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

```
# # Optimization of a State-to-State Transfer in a Two-Level-System
from krotov.shapes import flattop
# from krotov import Objective
from krotov import optimize_pulses, gate_objectives
from krotov.propagators import expm
# from krotov.functionals import chis_re
from krotov.info_hooks import print_table
from krotov.convergence import check_monotonic_error, Or, value_below
# from krotov.functionals import J_T_re
from krotov.functionals import chis_ss
from krotov.functionals import J_T_ss
from krotov.structural_conversions import pulse_onto_tlist

# from qutip.control import grape_unitary,
from qutip.control import plot_grape_control_fields
from qutip.control import grape_unitary_adaptive
from qutip.ui.progressbar import TextProgressBar
# from cw import crude_writer
from qutip.operators import sigmax, sigmaz
from qutip.operators import sigmay, identity
from qutip.qip.gates import cnot
from qutip.tensor import tensor

from qutip.states import basis
from qutip import ket, ket2dm
from numpy import linspace, array
# from numpy import zeros
from numpy import pi
from numpy import savez
# from math import sqrt
from os import getcwd, mkdir, chdir
# from sys import stdout
from mcf import plot_compare
from numpy.random import random_sample
from tej_plotter import tej_plotter
from cf2u import cf2u
from matplotlib import use
from matplotlib.pyplot import figure, axes
from matplotlib.pyplot import subplots

from matplotlib import rcParams
from matplotlib.pyplot import style
```

```
from matplotlib.pyplot import cla, close
use("TkAgg")
pgf_with_rc_fonts = {"pgf.texsystem": "pdflatex"}
rcParams.update(pgf_with_rc_fonts)
style.use('seaborn-whitegrid')

print("Please input a run_no ")
print("Only integers allowed")
run_no = int(input())
run_dir_name = 'autorun' + str(run_no)
mkdir(run_dir_name)
chdir(run_dir_name)
print("Current directory")
print(getcwd())
f = open('workfile.txt', 'w')
# This first example illustrates the basic use of the `krotov`
# package by solving
# a simple canonical optimization problem: the transfer of
# population in a two
# level system.

# ## Two-level-Hamiltonian

# We consider the Hamiltonian $\op{H}_{0} = - \frac{\omega}{2}
# \op{\sigma}_{z}$, representing
# a simple qubit with energy level splitting $\omega$ in the basis
# $\{\ket{0},\ket{1}\}$. The control field $\epsilon(t)$ is assumed to
# couple via
# the Hamiltonian $\op{H}_{1}(t) = \epsilon(t) \op{\sigma}_{x}$ to the
# qubit,
# i.e., the control field effectively drives transitions between both
# qubit states.

# In[20]:
T = 1  # 1  # 5
tlist = linspace(0, T, 500)
# T = 1
# tlist = linspace(0, T, 100)


def hamiltonian(omega=1.0, ampl0=0.2):
    """Two-level-system Hamiltonian

    Args:
        omega (float): energy separation of the qubit levels
        ampl0 (float): constant amplitude of the driving field
    """

    H_ops = [tensor(sigmax(), identity(2)),
             tensor(sigmay(), identity(2)),
             tensor(sigmaz(), identity(2)),
             tensor(identity(2), sigmax()),
             tensor(identity(2), sigmay()),
             tensor(identity(2), sigmaz()),
             tensor(sigmax(), sigmax()) +
             tensor(sigmay(), sigmay()) +
             tensor(sigmaz(), sigmaz())]

    H0 = 0 * pi * (tensor(sigmax(), identity(2)) + tensor(identity(2),
                   sigmax()))

    # H0 = -0.5 * omega * sigmaz()
    # H0 = 1 * pi * sigmaz()
    # H1 = sigmax()

    def guess_control(t, args):
        return ampl0 * flattop(
            t, t_start=0, t_stop=T, t_rise=0.3*T/5, func="sinsq"
```

```python
        )

    H_qutip_list = [H0]
    for h in H_ops:
        H_qutip_list.append([h, guess_control])

    # [H0, [H1, guess_control]]

    return H_qutip_list


# H = hamiltonian(ampl0=2 * pi) # pass no change
# H = hamiltonian() # fail
# H = hamiltonian(ampl0=1.0) # fail
# H = hamiltonian(ampl0=2.0) # fail
H = hamiltonian(ampl0=3.0)

U = cnot()
# U = sigmax()
# The control field here switches on from zero at $t=0$
# to it's maximum amplitude
# 0.2 within the time period 0.3 (the switch-on shape
# is half the first period of
# a $\sin^2$ function). It switches off again in the time period 0.3
# before the
# final time $T=5$). We use a time grid with 500 time steps between 0
#  and $T$:


def plot_pulse(pulse, tlist, result_string='r', rounding_limit=5):
    '''
    plot of the pulse


    Inputs:

    rounding_limit : int
                     The no of digits to which the file no should be
                     fixed to. Default is 5

    '''

    # the random number
    t_r_no = str(round(random_sample(), rounding_limit))
    gp_fig = figure()
    gp_ax = axes()
    # gp_fig, gp_ax = subplots()
    if callable(pulse):
        pulse = array([pulse(t, args=None) for t in tlist])
    gp_ax.plot(tlist, pulse)
    gp_ax.set_xlabel('time')
    gp_ax.set_ylabel('pulse amplitude')
    gp_ax.set(title='Plot of ' + result_string + 'pulse')
    fig_rand_name = result_string + 'pulse' + t_r_no + '.pdf'
    # Could also use date time
    # fig_path_name = path + fig_name

    gp_fig.savefig(fig_rand_name)
    cla()
    close(gp_fig)
    # plt.show(fig)
    return gp_fig, gp_ax


gp_fig, gp_ax = plot_pulse(H[1][1], tlist, result_string='guess')


# ## Optimization target
```

```

# The `krotov` package requires the goal of the optimization to be
# described by a
# list of `Objective` instances. In this example, there is only a single
# objective: the state-to-state transfer from initial state
# $\ket{\Psi_{\init}} =
# \ket{0}$ to the target state $\ket{\Psi_{\tgt}} = \ket{1}$,
# under the dynamics
# of the Hamiltonian $\op{H}(t)$:


'''
objectives = [
    Objective(
        initial_state=ket("0"), target=ket("1"), H=H
    )
]
'''
2qubit_basis_states = [tensor(ket("0"), ket("0")),
                       tensor(ket("0"), ket("1")),
                       tensor(ket("1"), ket("0")),
                       tensor(ket("1"), ket("1")), ]

objectives = gate_objectives(
    basis_states=2qubit_basis_states,
    gate=U,
    H=H
    )
print("objectives[0]\n", objectives[0])
print("objectives[1]\n", objectives[1])
# print("objectives[2]\n", objectives[2])

print("objectives\n", objectives, file=f)


# In addition, we would like to maintain the property of the control
# field to be
# zero at $t=0$ and $t=T$, with a smooth switch-on and switch-off.
# We can define
# an "update shape" $S(t) \in [0, 1]$ for this purpose: Krotov's method
# will
# update the field at each point in time proportionally to $S(t)$;
# wherever
# $S(t)$ is zero, the optimization will not change the value of the
# control from
# the original guess.

def S(t):
    """Shape function for the field update"""
    return flattop(
        t, t_start=0, t_stop=T, t_rise=(0.3*T)/5,
        t_fall=(0.3*T)/5, func='sinsq'
    )


# Beyond the shape, Krotov's method uses a parameter $\lambda_a$
# for each control
# field that determines the overall magnitude of the respective field
# in each
# iteration (the smaller $\lambda_a$, the larger the update;
# specifically, the
# update is proportional to $\frac{S(t)}{\lambda_a}$). Both the
# update-shape
# $S(t)$ and the $\lambda_a$ parameter must be passed to the
# optimization routine
# as "pulse options":

'''
```

```
pulse_options = {
    H[1][1]: dict(lambda_a=5, shape=S)
}

pulse_options = {
    H[1][1]: dict(lambda_a=5, update_shape=S)
}
'''

pulse_options = {
    H[i][1]: dict(lambda_a=5, update_shape=S)
    for i in range(1, len(H_ops) + 1, 1)
}


# ## Simulate dynamics under the guess field

# Before running the optimization procedure, we first simulate the
# dynamics under the
# guess field $\epsilon_{0}(t)$. The following solves equation
# of motion for the
# defined objective, which contains the initial state
# $\ket{\Psi_{\init}}$ and
# the Hamiltonian $\op{H}(t)$ defining its evolution.
# This delegates to QuTiP's
# usual `mesolve` function.

# We use the projectors $\op{P}_0 = \ket{0}\bra{0}$
# and $\op{P}_1 = \ket{1}\bra{1}$ for calculating the population:

proj0 = ket2dm((basis(4, 0)))
proj1 = ket2dm(U*(basis(4, 0)))

'''
proj0 = ket2dm((basis(2, 0) + basis(2, 1))/(1/sqrt(2)))
proj1 = ket2dm((basis(2, 0) - basis(2, 1))/(1/sqrt(2)))

proj0 = ket2dm(basis(2, 0) + 1j*basis(2, 1))
proj1 = ket2dm(basis(2, 0) - 1j*basis(2, 1))

Python 3.7.4 (default, Aug 13 2019, 20:35:49)
[GCC 7.3.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import qutip
>>> basis(0, 2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'basis' is not defined
>>> from qutip.operators import basis
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
ImportError: cannot import name 'basis' from 'qutip.operators'
(/home/tejas/anaconda3/envs/krotov/lib/python3.7/site-packages/qutip/
operators.py)
>>> doc
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'doc' is not defined
>>> doc(qutip)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'doc' is not defined
>>> from qutip.states import basis
>>> basis(0, 2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File
  "/home/tejas/anaconda3/envs/krotov/lib/python3.7/site-packages/qutip/
```

```
  states.py", line 105, in basis
    raise ValueError("basis vector index need to be in n <= N-1")
ValueError: basis vector index need to be in n <= N-1
>>> basis(2, 0)
Quantum object: dims = [[2], [1]], shape = (2, 1), type = ket
Qobj data =
[[1.]
 [0.]]
>>> basis(2, 0) + 1j*basis(2, 1)
Quantum object: dims = [[2], [1]], shape = (2, 1), type = ket
Qobj data =
[[1.+0.j]
 [0.+1.j]]
>>> from qutip.operators import sigmay
>>> sigmay()*(basis(2, 0) + 1j*basis(2, 1))
Quantum object: dims = [[2], [1]], shape = (2, 1), type = ket
Qobj data =
[[1.+0.j]
 [0.+1.j]]
>>>


proj0 = ket2dm(ket("0"))
proj1 = ket2dm(ket("1"))
'''
'''
Python 3.7.4 (default, Aug 13 2019, 20:35:49)
[GCC 7.3.0] :: Anaconda, Inc. on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> import qutip
>>> proj0 = qutip.ket2dm(qutip.ket("0"))
>>> pr
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
NameError: name 'pr' is not defined
>>> proj0
Quantum object: dims = [[2], [2]], shape = (2, 2), type = oper, isherm = True
Qobj data =
[[1. 0.]
 [0. 0.]]
>>> qutip.ket("0")
Quantum object: dims = [[2], [1]], shape = (2, 1), type = ket
Qobj data =
[[1.]
 [0.]]

'''

guess_dynamics = objectives[0].mesolve(tlist, e_ops=[proj0, proj1])


# The plot of the population dynamics shows that the guess field
# does not transfer
# the initial state $\ket{\Psi_{\init}} = \ket{0}$ to the desired
# target state
# $\ket{\Psi_{\tgt}} = \ket{1}$
# (so the optimization will have something to do).


def plot_population(result, result_string='r', rounding_limit=5):
    '''plot of the population dynamics
    Inputs:
    result          :
    result_string   : str
                      The thing whose population dynamics is
                      being plotted

```

```
    rounding_limit : int
                     The no of digits to which the file no should be
                     fixed to. Default is 5

    '''

    # the random number
    t_r_no = str(round(random_sample(), rounding_limit))
    # fig, ax = plt.subplots()
    pop_fig = figure()
    pop_ax = axes()
    pop_ax.plot(result.times, result.expect[0], label='0')
    pop_ax.plot(result.times, result.expect[1], label='1')
    pop_ax.legend()
    pop_ax.set_xlabel('time')
    pop_ax.set_ylabel('population')
    pop_ax.set(title='Plot of population dynamics: ' + result_string)
    pop_fig_rand_name = 'popula dynamics ' + result_string + t_r_no + '.pdf'
    # Could also use date time
    # fig_path_name = path + fig_name

    pop_fig.savefig(pop_fig_rand_name)
    cla()
    close(pop_fig)
    # plt.show(fig)
    return None

    # plt.show(fig)


# In[31]:


plot_population(guess_dynamics, result_string='guess')


# ## Optimize

# In the following we optimize the guess field $\epsilon_{0}(t)$ such
# that the intended state-to-state transfer
# $\ket{\Psi_{\init}} \rightarrow
# \ket{\Psi_{\tgt}}$ is solved, via the `krotov` package's central
# `optimize_pulses` routine.  It requires, besides the previously
# defined
# `objectives`, information about the optimization functional
# $J_T$ (implicitly,
# via `chi_constructor`, which calculates the states $\ket{\chi} =
# \frac{J_T}{\bra{\Psi}}$).
#
# Here, we choose $J_T = J_{T,\text{re}} = 1 - F_{\text{re}}$
# with $F_{\text{re}}
# = \Re\Braket{\Psi(T)}{\Psi_{\tgt}}$, with $\ket{\Psi(T)}$ the forward
# propagated state of $\ket{\Psi_{\init}}$. Even though $J_T$ is not
# explicitly
# required for the optimization, it is nonetheless useful to be able
# to calculate
# and print it as a way to provide some feedback about the optimization
# progress.
# Here, we pass as an `info_hook` the function
#  `krotov.info_hooks.print_table`,
# using `krotov.functionals.J_T_re` (which implements the above
# functional; the
# `krotov` library contains implementations of all the "standard"
# functionals used in
# quantum control). This `info_hook` prints a tabular overview after
# each
# iteration, containing the value of $J_T$,
# the magnitude of the integrated pulse
```

```python
# update, and information on how much $J_T$ (and the full Krotov
# functional $J$)
# changes between iterations. It also stores the value of $J_T$
# internally in the
# `Result.info_vals` attribute.
#
# The value of $J_T$ can also be used to check the convergence.
# In this example,
# we limit the number of total iterations to 10, but more generally,
# we could use
# the `check_convergence` parameter to stop the optimization
# when $J_T$ falls below
# some threshold. Here, we only pass a function that checks that
# the value of
# $J_T$ is monotonically decreasing. The
# `krotov.convergence.check_monotonic_error` relies on
# `krotov.info_hooks.print_table` internally having stored the value
# of $J_T$ to
# the `Result.info_vals` in each iteration.

# In[32]:


'''
oct_result = optimize_pulses(
    objectives,
    pulse_options=pulse_options,
    tlist=tlist,
    propagator=expm,
    chi_constructor=chis_re,
    info_hook=print_table(J_T=J_T_re, out=f),
    check_convergence=check_monotonic_error,
    iter_stop=20,
)

opt_result = optimize_pulses(
    objectives,
    pulse_options=pulse_options,
    tlist=tlist,
    propagator=expm,
    chi_constructor=chis_ss,
    info_hook=print_table(J_T=J_T_ss, out=f),
    check_convergence=Or(
        value_below('1e-3', name='J_T'),
        check_monotonic_error,
    ),
    store_all_pulses=True,
)
'''

opt_result = optimize_pulses(
    objectives,
    pulse_options=pulse_options,
    tlist=tlist,
    propagator=expm,
    chi_constructor=chis_ss,
    info_hook=print_table(J_T=J_T_ss, ),
    check_convergence=Or(
        value_below('1e-3', name='J_T'),
        check_monotonic_error,
    ),
    store_all_pulses=True,
)

# In[33]:
```

```python
print("opt_result\n", opt_result)
# print
print("opt_result\n", opt_result, file=f)

# ## Simulate the dynamics under the optimized field

# Having obtained the optimized control field, we can now plot it and
# calculate the population dynamics under this field.

# In[34]:
opt_pulse_index = 0
for pulse in opt_result.optimized_controls:
    plot_pulse(pulse, tlist,
               result_string='opt' + str(opt_pulse_index))
    opt_pulse_index = opt_pulse_index + 1

# print("\n opt_result.optimized_controls[0]",
#      opt_result.optimized_controls[0], file=f)

# print("type of opt_result.optimized_controls[0]",
#       type(opt_result.optimized_controls[0]))
#
# In contrast to the dynamics under the guess field,
# the optimized field indeed
# drives the initial state $\ket{\Psi_{\init}} = \ket{0}$
# to the desired target
# state $\ket{\Psi_{\tgt}} = \ket{1}$.

# In[35]:


opt_dynamics = opt_result.optimized_objectives[0].mesolve(
    tlist, e_ops=[proj0, proj1])


# In[36]:

for i in range(len()):
plot_population(opt_dynamics,
                result_string=' optimized control')


'''
To gain some intuition on how the controls and the dynamics
change throughout the optimization procedure,
we can generate a plot of the control fields and the dynamics
after each iteration of the optimization algorithm.
This is possible because we set `store_all_pulses=True`
in the call to `optimize_pulses`,
which allows to recover the optimized controls from each
iteration from `Result.all_pulses`.
The flag is not set to True by default,
as for long-running optimizations with thousands or
tens of thousands iterations,
the storage of all control fields may require significant memory.
'''


def plot_iterations(opt_result):
    """Plot the control fields in population dynamics over all iterations.

    This depends on ``store_all_pulses=True`` in the call to
    `optimize_pulses`.
    """
    it_fig, [ax_ctr, ax_dyn] = subplots(nrows=2, figsize=(8, 10))
    n_iters = len(opt_result.iters)
    for (iteration, pulses) in zip(opt_result.iters, opt_result.all_pulses):
        controls = [
```

```
            pulse_onto_tlist(pulse)
            for pulse in pulses
        ]
        objectives = opt_result.objectives_with_controls(controls)
        dynamics = objectives[0].mesolve(
            opt_result.tlist, e_ops=[proj0, proj1]
        )
        if iteration == 0:
            ls = '--'  # dashed
            alpha = 1  # full opacity
            ctr_label = 'guess'
            pop_labels = ['0 (guess)', '1 (guess)']
        elif iteration == opt_result.iters[-1]:
            ls = '-'  # solid
            alpha = 1  # full opacity
            ctr_label = 'optimized'
            pop_labels = ['0 (optimized)', '1 (optimized)']
        else:
            ls = '-'  # solid
            alpha = 0.5 * float(iteration) / float(n_iters)  # max 50%
            ctr_label = None
            pop_labels = [None, None]
        ax_ctr.plot(
            dynamics.times,
            controls[0],
            label=ctr_label,
            color='black',
            ls=ls,
            alpha=alpha,
        )
        ax_dyn.plot(
            dynamics.times,
            dynamics.expect[0],
            label=pop_labels[0],
            color='#1f77b4',  # default blue
            ls=ls,
            alpha=alpha,
        )
        ax_dyn.plot(
            dynamics.times,
            dynamics.expect[1],
            label=pop_labels[1],
            color='#ff7f0e',  # default orange
            ls=ls,
            alpha=alpha,
        )
    ax_dyn.legend()
    ax_dyn.set_xlabel('time')
    ax_dyn.set_ylabel('population')
    ax_ctr.legend()
    ax_ctr.set_xlabel('time')
    ax_ctr.set_ylabel('control amplitude')
    # plt.show(fig)

    # ax.set(title='iterations: ')
    it_fig_rand_name = 'Iterations.pdf'

    it_fig.savefig(it_fig_rand_name)
    cla()
    close(it_fig)


plot_iterations(opt_result)
'''
The initial guess (dashed) and final optimized (solid)
control amplitude and resulting dynamics are shown with full opacity,
whereas the curves corresponding intermediate iterations
```

```
are shown with decreasing transparency.
'''
'''

Traceback (most recent call last):
  File "k5.py", line 499, in <module>
    [], tlist)
  File "/home/tejas/Dropbox/Lab computer code to run/gkrape/krotov-try/
  k5/cf2unitary.py", line 68, in cf2unitary
    for U in U_list_tej:
  File "/home/tejas/Dropbox/Lab computer code to run/gkrape/krotov-try/
  k5/cf2unitary.py", line 66, in <genexpr>
    for idx in range(M-1))
  File "/home/tejas/Dropbox/Lab computer code to run/gkrape/krotov-try/
  k5/cf2unitary.py", line 45, in _A_tej
    H = H0 + sum([ug[j, idx] * H_ops[j] for j in range(J)])
  File "/home/tejas/Dropbox/Lab computer code to run/gkrape/krotov-try/
  k5/cf2unitary.py", line 45, in <listcomp>
    H = H0 + sum([ug[j, idx] * H_ops[j] for j in range(J)])
IndexError: too many indices for array


>>> from numpy import array, arange
>>> yahoo = arange(5)
>>> yahoo
array([0, 1, 2, 3, 4])
>>> yar = array(yahoo)
>>> yar
array([0, 1, 2, 3, 4])
>>> yar.shape()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object is not callable
>>> yar.shape()
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: 'tuple' object is not callable
>>> yar.shape
(5,)
>>> yar_list  = list(yar, yar +  2)
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
TypeError: list expected at most 1 arguments, got 2
>>> yar_list  = list([^[[Cyar, yar +  2])
  File "<stdin>", line 1
    yar_list  = list([ yar, yar +  2])
                      ^
SyntaxError: invalid syntax
>>> yar_list  = [yar, yar +  2]
>>> yar_list
[array([0, 1, 2, 3, 4]), array([2, 3, 4, 5, 6])]
>>> aarr_yar_list = array(yar_list)
>>> aarr_yar_list
array([[0, 1, 2, 3, 4],
       [2, 3, 4, 5, 6]])
>>> lonely_yar_list = [yar]
>>> ar_lonely_yar_list =  array(lonely_yar_list)
>>> ar_lonely_yar_list
array([[0, 1, 2, 3, 4]])
>>> ar_lonely_yar_list.shape
(1, 5)
>>> aarr_yar_list.shape
(2, 5)



'''
'''
```

```python

a = zeros()
U_kopt = cf2unitary(array(array(opt_result.optimized_controls[0])),
                    H[0], [H[1][1]],
                    [], tlist)
tej_plotter(U, 'U')
tej_plotter(U_kopt, 'U_kopt')
'''
krotov_opt_controls = array([array(opt_result.optimized_controls[0]), ])
print('krotov_opt_controls.shape', krotov_opt_controls.shape)

U_kopt = cf2u(krotov_opt_controls,
              H[0], [H[1][0]],
              tlist)


def _overlap(A, B):
    return (A.dag() * B).tr() / A.shape[0]
    # return cy_overlap(A.data, B.data)


krotov_fid = _overlap(U, U_kopt)
abs_kf = abs(krotov_fid)
print("krotov_fid", krotov_fid)
print("krotov_fid", krotov_fid, file=f)
print('absolute value of krotov_fid', abs_kf,)
print('absolute value of krotov_fid', abs_kf, file=f)
tej_plotter(U, 'U')
tej_plotter(U_kopt, 'U_kopt')

savez('krotov-opt',
      krotov_field=array(opt_result.optimized_controls[0])
      )

'''
if opt_result.all_pulses:
    savez('krotov_extra',
          krotov_field_all_iter=opt_result.all_pulses)

'''
# GRAPE begins
print('\n\n GRAPE begins')
u_limits = None  # [0, 1 * 2 * pi]
alpha = None
R = 600  # 150, 500
# Increasing no of of iterations improves fidelity
# H_labels = [r'$u_{1}$', ]

H_labels = [r'$u_{1x}$', r'$u_{1y}$', r'$u_{1z}$',
            r'$u_{2x}$', r'$u_{1y}$', r'$u_{2z}$',
            r'$u_{xx}$',
            r'$u_{yy}$',
            r'$u_{zz}$',
            ]
# eps =  2*pi, None, pi, 3*pi, 10*pi
# Apparently lower the eps lower the  absolute value of the fidelity.
# this fails if we increase the value of the eps to 10*pi
# Can also change to grape unitary adaptive, increase eps to 3*pi, 10*pi
grape_result = grape_unitary_adaptive(U, H[0], [H[1][0]],
                                      R, tlist,
                                      u_start=krotov_opt_controls,
                                      u_limits=u_limits,
                                      eps=3.5*pi,
                                      alpha=alpha,
                                      phase_sensitive=False,
                                      progress_bar=TextProgressBar()
                                      )
```

```

'''
grape_result = grape_unitary(U, H[0], [H[1][0]],
                             R, tlist,
                             u_start=krotov_opt_controls,
                             u_limits=u_limits,
                             eps=4*pi*1,
                             alpha=alpha,
                             phase_sensitive=False,
                             progress_bar=TextProgressBar()
                             )

'''
# Plotting the control field
gr_f_fig, gr_f_axes = plot_grape_control_fields(tlist,
                                                grape_result.u,
                                                H_labels,
                                                uniform_axes=True)


f_rand_name = 'f' + str(random_sample()) + '.pdf'
# Could also use date time
# fig_path_name = path + fig_name
gr_f_fig.savefig(f_rand_name)
cla()
close(gr_f_fig)

tej_plotter(grape_result.U_f, 'U_gr')
savez('grape_result',
      resultu=grape_result.u, resultU_f=grape_result.U_f.full())

'''
U_gopt = cf2u(grape_result.u,
              H[0], [H[1][0]],
              tlist)
'''
print('grape_result.shape', grape_result.u.shape)
grape_fid = _overlap(U, grape_result.U_f)
abs_gf = abs(grape_fid)
print("grape_fid", grape_fid)
print("grape_fid", grape_fid, file=f)
print('absolute value of grape_fid', abs_gf)
print('absolute value of grape_fid', abs_gf, file=f)
# tej_plotter(U, 'U')
# tej_plotter(U_gopt, 'U_gopt')

# plot_compare(tlist, kcf, gcf_full, rounding_limit=5)
plot_compare(tlist, krotov_opt_controls,
             grape_result.u, rounding_limit=5)

chdir('../')

print('We are in' + getcwd())
```