proj1])
364
365
366 # The plot of the population dynamics shows that the guess field
367 # does not transfer
368 # the initial state  $|\Psi_{\text{init}}\rangle = |\text{ket}\{0\}\rangle$  to the desired
369 # target state
370 #  $|\Psi_{\text{tgt}}\rangle = |\text{ket}\{1\}\rangle$ 
371 # (so the optimization will have something to do).
372
373
374 def plot_population(result, result_string='r', rounding_limit=5):
375     '''plot of the population dynamics
376     Inputs:
377     result      :
378     result_string : str
379                   The thing whose population dynamics is
380                   being plotted
381

```

```

382     rounding_limit : int
383         The no of digits to which the file no should be
384         fixed to. Default is 5
385
386     '''
387
388     # the random number
389     t_r_no = str(round(random_sample(), rounding_limit))
390     # fig, ax = plt.subplots()
391     pop_fig = figure()
392     pop_ax = axes()
393     pop_ax.plot(result.times, result.expect[0], label='0')
394     pop_ax.plot(result.times, result.expect[1], label='1')
395     pop_ax.legend()
396     pop_ax.set_xlabel('time')
397     pop_ax.set_ylabel('population')
398     pop_ax.set(title='Plot of population dynamics: ' + result_string)
399     pop_fig_rand_name = 'popula dynamics ' + result_string + t_r_no + '.pdf'
400     # Could also use date time
401     # fig_path_name = path + fig_name
402
403     pop_fig.savefig(pop_fig_rand_name)
404     cla()
405     close(pop_fig)
406     # plt.show(fig)
407     return None
408
409     # plt.show(fig)
410
411
412 # In[31]:
413
414
415 plot_population(guess_dynamics, result_string='guess')
416
417
418 # ## Optimize
419
420 # In the following we optimize the guess field  $\epsilon_0(t)$  such
421 # that the intended state-to-state transfer
422 #  $|\Psi_{\text{init}}\rangle \rightarrow$ 
423 #  $|\Psi_{\text{tgt}}\rangle$  is solved, via the `krotov` package's central
424 # `optimize_pulses` routine. It requires, besides the previously
425 # defined
426 # `objectives`, information about the optimization functional
427 #  $J_T$  (implicitly,
428 # via `chi_constructor`, which calculates the states  $|\chi\rangle =$ 
429 #  $\frac{J_T}{\langle \Psi \rangle}$ ).
430 #
431 # Here, we choose  $J_T = J_{T, \text{re}} = 1 - F_{\text{re}}$ 
432 # with  $F_{\text{re}}$ 
433 #  $= \langle \Psi(T) | \Psi_{\text{tgt}} \rangle$ , with  $|\Psi(T)\rangle$  the forward
434 # propagated state of  $|\Psi_{\text{init}}\rangle$ . Even though  $J_T$  is not
435 # explicitly
436 # required for the optimization, it is nonetheless useful to be able
437 # to calculate
438 # and print it as a way to provide some feedback about the optimization
439 # progress.
440 # Here, we pass as an `info_hook` the function
441 # `krotov.info_hooks.print_table`,
442 # using `krotov.functionals.J_T_re` (which implements the above
443 # functional; the
444 # `krotov` library contains implementations of all the "standard"
445 # functionals used in
446 # quantum control). This `info_hook` prints a tabular overview after
447 # each
448 # iteration, containing the value of  $J_T$ ,
449 # the magnitude of the integrated pulse

```

```

450 # update, and information on how much $J_T$ (and the full Krotov
451 # functional $J$)
452 # changes between iterations. It also stores the value of $J_T$
453 # internally in the
454 # `Result.info_vals` attribute.
455 #
456 # The value of $J_T$ can also be used to check the convergence.
457 # In this example,
458 # we limit the number of total iterations to 10, but more generally,
459 # we could use
460 # the `check_convergence` parameter to stop the optimization
461 # when $J_T$ falls below
462 # some threshold. Here, we only pass a function that checks that
463 # the value of
464 # $J_T$ is monotonically decreasing. The
465 # `krotov.convergence.check_monotonic_error` relies on
466 # `krotov.info_hooks.print_table` internally having stored the value
467 # of $J_T$ to
468 # the `Result.info_vals` in each iteration.
469
470 # In[32]:
471
472
473 '''
474 oct_result = optimize_pulses(
475     objectives,
476     pulse_options=pulse_options,
477     tlist=tlist,
478     propagator=expm,
479     chi_constructor=chis_re,
480     info_hook=print_table(J_T=J_T_re, out=f),
481     check_convergence=check_monotonic_error,
482     iter_stop=20,
483 )
484
485 opt_result = optimize_pulses(
486     objectives,
487     pulse_options=pulse_options,
488     tlist=tlist,
489     propagator=expm,
490     chi_constructor=chis_ss,
491     info_hook=print_table(J_T=J_T_ss, out=f),
492     check_convergence=Or(
493         value_below('1e-3', name='J_T'),
494         check_monotonic_error,
495     ),
496     store_all_pulses=True,
497 )
498 '''
499
500 opt_result = optimize_pulses(
501     objectives,
502     pulse_options=pulse_options,
503     tlist=tlist,
504     propagator=expm,
505     chi_constructor=chis_ss,
506     info_hook=print_table(J_T=J_T_ss, ),
507     check_convergence=Or(
508         value_below('1e-3', name='J_T'),
509         check_monotonic_error,
510     ),
511     store_all_pulses=True,
512 )
513
514 # In[33]:
515
516

```



```

517 print("opt_result\n", opt_result)
518 # print
519 print("opt_result\n", opt_result, file=f)
520
521 # ## Simulate the dynamics under the optimized field
522
523 # Having obtained the optimized control field, we can now plot it and
524 # calculate the population dynamics under this field.
525
526 # In[34]:
527 opt_pulse_index = 0
528 for pulse in opt_result.optimized_controls:
529     plot_pulse(pulse, tlist,
530                result_string='opt' + str(opt_pulse_index))
531     opt_pulse_index = opt_pulse_index + 1
532
533 # print("\n opt_result.optimized_controls[0]",
534 #       opt_result.optimized_controls[0], file=f)
535
536 # print("type of opt_result.optimized_controls[0]",
537 #       type(opt_result.optimized_controls[0]))
538 #
539 # In contrast to the dynamics under the guess field,
540 # the optimized field indeed
541 # drives the initial state  $|\Psi_{\text{init}}\rangle = |\text{ket}\{0\}\rangle$ 
542 # to the desired target
543 # state  $|\Psi_{\text{tgt}}\rangle = |\text{ket}\{1\}\rangle$ .
544
545 # In[35]:
546
547
548 opt_dynamics = opt_result.optimized_objectives[0].mesolve(
549     tlist, e_ops=[proj0, proj1])
550
551
552 # In[36]:
553
554 for i in range(len()):
555     plot_population(opt_dynamics,
556                    result_string=' optimized control')
557
558
559 '''
560 To gain some intuition on how the controls and the dynamics
561 change throughout the optimization procedure,
562 we can generate a plot of the control fields and the dynamics
563 after each iteration of the optimization algorithm.
564 This is possible because we set `store_all_pulses=True`
565 in the call to `optimize_pulses`,
566 which allows to recover the optimized controls from each
567 iteration from `Result.all_pulses`.
568 The flag is not set to True by default,
569 as for long-running optimizations with thousands or
570 tens of thousands iterations,
571 the storage of all control fields may require significant memory.
572 '''
573
574
575 def plot_iterations(opt_result):
576     """Plot the control fields in population dynamics over all iterations.
577
578     This depends on ``store_all_pulses=True`` in the call to
579     `optimize_pulses`.
580     """
581     it_fig, [ax_ctr, ax_dyn] = subplots(nrows=2, figsize=(8, 10))
582     n_iters = len(opt_result.iters)
583     for (iteration, pulses) in zip(opt_result.iters, opt_result.all_pulses):
584         controls = [

```

```

585         pulse_onto_tlist(pulse)
586         for pulse in pulses
587     ]
588     objectives = opt_result.objectives_with_controls(controls)
589     dynamics = objectives[0].mesolve(
590         opt_result.tlist, e_ops=[proj0, proj1]
591     )
592     if iteration == 0:
593         ls = '--' # dashed
594         alpha = 1 # full opacity
595         ctr_label = 'guess'
596         pop_labels = ['0 (guess)', '1 (guess)']
597     elif iteration == opt_result.iters[-1]:
598         ls = '-' # solid
599         alpha = 1 # full opacity
600         ctr_label = 'optimized'
601         pop_labels = ['0 (optimized)', '1 (optimized)']
602     else:
603         ls = '-' # solid
604         alpha = 0.5 * float(iteration) / float(n_iters) # max 50%
605         ctr_label = None
606         pop_labels = [None, None]
607     ax_ctr.plot(
608         dynamics.times,
609         controls[0],
610         label=ctr_label,
611         color='black',
612         ls=ls,
613         alpha=alpha,
614     )
615     ax_dyn.plot(
616         dynamics.times,
617         dynamics.expect[0],
618         label=pop_labels[0],
619         color='#1f77b4', # default blue
620         ls=ls,
621         alpha=alpha,
622     )
623     ax_dyn.plot(
624         dynamics.times,
625         dynamics.expect[1],
626         label=pop_labels[1],
627         color='#ff7f0e', # default orange
628         ls=ls,
629         alpha=alpha,
630     )
631     ax_dyn.legend()
632     ax_dyn.set_xlabel('time')
633     ax_dyn.set_ylabel('population')
634     ax_ctr.legend()
635     ax_ctr.set_xlabel('time')
636     ax_ctr.set_ylabel('control amplitude')
637     # plt.show(fig)
638
639     # ax.set(title='iterations: ')
640     it_fig_rand_name = 'Iterations.pdf'
641
642     it_fig.savefig(it_fig_rand_name)
643     cla()
644     close(it_fig)
645
646
647 plot_iterations(opt_result)
648 '''
649 The initial guess (dashed) and final optimized (solid)
650 control amplitude and resulting dynamics are shown with full opacity,
651 whereas the curves corresponding intermediate iterations

```

```

652 are shown with decreasing transparency.
653 '''
654 '''
655
656 Traceback (most recent call last):
657   File "k5.py", line 499, in <module>
658     [], tlist)
659   File "/home/tejas/Dropbox/Lab computer code to run/gkrape/krotov-try/
660     k5/cf2unitary.py", line 68, in cf2unitary
661     for U in U_list_tej:
662   File "/home/tejas/Dropbox/Lab computer code to run/gkrape/krotov-try/
663     k5/cf2unitary.py", line 66, in <genexpr>
664     for idx in range(M-1))
665   File "/home/tejas/Dropbox/Lab computer code to run/gkrape/krotov-try/
666     k5/cf2unitary.py", line 45, in _A_tej
667     H = H0 + sum([ug[j, idx] * H_ops[j] for j in range(J)])
668   File "/home/tejas/Dropbox/Lab computer code to run/gkrape/krotov-try/
669     k5/cf2unitary.py", line 45, in <listcomp>
670     H = H0 + sum([ug[j, idx] * H_ops[j] for j in range(J)])
671 IndexError: too many indices for array
672
673
674 >>> from numpy import array, arange
675 >>> yahoo = arange(5)
676 >>> yahoo
677 array([0, 1, 2, 3, 4])
678 >>> yar = array(yahoo)
679 >>> yar
680 array([0, 1, 2, 3, 4])
681 >>> yar.shape()
682 Traceback (most recent call last):
683   File "<stdin>", line 1, in <module>
684 TypeError: 'tuple' object is not callable
685 >>> yar.shape()
686 Traceback (most recent call last):
687   File "<stdin>", line 1, in <module>
688 TypeError: 'tuple' object is not callable
689 >>> yar.shape
690 (5,)
691 >>> yar_list = list(yar, yar + 2)
692 Traceback (most recent call last):
693   File "<stdin>", line 1, in <module>
694 TypeError: list expected at most 1 arguments, got 2
695 >>> yar_list = list([^[Cyar, yar + 2])
696   File "<stdin>", line 1
697     yar_list = list([ yar, yar + 2])
698                     ^
699 SyntaxError: invalid syntax
700 >>> yar_list = [yar, yar + 2]
701 >>> yar_list
702 [array([0, 1, 2, 3, 4]), array([2, 3, 4, 5, 6])]
703 >>> aarr_yar_list = array(yar_list)
704 >>> aarr_yar_list
705 array([[0, 1, 2, 3, 4],
706        [2, 3, 4, 5, 6]])
707 >>> lonely_yar_list = [yar]
708 >>> ar_lonely_yar_list = array(lonely_yar_list)
709 >>> ar_lonely_yar_list
710 array([[0, 1, 2, 3, 4]])
711 >>> ar_lonely_yar_list.shape
712 (1, 5)
713 >>> aarr_yar_list.shape
714 (2, 5)
715
716
717
718 '''
719 '''

```

```

720
721 a = zeros()
722 U_kopt = cf2unitary(array(array(opt_result.optimized_controls[0])),
723                       H[0], [H[1][1]],
724                       [], tlist)
725 tej_plotter(U, 'U')
726 tej_plotter(U_kopt, 'U_kopt')
727 '''
728 krotov_opt_controls = array([array(opt_result.optimized_controls[0]), ])
729 print('krotov_opt_controls.shape', krotov_opt_controls.shape)
730
731 U_kopt = cf2u(krotov_opt_controls,
732               H[0], [H[1][0]],
733               tlist)
734
735
736 def _overlap(A, B):
737     return (A.dag() * B).tr() / A.shape[0]
738     # return cy_overlap(A.data, B.data)
739
740
741 krotov_fid = _overlap(U, U_kopt)
742 abs_kf = abs(krotov_fid)
743 print("krotov_fid", krotov_fid)
744 print("krotov_fid", krotov_fid, file=f)
745 print('absolute value of krotov_fid', abs_kf,)
746 print('absolute value of krotov_fid', abs_kf, file=f)
747 tej_plotter(U, 'U')
748 tej_plotter(U_kopt, 'U_kopt')
749
750 savez('krotov-opt',
751       krotov_field=array(opt_result.optimized_controls[0])
752       )
753
754 '''
755 if opt_result.all_pulses:
756     savez('krotov_extra',
757           krotov_field_all_iter=opt_result.all_pulses)
758
759 '''
760 # GRAPE begins
761 print('\n\n GRAPE begins')
762 u_limits = None # [0, 1 * 2 * pi]
763 alpha = None
764 R = 600 # 150, 500
765 # Increasing no of of iterations improves fidelity
766 # H_labels = [r'$u_{1}$', ]
767
768 H_labels = [r'$u_{1x}$', r'$u_{1y}$', r'$u_{1z}$',
769             r'$u_{2x}$', r'$u_{1y}$', r'$u_{2z}$',
770             r'$u_{xx}$',
771             r'$u_{yy}$',
772             r'$u_{zz}$',
773             ]
774 # eps = 2*pi, None, pi, 3*pi, 10*pi
775 # Apparently lower the eps lower the absolute value of the fidelity.
776 # this fails if we increase the value of the eps to 10*pi
777 # Can also change to grape unitary adaptive, increase eps to 3*pi, 10*pi
778 grape_result = grape_unitary_adaptive(U, H[0], [H[1][0]],
779                                       R, tlist,
780                                       u_start=krotov_opt_controls,
781                                       u_limits=u_limits,
782                                       eps=3.5*pi,
783                                       alpha=alpha,
784                                       phase_sensitive=False,
785                                       progress_bar=TextProgressBar()
786                                       )

```

```

787
788 '''
789 grape_result = grape_unitary(U, H[0], [H[1][0]],
790                               R, tlist,
791                               u_start=krotov_opt_controls,
792                               u_limits=u_limits,
793                               eps=4*pi*1,
794                               alpha=alpha,
795                               phase_sensitive=False,
796                               progress_bar=TextProgressBar()
797                               )
798
799 '''
800 # Plotting the control field
801 gr_f_fig, gr_f_axes = plot_grape_control_fields(tlist,
802                                                  grape_result.u,
803                                                  H_labels,
804                                                  uniform_axes=True)
805
806
807 f_rand_name = 'f' + str(random_sample()) + '.pdf'
808 # Could also use date time
809 # fig_path_name = path + fig_name
810 gr_f_fig.savefig(f_rand_name)
811 cla()
812 close(gr_f_fig)
813
814 tej_plotter(grape_result.U_f, 'U_gr')
815 savez('grape_result',
816       resultu=grape_result.u, resultU_f=grape_result.U_f.full())
817
818 '''
819 U_gopt = cf2u(grape_result.u,
820               H[0], [H[1][0]],
821               tlist)
822 '''
823 print('grape_result.shape', grape_result.u.shape)
824 grape_fid = _overlap(U, grape_result.U_f)
825 abs_gf = abs(grape_fid)
826 print("grape_fid", grape_fid)
827 print("grape_fid", grape_fid, file=f)
828 print('absolute value of grape_fid', abs_gf)
829 print('absolute value of grape_fid', abs_gf, file=f)
830 # tej_plotter(U, 'U')
831 # tej_plotter(U_gopt, 'U_gopt')
832
833 # plot_compare(tlist, kcf, gcf_full, rounding_limit=5)
834 plot_compare(tlist, krotov_opt_controls,
835             grape_result.u, rounding_limit=5)
836
837 chdir('../')
838
839 print('We are in' + getcwd())

```