oqcg

February 17, 2019

1 Imports

Need to have jate.py in your folder

```
In [241]: %run jate.py #will import everything
```

2 Next chapter

2.1 memory clear (uses regex, so be careful)

```
In [242]: %reset_selective -f var1, var2  # replace var1, var2 with your defined ones
```

2.2 Building parts

2.2.1 Building the things to be calculated only once

```
In [243]: def maker(omega_1, H_0, H_1, T_s, Lin, d=2, gamma=0.1):
              r"""maker
              Makes all the things that remain constant throught the program, but are
              repeatedly used.


              Parameters
              ----------
              omega_1 : float
                        frequency corresponding to half of the difference between
                        energy levels of the qubit

              H_0     : Qobj
                        Bare Hamiltonian

              H_1     : Qobj
                        Interaction Hamiltonian

              T_s     : Qobj
                        Unitary to be implemented in the Hilbert space
```

```python
    Lin      : Qobj
               Linbladian operators

    d        : int
               Dimension of the matrix. Defaults to 2

    gamma    : float
               Damping constant of the Linbladian


    Returns
    -------

    ih0      : Qobj
               $I\otimes H_{0}$

    ih1      : Qobj
               $I\otimes H_{1}$

    h0ci     : Qobj
               $H_{0}^{*}\otimes I $

    h1ci     : Qobj
               $H_{1}^{*}\otimes I $

    T        : Qobj
               Target unitary transformed to the Liouville space

    linbladian : Qobj
                 The full lindbladian term as it appears on transformation to
                 the Liouville space.

    """
    I = identity(d)
    L_I = tensor(I, I)
    ih0 = tensor(I, H_0)
    ih1 = tensor(I, H_1)
    h0ci = tensor(H_0.conj(), I)
    h1ci = tensor(H_1.conj(), I)
    x_k = ih1 - h1ci
    term1 = tensor(Lin.trans(), Lin)
    term2 = tensor(I, ((Lin.dag())*(Lin)))
    term3 = tensor(((Lin.trans())*(Lin.conj())), I)
    lindbladian = 1j*(gamma)*(term1 - 0.5*(term2 + term3))
    T = tensor(T_s.trans(), T_s) # Transforming $T_{s}$ to liouville space


    return ih0, ih1, h0ci, h1ci, x_k, lindbladian, T, L_I
```

```
In [244]: omega_1 = 0.5
          H_0 = omega_1*sigmaz()
          H_1 = sigmay()
          T_s = sigmax()
          Lin = sigmaz()
          gamma = 0.1 # check for default value
          ih0, ih1, h0ci, h1ci, x_k, lindbladian, T, L_I  = maker(omega_1,
                                                                   H_0, H_1, T_s,
                                                                   Lin, d=2, gamma=gamma)
```

```
In [245]: gamma
```

```
Out[245]: 0.1
```

```
In [246]: L_I
```

```
Out[246]:
```

Quantum object: dims = [[2, 2], [2, 2]], shape = (4, 4), type = oper, isherm = True

$$\begin{pmatrix} 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \end{pmatrix}$$

2.2.2 Building $A(t)$

```
In [247]: def A(xi):
              r"""making $A(t)$"""
              A = ih0 - h0ci + xi*(ih1 - h1ci) + lindbladian
              return A
```

```
In [248]: A(0.5)
```

```
Out[248]:
```

Quantum object: dims = [[2, 2], [2, 2]], shape = (4, 4), type = oper, isherm = False

$$\begin{pmatrix} 0.0 & -0.500j & -0.500j & 0.0 \\ 0.500j & (-1.0-0.200j) & 0.0 & -0.500j \\ 0.500j & 0.0 & (1.0-0.200j) & -0.500j \\ 0.0 & 0.500j & 0.500j & 0.0 \end{pmatrix}$$

2.2.3 Building $L(t)$ and the Identity in the Liouville space

```
In [249]: def L(xi, dt):
              r"""Making $L(t) from $A(t)$"""
              L = (-1j*A(xi)*dt).expm()
              return L
```

```
In [250]: L(0.5, 0.001)
```

Out[250]:

Quantum object: dims = [[2, 2], [2, 2]], shape = (4, 4), type = oper, isherm = False

$$\left(\begin{array}{cccc} 1.000 & (-4.999\times10^{-04}-2.500\times10^{-07}j) & (-4.999\times10^{-04}+2.500\times10^{-07}j) & \\ (4.999\times10^{-04}+2.500\times10^{-07}j) & (1.000+9.998\times10^{-04}j) & -2.500\times10^{-07} & (- \\ (4.999\times10^{-04}-2.500\times10^{-07}j) & -2.500\times10^{-07} & (1.000-9.998\times10^{-04}j) & (- \\ 2.500\times10^{-07} & (4.999\times10^{-04}+2.500\times10^{-07}j) & (4.999\times10^{-04}-2.500\times10^{-07}j) & \end{array}\right.$$

2.3 Major functions

2.3.1 Major functions 1

In [251]:
```
# building the function to optimize (optimizee)
def L_vec(xi_vec, dt):
    r"""Building the vector of differential $L(t)$"""
    L_vec = [L(xi, dt) for xi in xi_vec]
    return L_vec
```

In [252]:
```
def fidelity_calc(A, B):
    r"""Making a generalised fidelity function"""
    first_part = (A - B).dag()
    second_part = (A - B)
    f_int = (first_part* second_part)
    f = f_int.tr()
    return f
```

In [253]:
```
def L_full_maker(xi_vec, dt):
    r"""Building the $L(t)$ for the total time $t$"""
    xi_vec_size = xi_vec.size # finding the size of xi
    L_full = L_I # Identity for the for loop of L
    L_v = L_vec(xi_vec, dt) # calling L_vec
    for i in range(xi_vec_size): # generating L_full
        L_full = L_full*L_v[xi_vec_size - 1 - i]
    return L_full
```

In [254]:
```
def F(xi_vec, dt):
    r"""Using the fidelity metric to find out the closeness between $T$
    and $L(t)$"""
    L_full = L_full_maker(xi_vec, dt)
    F = real(-fidelity_calc(T, L_full))
    return F
```

2.3.2 Testing major functions 1

In [255]:
```
fidelity_calc(sigmax(), sigmay())
```

Out[255]: 4.0

In [256]:
```
fidelity_calc(sigmay(), sigmay())
```

Out[256]: 0.0

```
In [257]: xi_vec_test = array([1.0, 2.0])
          xi_vec_test
```

Out[257]: array([1., 2.])

```
In [258]: xi_vec_test.size
```

Out[258]: 2

```
In [259]: w_vec = [xi**2 for xi in xi_vec_test]
          w_vec
```

Out[259]: [1.0, 4.0]

```
In [260]: # F(xi_vec, dt)
          F(xi_vec_test, 0.001)
```

Out[260]: -7.998400634493138

```
In [261]: L_v = L_vec(xi_vec_test, 0.001)
```

```
In [262]: L_v
```

```
Out[262]: [Quantum object: dims = [[2, 2], [2, 2]], shape = (4, 4), type = oper, isherm = False
           Qobj data =
           [[ 9.99999000e-01+0.00000000e+00j -9.99899173e-04-4.99933130e-07j
             -9.99899173e-04+4.99933130e-07j  9.99932920e-07+0.00000000e+00j]
            [ 9.99899173e-04+4.99933130e-07j  9.99798520e-01+9.99799187e-04j
             -9.99866260e-07+0.00000000e+00j -9.99899173e-04-4.99933130e-07j]
            [ 9.99899173e-04-4.99933130e-07j -9.99866260e-07+0.00000000e+00j
              9.99798520e-01-9.99799187e-04j -9.99899173e-04+4.99933130e-07j]
            [ 9.99932920e-07+0.00000000e+00j  9.99899173e-04+4.99933130e-07j
              9.99899173e-04-4.99933130e-07j  9.99999000e-01+0.00000000e+00j]],
           Quantum object: dims = [[2, 2], [2, 2]], shape = (4, 4), type = oper, isherm = False
           Qobj data =
           [[ 9.99996000e-01+0.00000000e+00j -1.99979435e-03-9.99865260e-07j
             -1.99979435e-03+9.99865260e-07j  3.99972768e-06+0.00000000e+00j]
            [ 1.99979435e-03+9.99865260e-07j  9.99795521e-01+9.99797187e-04j
             -3.99946104e-06+0.00000000e+00j -1.99979435e-03-9.99865260e-07j]
            [ 1.99979435e-03-9.99865260e-07j -3.99946104e-06+0.00000000e+00j
              9.99795521e-01-9.99797187e-04j -1.99979435e-03+9.99865260e-07j]
            [ 3.99972768e-06+0.00000000e+00j  1.99979435e-03+9.99865260e-07j
              1.99979435e-03-9.99865260e-07j  9.99996000e-01+0.00000000e+00j]]]
```

2.3.3 Major Functions 2

```
In [263]: def L_comma_k_maker(xi_vec, k, dt):
              r"""Making of the derivative of full $L(t)$ at time $t_{k}$"""
              N = xi_vec.size
              # Determining the size of xi, and thus the time_steps indirectly.
              L_v = L_vec(xi_vec, dt)# Making of the full $L(t)$
              inner_part = L_I # Beginner for the for loop
              for i in range(N):
                  if i == ( N - 1 - k ):
                      # The step at which $X_{k}(t)$ has to be inserted
                      inner_part = inner_part*x_k*L_v[k - 1]
                  else:
                      # Usual multiplications of $L_{k}$
                      inner_part = inner_part*L_v[N - 1 - i]
              l_comma_k = inner_part
              return l_comma_k
```

```
In [264]: # L_comma_k_maker(xi_vec, k, dt)
          L_comma_k_maker(xi_vec_test, 2, 0.001)
```

Out[264]:

Quantum object: dims = [[2, 2], [2, 2]], shape = (4, 4), type = oper, isherm = False

$$\begin{pmatrix} 1.000 & (-0.003-3.499\times10^{-06}j) & (-0.003+3.499\times10^{-06}j) & 8.999 \\ (0.003+2.499\times10^{-06}j) & (1.000+0.002j) & (-8.998\times10^{-06}+2.999\times10^{-09}j) & (-0.003-2 \\ (0.003-2.499\times10^{-06}j) & (-8.998\times10^{-06}-2.999\times10^{-09}j) & (1.000-0.002j) & (-0.003+2 \\ 8.999\times10^{-06} & (0.003+3.499\times10^{-06}j) & (0.003-3.499\times10^{-06}j) & 1 \end{pmatrix}$$

```
In [265]: def updater(xi_vec, dt, epsilon):
              r"""Implementing the GRAPE update step"""
              xi_vec_size = xi_vec.size # finding the size of xi
              L_full = L_full_maker(xi_vec, dt)
              di = []
              for k in range(xi_vec_size):
                  # Building the thing to be added to the old function
                  L_comma_k = L_comma_k_maker(xi_vec, k, dt)
                  differentiated = T - L_comma_k
                  plain = T - L_full
                  c = -differentiated.dag()*plain
                  d = -plain.dag()*differentiated
                  inside = c.tr() + d.tr()
                  di.append(epsilon*inside)

              diff = array(di)
              xi_new_vec = xi_vec + diff
              return diff, xi_new_vec
```

In [266]:
```
#  updater(xi_vec, dt, epsilon)
updater(xi_vec_test, 0.001, 0.001)
```

Out[266]:
```
(array([-0.008+0.j, -0.008+0.j]), array([0.992+0.j, 1.992+0.j]))
```

2.4 Qutip grape for closed system

In [267]:
```
import time
```

In [268]:
```
total_time_evo = 2*pi # total time allowed for evolution
```

In [269]:
```
times = linspace(0, total_time_evo, 500)
```

In [270]:
```
# vector of times at which discretization
# is carried out
```

In [271]:
```
U = T_s
U
```

Out[271]:

Quantum object: dims = [[2], [2]], shape = (2, 2), type = oper, isherm = True

$$\begin{pmatrix} 0.0 & 1.0 \\ 1.0 & 0.0 \end{pmatrix}$$

In [272]:
```
R = 500
```

In [273]:
```
H_ops = [H_1]
H_ops
```

Out[273]:
```
[Quantum object: dims = [[2], [2]], shape = (2, 2), type = oper, isherm = True
 Qobj data =
 [[0.+0.j 0.-1.j]
  [0.+1.j 0.+0.j]]]
```

In [274]:
```
H_labels = [r'$g_{no diss}$']
H_labels
```

Out[274]:
```
['$g_{no diss}$']
```

In [275]:
```
H0 = H_0
H0
```

Out[275]:

Quantum object: dims = [[2], [2]], shape = (2, 2), type = oper, isherm = True

$$\begin{pmatrix} 0.500 & 0.0 \\ 0.0 & -0.500 \end{pmatrix}$$

In [276]:
```
c_ops = []
```

```
In [277]: from qutip.control.grape import plot_grape_control_fields, _overlap
          from qutip.control.grape import grape_unitary_adaptive, cy_grape_unitary
```

```
In [278]: from scipy.interpolate import interp1d
          from qutip.ui.progressbar import TextProgressBar, EnhancedTextProgressBar
```

```
In [279]: u0 = array([rand(len(times)) * 2 * pi * 0.05 for _ in range(len(H_ops))])
```

```
In [280]: from numpy import convolve
          u0 = [convolve(ones(10)/10, u0[idx,:], mode='same') for idx in range(len(H_ops))]
```

```
In [281]: u_limits = None #[0, 1 * 2 * pi]
          alpha = None
```

```
In [283]: result = cy_grape_unitary(U, H0, H_ops, R, times, u_start=u0, u_limits=u_limits,
                                    eps=2*np.pi*1, alpha=alpha, phase_sensitive=False,
                                    progress_bar=TextProgressBar())
```

```
    ---------------------------------------------------------------------------

    KeyboardInterrupt                         Traceback (most recent call last)

    <ipython-input-283-1cb301d5355f> in <module>()
      1 result = cy_grape_unitary(U, H0, H_ops, R, times, u_start=u0, u_limits=u_limits,
      2                           eps=2*np.pi*1, alpha=alpha, phase_sensitive=False,
----> 3                           progress_bar=TextProgressBar())


    ~/anaconda3/envs/qutip-env/lib/python3.6/site-packages/qutip/control/grape.py in cy_gra
    385
    386             U_list = [(-1j * _H_idx(idx) * dt).expm().data
--> 387                       for idx in range(M-1)]
    388
    389         U_f_list = []


    ~/anaconda3/envs/qutip-env/lib/python3.6/site-packages/qutip/control/grape.py in <list
    385
    386             U_list = [(-1j * _H_idx(idx) * dt).expm().data
--> 387                       for idx in range(M-1)]
    388
    389         U_f_list = []


    ~/anaconda3/envs/qutip-env/lib/python3.6/site-packages/qutip/control/grape.py in _H_id
    382         else:
    383             def _H_idx(idx):
--> 384                 return H0 + sum([u[r, j, idx] * H_ops[j] for j in range(J)])
```

```
    385
    386             U_list = [(-1j * _H_idx(idx) * dt).expm().data


    ~/anaconda3/envs/qutip-env/lib/python3.6/site-packages/qutip/qobj.py in __radd__(self,
    461         ADDITION with Qobj on RIGHT [ ex. 4+Qobj ]
    462         """
--> 463         return self + other
    464
    465     def __sub__(self, other):


    ~/anaconda3/envs/qutip-env/lib/python3.6/site-packages/qutip/qobj.py in __add__(self,
    368
    369         if not isinstance(other, Qobj):
--> 370             other = Qobj(other)
    371
    372         if np.prod(other.shape) == 1 and np.prod(self.shape) != 1:


    ~/anaconda3/envs/qutip-env/lib/python3.6/site-packages/qutip/qobj.py in __init__(self,
    306                                 np.integer, np.floating, np.complexfloating)):
    307             # if input is int, float, or complex then convert to array
--> 308             _tmp = sp.csr_matrix([[inpt]], dtype=complex)
    309             self._data = fast_csr_matrix((_tmp.data, _tmp.indices, _tmp.indptr),
    310                                          shape=_tmp.shape)


    ~/anaconda3/envs/qutip-env/lib/python3.6/site-packages/scipy/sparse/compressed.py in _
     77                         self.format)
     78             from .coo import coo_matrix
---> 79             self._set_self(self.__class__(coo_matrix(arg1, dtype=dtype)))
     80
     81         # Read matrix dimensions given, if any


    ~/anaconda3/envs/qutip-env/lib/python3.6/site-packages/scipy/sparse/coo.py in __init__
    188
    189         if dtype is not None:
--> 190             self.data = self.data.astype(dtype, copy=False)
    191
    192         self._check()


    KeyboardInterrupt:
```

Plot of optimized control field without dissipation

```
In [ ]: plot_grape_control_fields(times,
                                   result.u / (2 * np.pi), H_labels, uniform_axes=True);
```

```
In [ ]: U
```

```
In [ ]: result.U_f
```

2.5 Analysis of result of qutip grape for closed system

```
In [ ]: result.U_f/result.U_f[0,0]
```

```
In [ ]: matrix_histogram(U)
```

```
In [ ]: matrix_histogram(result.U_f)
```

```
In [ ]: matrix_histogram_complex(U)
```

```
In [ ]: matrix_histogram_complex(result.U_f)
```

```
In [ ]: hinton(U)
```

```
In [ ]: hinton(result.U_f)
```

```
In [ ]: updater(result.u[-1, 0, : ] , (2*pi)/500, epsilon=((0.1*2*pi)/(10**3)))
```

```
In [ ]: times[-1]
```

```
In [ ]: total_time_evo
```

2.6 joining qutip to my code

2.7 ##### total_time

NameError Traceback (most recent call last) in () ----> 1 total_time
NameError: name 'total_time' is not defined

```
In [ ]: len(times)
```

def terminator(max_iter, time_steps=len(times), total_time= total_time_evo, epsilon= 2*pi*1): r"""Brief description of the function"""

```
xi_initial = result.u[-1, 0, : ]
#1000*random_sample((time_steps,))
dt = (2*pi)/500  #total_time/time_steps
xi_diff, xi_new_vec = updater(xi_initial, dt, epsilon)

for i in range(max_iter):
    if amax(xi_diff) < epsilon**2 :
        xi_final = xi_new_vec
```

```
        break
    else :
        xi_diff, xi_new_vec = updater(xi_new_vec, dt, epsilon)
        print(i)
        print(amax(xi_diff))


xi_final = xi_new_vec
return xi_final
```

def terminator(max_iter, time_steps=len(times), total_time=total_time_evo, epsilon=2*pi*1): r"""Brief description of the function""" xi_initial = result.u[-1, 0, :] # 1000*random_sample((time_steps,))* dt = (2pi)/500 #total_time/time_steps xi_diff, xi_new_vec = updater(xi_initial, dt, epsilon)

```
for i in range(max_iter):
    if amax(xi_diff) < epsilon**2 :
        xi_final = xi_new_vec
        print("Tejas is unlucky")
        break
    else :
        xi_diff, xi_new_vec = updater(xi_new_vec, dt, epsilon)
        print("Tejas is a good boy")
        print(i)
        print(amax(xi_diff))

return xi_final
```

```
In [ ]: def terminator(max_iter, time_steps=len(times),
                       total_time=total_time_evo,
                       epsilon=2*pi*1):
            r"""Brief description of the function"""
            xi_initial = result.u[-1, 0, :]
            # 1000*random_sample((time_steps,))
            dt = (2*pi)/500 #total_time/time_steps
            xi_diff, xi_new_vec = updater(xi_initial, dt, epsilon)
            min_iter = int(max_iter/2)
            for i in range(max_iter):
                if i == 0:
                    print("Hi")


                if i > min_iter :

                    if i > min_iter + 1 :
                        print("Surpassed minimum iteration barrier")
                    if amax(xi_diff) < epsilon**2 :
                        xi_final = xi_new_vec
```

```
                    #print("Tejas is unlucky")
                    print("Attempted iterations ", i)
                    break
                else :
                    xi_diff, xi_new_vec = updater(xi_new_vec, dt, epsilon)
                    #print("Tejas is a good boy")
                    print(i)
                    print(amax(xi_diff))

            else :
                #print("Normal life")
                xi_diff, xi_new_vec = updater(xi_new_vec, dt, epsilon)
                print(i)
                print(amax(xi_diff))

        return xi_final
```

2.8 sub topic 3

```
In [ ]: new_label = [r'$g_{with diss}$']
```

2.8.1 try

```
In [ ]: xi_opt = terminator(10)
```

```
In [ ]: time_steps=len(times)
        total_time= total_time_evo
        epsilon= 2*pi*1
```

```
In [ ]: dt = (2*pi)/500
        F(xi_opt, dt)
```

```
In [ ]: L_full_maker(xi_opt, dt)
```

```
In [ ]: ax = axes()
        ax.plot(times, xi_opt)
        xi_max = amax(xi_opt) + 0.1
        xi_min = amin(xi_opt) - 0.1

        ax.set(xlim=(0, total_time_evo), ylim=(xi_min, xi_max),
               xlabel='time', ylabel= r'$g_{with diss}$',
               title='Control field in the presence of dissipation');
```

```
In [284]: def cF2p(Q_G, tejas_c):
              r"""Takes in the 2 control fields and produces the plot directly.
              The plot contains the following lines:
              1. Qutip grape without dissipation
              2. Qutip grape with dissipation
              3. My  control field with dissipation
```

```
Does some stuff.

Parameters
----------
Q_G      : numpy array?
           control field Qutip grape without dissipation

tejas_c : numpy array?
           My  control field with dissipation for open quantum system


Returns
-------
inFidelity vector for
1. Qutip grape without dissipation    : infid_grape_no_diss
2. Qutip grape with dissipation       : infid_grape_diss
3. My  control field with dissipation : infid_grape_diss_optimized


"""
qone = basis(2, 0)
qzero = basis(2, 1)
c_ops_tejas = sqrt(gamma)*Lin
# state list building
H_qutip = [H_0, [H_1, Q_G]] #result.u[-1, 0, :]
print("grape_no_diss")
grape_no_diss = mesolve(H_qutip, qzero, times, c_ops=[],
                        e_ops=[], args={}, options=None,
                        progress_bar=TextProgressBar() )
print("")
print("grape_diss")
grape_diss = mesolve(H_qutip, qzero, times, c_ops=[c_ops_tejas],
                     e_ops=[], args={}, options=None,
                     progress_bar=TextProgressBar() )
print("")
print("grape_diss_optimized")
H_diss_optimized = [H_0, [H_1, tejas_c ]]
grape_diss_optimized = mesolve(H_diss_optimized, qzero, times,
                               c_ops=[c_ops_tejas], e_ops=[],
                               args={}, options=None,
                               progress_bar=TextProgressBar() )
# H_diss_optimized should have been written isntead of H_no_diss
print("")
print("")

# states list to fidelity list
one_dm = ket2dm(qone)
zero_dm = ket2dm(qzero)
```

```python
    def infidelity_tejas(dm, d=2):
        r"""Brief description of the function"""
        infid = 1 - fidelity(one_dm, dm)
        return infid

    infid_dm_vec = vectorize(infidelity_tejas)
    infid_grape_no_diss = infid_dm_vec(grape_no_diss.states)
    infid_grape_diss = infid_dm_vec(grape_diss.states)
    infid_grape_diss_optimized = infid_dm_vec(grape_diss_optimized.states)
    list_of_infid_vec = [infid_grape_no_diss, infid_grape_diss,
                         infid_grape_diss_optimized]
    return list_of_infid_vec
```

```python
In [285]: list_of_infid_vec = cF2p(result.u[-1, 0, :], xi_opt)
```

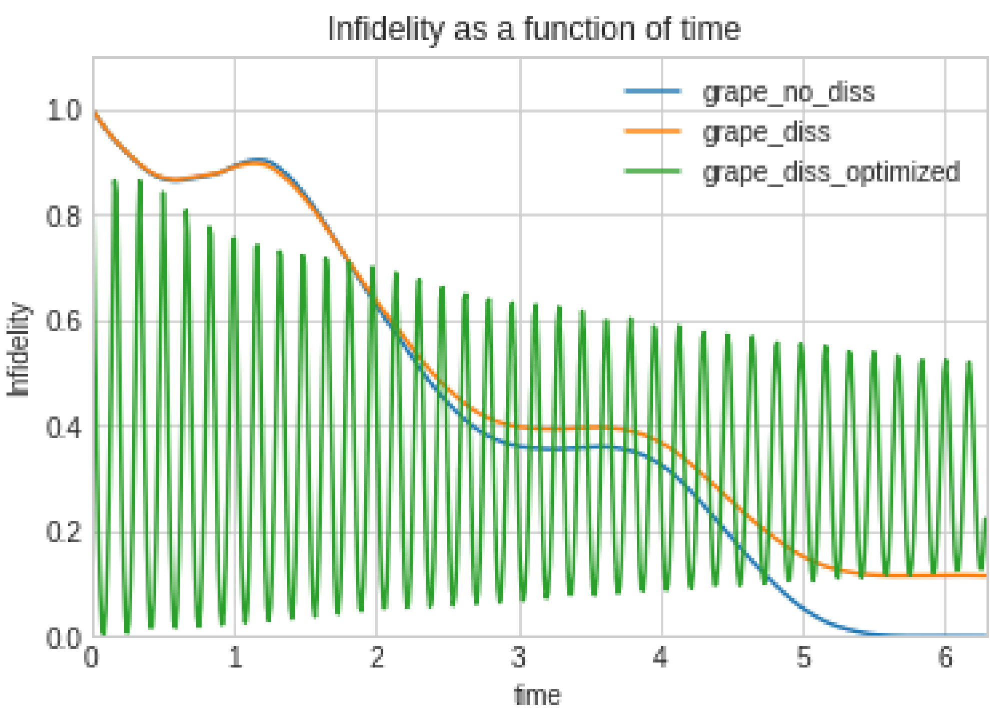
```
grape_no_diss
10.0%. Run time:   0.02s. Est. time left: 00:00:00:00
20.0%. Run time:   0.05s. Est. time left: 00:00:00:00
30.0%. Run time:   0.08s. Est. time left: 00:00:00:00
40.0%. Run time:   0.10s. Est. time left: 00:00:00:00
50.0%. Run time:   0.14s. Est. time left: 00:00:00:00
60.0%. Run time:   0.18s. Est. time left: 00:00:00:00
70.0%. Run time:   0.22s. Est. time left: 00:00:00:00
80.0%. Run time:   0.26s. Est. time left: 00:00:00:00
90.0%. Run time:   0.31s. Est. time left: 00:00:00:00
Total run time:   0.34s

grape_diss
10.0%. Run time:   0.01s. Est. time left: 00:00:00:00
20.0%. Run time:   0.02s. Est. time left: 00:00:00:00
30.0%. Run time:   0.04s. Est. time left: 00:00:00:00
40.0%. Run time:   0.05s. Est. time left: 00:00:00:00
50.0%. Run time:   0.07s. Est. time left: 00:00:00:00
60.0%. Run time:   0.08s. Est. time left: 00:00:00:00
70.0%. Run time:   0.10s. Est. time left: 00:00:00:00
80.0%. Run time:   0.13s. Est. time left: 00:00:00:00
90.0%. Run time:   0.15s. Est. time left: 00:00:00:00
Total run time:   0.17s

grape_diss_optimized
10.0%. Run time:   0.02s. Est. time left: 00:00:00:00
20.0%. Run time:   0.05s. Est. time left: 00:00:00:00
30.0%. Run time:   0.08s. Est. time left: 00:00:00:00
40.0%. Run time:   0.11s. Est. time left: 00:00:00:00
50.0%. Run time:   0.15s. Est. time left: 00:00:00:00
60.0%. Run time:   0.18s. Est. time left: 00:00:00:00
70.0%. Run time:   0.22s. Est. time left: 00:00:00:00
80.0%. Run time:   0.25s. Est. time left: 00:00:00:00
```

```
90.0%. Run time:   0.30s. Est. time left: 00:00:00:00
Total run time:   0.35s
```

In [286]:
```python
ax = axes()
ax.plot(times, list_of_infid_vec[0], label='grape_no_diss')
ax.plot(times, list_of_infid_vec[1], label='grape_diss')
ax.plot(times, list_of_infid_vec[2], label='grape_diss_optimized')
#ax.plot(x, cos(x), ':b', label='cos(x)')'-g',
#ax.axis('equal')
ax.legend()
ax.set(xlim=(times[0],times[-1]), ylim=(0, 1.1),
       xlabel='time', ylabel='Infidelity',
       title='Infidelity as a function of time ');
```



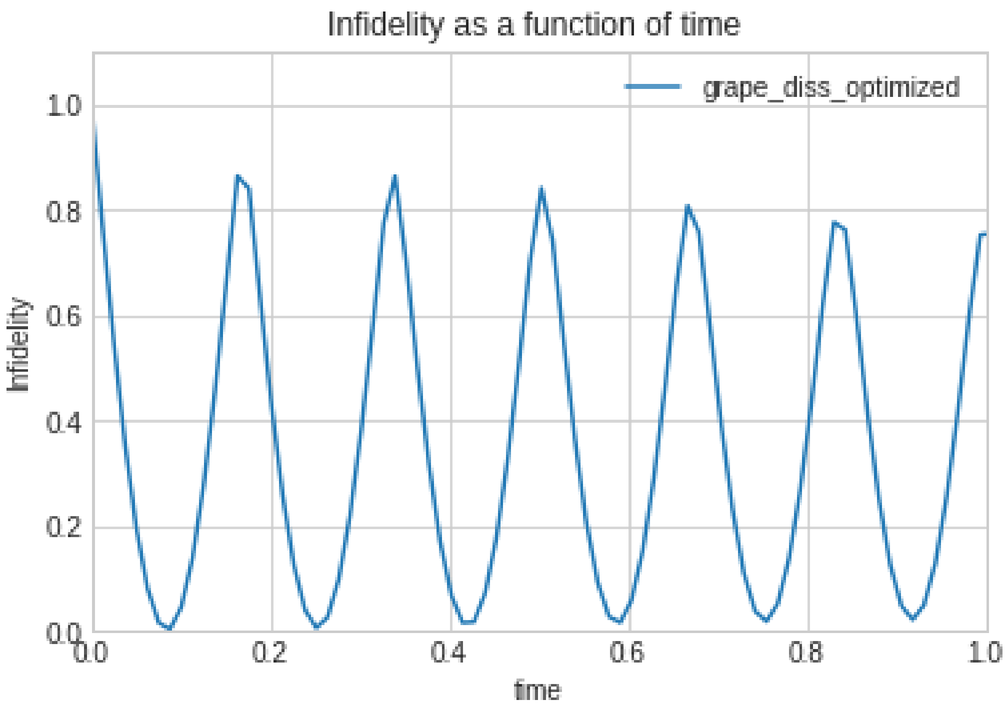
In [287]:
```python
list_of_infid_vec[2][0:10]
```

Out[287]:
```
array([1.        , 0.76726177, 0.54871542, 0.35498059, 0.19641811,
       0.08146654, 0.01624441, 0.00419602, 0.04598029, 0.13924153])
```

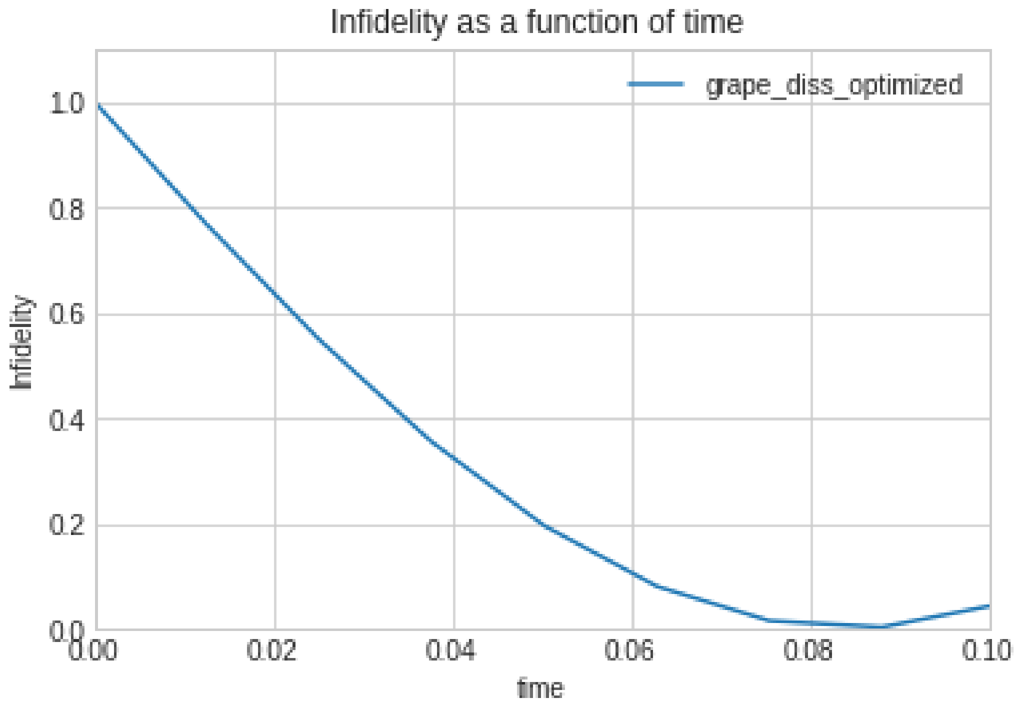
In [288]:
```python
len(list_of_infid_vec[2])
```

Out[288]:
```
500
```

In [289]:
```
ax = axes()
#ax.plot(times, list_of_infid_vec[0], label='grape_no_diss')
#ax.plot(times, list_of_infid_vec[1], label='grape_diss')
ax.plot(times, list_of_infid_vec[2], label='grape_diss_optimized')
#ax.plot(x, cos(x), ':b', label='cos(x)')'-g',
#ax.axis('equal')
ax.legend()
ax.set(xlim=(times[0], 1), ylim=(0, 1.1),
       xlabel='time', ylabel='Infidelity',
       title='Infidelity as a function of time ');
```



In [290]:
```
ax = axes()
#ax.plot(times, list_of_infid_vec[0], label='grape_no_diss')
#ax.plot(times, list_of_infid_vec[1], label='grape_diss')
ax.plot(times, list_of_infid_vec[2], label='grape_diss_optimized')
#ax.plot(x, cos(x), ':b', label='cos(x)')'-g',
#ax.axis('equal')
ax.legend()
ax.set(xlim=(times[0], 0.1), ylim=(0, 1.1),
       xlabel='time', ylabel='Infidelity',
       title='Infidelity as a function of time ');
```




```
In [291]: list_of_infid_vec[2][0:10]

Out[291]: array([1.        , 0.76726177, 0.54871542, 0.35498059, 0.19641811,
                 0.08146654, 0.01624441, 0.00419602, 0.04598029, 0.13924153])

In [298]: for n in range(len(times)) :
              print("")
```

```
In [295]: the_ones = [n for n in range(times[-1]) ]
```

```
    ---------------------------------------------------------------------------

    TypeError                                 Traceback (most recent call last)

    <ipython-input-295-a3044dec3033> in <module>()
----> 1 the_ones = [n for n in range(times[-1]) ]


    TypeError: 'numpy.float64' object cannot be interpreted as an integer
```

```
In [299]: the_chosen_ones = [n for n in range(times[-1])
                             if abs(list_of_infid_vec[2][n] < 0.001)]
```

```
    ---------------------------------------------------------------------------

    TypeError                                 Traceback (most recent call last)

    <ipython-input-299-0e1b2f1921ae> in <module>()
----> 1 the_chosen_ones = [n for n in range(times[-1])
      2                              if abs(list_of_infid_vec[2][n] < 0.001)]


    TypeError: 'numpy.float64' object cannot be interpreted as an integer
```

```
In [294]: abs(list_of_infid_vec[2][3] < 0.001)

Out[294]: False

In [300]: the_chosen_ones = [n for n in range(len(times))
                             if abs(list_of_infid_vec[2][n] < 0.001)]

In [301]: the_chosen_ones

Out[301]: []

In [302]: the_chosen_ones = [n for n in range(len(times))
                             if abs(list_of_infid_vec[2][n]) < 0.001 ]
```

In [303]: the_chosen_ones

Out[303]: []

```
In [304]: the_chosen_ones = [n for n in range(len(times))
                             if abs(list_of_infid_vec[2][n]) > 0.001 ]
```

In [305]: len(the_chosen_ones)

Out[305]: 500

In [306]: amin(list_of_infid_vec[2])

Out[306]: 0.004196015277072251

```
In [307]: the_chosen_ones = [n for n in range(len(times))
                             if abs(list_of_infid_vec[2][n])
                              < (10**(-4) + amin(list_of_infid_vec[2]))  ]
```

In [308]: the_chosen_ones

Out[308]: [7]

In [309]: list_of_infid_vec[2][7]

Out[309]: 0.004196015277072251

2.8.2 try

```
In [ ]: xi_opt = terminator(1000)
```

```
In [ ]: time_steps=len(times)
        total_time= total_time_evo
        epsilon= 2*pi*1
```

```
In [ ]: dt = (2*pi)/500
        F(xi_opt, dt)
```

```
In [ ]: L_full_maker(xi_opt, dt)
```

```
In [ ]: ax = axes()
        ax.plot(times, xi_opt)
        xi_max = amax(xi_opt) + 0.1
        xi_min = amin(xi_opt) - 0.1

        ax.set(xlim=(0, total_time_evo), ylim=(xi_min, xi_max),
               xlabel='time', ylabel= r'$g_{with diss}$',
               title='Control field in the presence of dissipation');
```

```
In [ ]: list_of_infid_vec = cF2p(result.u[-1, 0, :], xi_opt)
```

In [ ]:
```
ax = axes()
ax.plot(times, list_of_infid_vec[0], label='grape_no_diss')
ax.plot(times, list_of_infid_vec[1], label='grape_diss')
ax.plot(times, list_of_infid_vec[2], label='grape_diss_optimized')
#ax.plot(x, cos(x), ':b', label='cos(x)')'-g',
#ax.axis('equal')
ax.legend()
ax.set(xlim=(times[0],times[-1]), ylim=(0, 1.1),
       xlabel='time', ylabel='Infidelity',
       title='Infidelity as a function of time ');
```

2.8.3 try

In [ ]:
```
xi_opt = terminator(1000,time_steps=len(times), total_time= total_time_evo,
                    epsilon= ((0.1*2*pi)/(times[-1])))
```

In [ ]:
```
time_steps=len(times)
total_time= total_time_evo
epsilon = ((0.1*2*pi)/(times[-1]))
```

In [ ]:
```
dt = (2*pi)/500
F(xi_opt, dt)
```

In [ ]:
```
L_full_maker(xi_opt, dt)
```

In [ ]:
```
ax = axes()
ax.plot(times, xi_opt)
xi_max = amax(xi_opt) + 0.1
xi_min = amin(xi_opt) - 0.1

ax.set(xlim=(0, total_time_evo), ylim=(xi_min, xi_max),
       xlabel='time', ylabel= r'$g_{with diss}$',
       title='Control field in the presence of dissipation');
```

In [ ]:
```
list_of_infid_vec = cF2p(result.u[-1, 0, :], xi_opt)
```

In [ ]:
```
ax = axes()
ax.plot(times, list_of_infid_vec[0], label='grape_no_diss')
ax.plot(times, list_of_infid_vec[1], label='grape_diss')
ax.plot(times, list_of_infid_vec[2], label='grape_diss_optimized')
#ax.plot(x, cos(x), ':b', label='cos(x)')'-g',
#ax.axis('equal')
ax.legend()
ax.set(xlim=(times[0],times[-1]), ylim=(0, 1.1),
       xlabel='time', ylabel='Infidelity',
       title='Infidelity as a function of time ');
```

2.8.4 try

```
In [ ]: xi_opt = terminator(10,time_steps=len(times), total_time= total_time_evo,
                      epsilon= ((0.1*2*pi)/(times[-1])))
```

```
In [ ]: time_steps=len(times)
        total_time= total_time_evo
        epsilon = ((0.1*2*pi)/(times[-1]))
```

```
In [ ]: dt = (2*pi)/500
        F(xi_opt, dt)
```

```
In [ ]: L_full_maker(xi_opt, dt)
```

```
In [ ]: ax = axes()
        ax.plot(times, xi_opt)
        xi_max = amax(xi_opt) + 0.1
        xi_min = amin(xi_opt) - 0.1

        ax.set(xlim=(0, total_time_evo), ylim=(xi_min, xi_max),
               xlabel='time', ylabel= r'$g_{with diss}$',
               title='Control field in the presence of dissipation');
```

```
In [ ]: list_of_infid_vec = cF2p(result.u[-1, 0, :], xi_opt)
```

```
In [ ]: ax = axes()
        ax.plot(times, list_of_infid_vec[0], label='grape_no_diss')
        ax.plot(times, list_of_infid_vec[1], label='grape_diss')
        ax.plot(times, list_of_infid_vec[2], label='grape_diss_optimized')
        #ax.plot(x, cos(x), ':b', label='cos(x)')'-g',
        #ax.axis('equal')
        ax.legend()
        ax.set(xlim=(times[0],times[-1]), ylim=(0, 1.1),
               xlabel='time', ylabel='Infidelity',
               title='Infidelity as a function of time ');
```

2.8.5 try

```
In [ ]: xi_opt = terminator(1000,time_steps=len(times), total_time= total_time_evo,
                      epsilon= ((0.1*2*pi)/(10**3)))
```

```
In [ ]: time_steps=len(times)
        total_time= total_time_evo
        epsilon = ((0.1*2*pi)/(times[-1]))
```

```
In [ ]: dt = (2*pi)/500
        F(xi_opt, dt)
```

```
In [ ]: F(result.u[-1, 0, : ], dt)
```

```
In [ ]: L_full_maker(xi_opt, dt)
```

```
In [ ]: len(times)
```

```
In [ ]: new_label = [r'$g_{with diss}$']
```

"'plot_grape_control_fields(times, xi_opt / (2 * np.pi), new_label, uniform_axes=True); ValueError Traceback (most recent call last) in () 1 plot_grape_control_fields(times, ----> 2 xi_opt / (2 * np.pi), new_label, uniform_axes=True);

/anaconda3/envs/qutip-env/lib/python3.6/site-packages/qutip/control/grape.py in plot_grape_control_fields(times, u, labels, uniform_axes) 101 import matplotlib.pyplot as plt 102 --> 103 R, J, M = u.shape 104 105 fig, axes = plt.subplots(J, 1, figsize=(8, 2 * J), squeeze=False)

ValueError: not enough values to unpack (expected 3, got 1) "'

```
In [ ]: ax = axes()
        ax.plot(times, xi_opt)
        ax.set(xlim=(0, total_time_evo), ylim=(-0.8,0.8 ),
               xlabel='time', ylabel= r'$g_{with diss}$',
               title='Control field in the presence of dissipation');
```

```
In [ ]: list_of_infid_vec = cF2p(result.u[-1, 0, :], xi_opt)
```

```
In [ ]: ax = axes()
        ax.plot(times, list_of_infid_vec[0], label='grape_no_diss')
        ax.plot(times, list_of_infid_vec[1], label='grape_diss')
        ax.plot(times, list_of_infid_vec[2], label='grape_diss_optimized')
        #ax.plot(x, cos(x), ':b', label='cos(x)')'-g',
        #ax.axis('equal')
        ax.legend()
        ax.set(xlim=(times[0],times[-1]), ylim=(0, 1.1),
               xlabel='time', ylabel='Infidelity',
               title='Infidelity as a function of time ');
```

```
In [ ]: ax = axes()
        ax.plot(times, xi_opt)
        xi_max = amax(xi_opt) + 0.1
        xi_min = amin(xi_opt) - 0.1

        ax.set(xlim=(0, total_time_evo), ylim=(xi_min, xi_max),
               xlabel='time', ylabel= r'$g_{with diss}$',
               title='Control field in the presence of dissipation');
```

```
In [ ]: ax = axes()
        ax.plot(times, result.u[-1, 0, : ])
        xi_max = amax(result.u[-1, 0, : ]) + 0.1
        xi_min = amin(result.u[-1, 0, : ]) - 0.1

        ax.set(xlim=(0, total_time_evo), ylim=(xi_min, xi_max),
               xlabel='time', ylabel= r'$g_{with diss}$',
               title='old Control field in the presence of \
                      dissipation produced by qutip');
```

2.8.6 try

```
In [ ]: xi_opt = terminator(1000,time_steps=len(times), total_time= total_time_evo,
                            epsilon= ((0.1*2*pi)/(10**4)))
```

```
In [ ]: dt = (2*pi)/500
        F(xi_opt, dt)
```

```
In [ ]: L_full_maker(xi_opt, dt)
```

```
In [ ]: ax = axes()
        ax.plot(times, xi_opt)
        xi_max = amax(xi_opt) + 0.1
        xi_min = amin(xi_opt) - 0.1

        ax.set(xlim=(0, total_time_evo), ylim=(xi_min, xi_max),
               xlabel='time', ylabel= r'$g_{with diss}$',
               title='Control field in the presence of dissipation');
```

```
In [ ]: list_of_infid_vec = cF2p(result.u[-1, 0, :], xi_opt)
```

```
In [ ]: ax = axes()
        ax.plot(times, list_of_infid_vec[0], label='grape_no_diss')
        ax.plot(times, list_of_infid_vec[1], label='grape_diss')
        ax.plot(times, list_of_infid_vec[2], label='grape_diss_optimized')
        #ax.plot(x, cos(x), ':b', label='cos(x)')'-g',
        #ax.axis('equal')
        ax.legend()
        ax.set(xlim=(times[0],times[-1]), ylim=(0, 1.1),
               xlabel='time', ylabel='Infidelity',
               title='Infidelity as a function of time ');
```

2.8.7 try

```
In [ ]: xi_opt = terminator(10**4,time_steps=len(times), total_time= total_time_evo,
                            epsilon= ((0.1*2*pi)/(10**3)))
```

```
In [ ]: dt = (2*pi)/500
        F(xi_opt, dt)
```

```
In [ ]: L_full_maker(xi_opt, dt)
```

```
In [ ]: ax = axes()
        ax.plot(times, xi_opt)
        xi_max = amax(xi_opt) + 0.1
        xi_min = amin(xi_opt) - 0.1

        ax.set(xlim=(0, total_time_evo), ylim=(xi_min, xi_max),
               xlabel='time', ylabel= r'$g_{with diss}$',
               title='Control field in the presence of dissipation');
```

```
In [ ]: list_of_infid_vec = cF2p(result.u[-1, 0, :], xi_opt)
```

```
In [ ]: ax = axes()
        ax.plot(times, list_of_infid_vec[0], label='grape_no_diss')
        ax.plot(times, list_of_infid_vec[1], label='grape_diss')
        ax.plot(times, list_of_infid_vec[2], label='grape_diss_optimized')
        #ax.plot(x, cos(x), ':b', label='cos(x)')'-g',
        #ax.axis('equal')
        ax.legend()
        ax.set(xlim=(times[0],times[-1]), ylim=(0, 1.1),
               xlabel='time', ylabel='Infidelity',
               title='Infidelity as a function of time ');
```

2.8.8 try

```
In [ ]: xi_opt = terminator(10**4,time_steps=len(times), total_time= total_time_evo,
                             epsilon= ((0.1*2*pi)/(10**4)))
```

```
In [ ]: dt = (2*pi)/500
        F(xi_opt, dt)
```

```
In [ ]: L_full_maker(xi_opt, dt)
```

```
In [ ]: ax = axes()
        ax.plot(times, xi_opt)
        xi_max = amax(xi_opt) + 0.1
        xi_min = amin(xi_opt) - 0.1

        ax.set(xlim=(0, total_time_evo), ylim=(xi_min, xi_max),
               xlabel='time', ylabel= r'$g_{with diss}$',
               title='Control field in the presence of dissipation');
```

```
In [ ]: list_of_infid_vec = cF2p(result.u[-1, 0, :], xi_opt)
```

```
In [ ]: ax = axes()
        ax.plot(times, list_of_infid_vec[0], label='grape_no_diss')
        ax.plot(times, list_of_infid_vec[1], label='grape_diss')
        ax.plot(times, list_of_infid_vec[2], label='grape_diss_optimized')
        #ax.plot(x, cos(x), ':b', label='cos(x)')'-g',
        #ax.axis('equal')
        ax.legend()
        ax.set(xlim=(times[0],times[-1]), ylim=(0, 1.1),
               xlabel='time', ylabel='Infidelity',
               title='Infidelity as a function of time ');
```

2.8.9 try

```
In [ ]: xi_opt = terminator(10**4,time_steps=len(times), total_time= total_time_evo,
                             epsilon= ((0.1*2*pi)/(10**10)))
```

```
In [ ]: dt = (2*pi)/500
        F(xi_opt, dt)
```

```
In [ ]: L_full_maker(xi_opt, dt)
```

```
In [ ]: ax = axes()
        ax.plot(times, xi_opt)
        xi_max = amax(xi_opt) + 0.1
        xi_min = amin(xi_opt) - 0.1

        ax.set(xlim=(0, total_time_evo), ylim=(xi_min, xi_max),
               xlabel='time', ylabel= r'$g_{with diss}$',
               title='Control field in the presence of dissipation');
```

```
In [ ]: list_of_infid_vec = cF2p(result.u[-1, 0, :], xi_opt)
```

```
In [ ]: ax = axes()
        ax.plot(times, list_of_infid_vec[0], label='grape_no_diss')
        ax.plot(times, list_of_infid_vec[1], label='grape_diss')
        ax.plot(times, list_of_infid_vec[2], label='grape_diss_optimized')
        #ax.plot(x, cos(x), ':b', label='cos(x)')'-g',
        #ax.axis('equal')
        ax.legend()
        ax.set(xlim=(times[0],times[-1]), ylim=(0, 1.1),
               xlabel='time', ylabel='Infidelity',
               title='Infidelity as a function of time ');
```

2.8.10 try

```
In [ ]: xi_opt = terminator(10**4,time_steps=10**3, total_time= total_time_evo,
                           epsilon= ((0.1*2*pi)/(10**3)))
```

```
In [ ]: dt = (2*pi)/(10**3)#(2*pi)/500
        # probaly happened because of hard coding of dt inside terminator
        # must try it again after changing that
        F(xi_opt, dt)
```

```
In [ ]: L_full_maker(xi_opt, dt)
```

```
In [ ]: ax = axes()
        ax.plot(times, xi_opt)
        xi_max = amax(xi_opt) + 0.1
        xi_min = amin(xi_opt) - 0.1

        ax.set(xlim=(0, total_time_evo), ylim=(xi_min, xi_max),
               xlabel='time', ylabel= r'$g_{with diss}$',
               title='Control field in the presence of dissipation');
```

```
In [ ]: list_of_infid_vec = cF2p(result.u[-1, 0, :], xi_opt)
```

```
In [ ]: ax = axes()
        ax.plot(times, list_of_infid_vec[0], label='grape_no_diss')
        ax.plot(times, list_of_infid_vec[1], label='grape_diss')
        ax.plot(times, list_of_infid_vec[2], label='grape_diss_optimized')
        #ax.plot(x, cos(x), ':b', label='cos(x)')'-g',
        #ax.axis('equal')
        ax.legend()
        ax.set(xlim=(times[0],times[-1]), ylim=(0, 1.1),
               xlabel='time', ylabel='Infidelity',
               title='Infidelity as a function of time ');
```

2.9 APS stuff

2.9.1 infidelity function tryouts

```
In [ ]: qone = basis(2, 0)
        qone
```

```
In [ ]: qzero = basis(2, 1)
        qzero
```

2.10 ###### ket2bra

NameError Traceback (most recent call last) in () ----> 1 ket2bra
NameError: name 'ket2bra' is not defined

```
In [ ]: (sigmax()).matrix_element(qone, qzero)
```

```
In [ ]: abs((sigmax()).matrix_element(qone, qzero))
```

```
In [ ]: abs(2 + 3j)
```

```
In [ ]: abs(3 + 4j)
```

```
In [ ]: def infidelity_to_one(U):
            r"""infidelity to state one"""
            fidelity = (abs(U.matrix_element(qone, qzero)))**2
            infidelity = 1 - fidelity
            return infidelity
```

```
In [ ]: infidelity_to_one(sigmax())
        # 1 - abs((sigmax()).matrix_element(qone, qzero))
        # 1 - 1 = 0
```

result.u, result.u but lindbladian evolution, xi_opt

```
In [ ]: qone.overlap(qzero)
```

```
In [ ]: qzero.overlap(qzero)
```

```
In [ ]: qone.overlap(qone)
```

```
In [ ]: def infidelity_to_state(psi):
            r"""infidelity to state one"""
            fidelity = (abs(qone.overlap(psi)))**2
            infidelity = 1 - fidelity
            return infidelity
```

```
In [ ]: infidelity_to_state(qzero)
```

```
In [ ]: infidelity_to_state(qone)
```

```
In [ ]: #mesolve
```

2.10.1 state building

```
In [ ]: H_no_diss = [H_0, [H_1, result.u[-1, 0, :] ]]
        grape_no_diss = mesolve(H_no_diss, qzero, times, c_ops=[], e_ops=[],
                                args={}, options=None,
                                progress_bar=EnhancedTextProgressBar() )
```

```
In [ ]: len(grape_no_diss.states)
```

```
In [ ]: Lin
```

```
In [ ]: sqrt(gamma)
```

```
In [ ]: c_ops_tejas = sqrt(gamma)*Lin
        c_ops_tejas
```

```
In [ ]: H_diss = [H_0, [H_1, result.u[-1, 0, :] ]]
        grape_diss = mesolve(H_no_diss, qzero, times, c_ops=[c_ops_tejas], e_ops=[],
                                args={}, options=None,
                                progress_bar=EnhancedTextProgressBar() )
```

```
In [ ]: len(grape_diss.states)
```

```
In [ ]: H_diss_optimized = [H_0, [H_1, xi_opt ]]
        grape_diss_optimized = mesolve(H_diss_optimized, qzero, times, c_ops=[c_ops_tejas],
                                       e_ops=[], args={}, options=None,
                                       progress_bar=TextProgressBar() )
        # H_diss_optimized should have been written isntead of H_no_diss
```

```
In [ ]: len(grape_diss_optimized.states)
```

2.10.2 states list to fidelity list

```
In [ ]: infidelity_to_state_vec = vectorize(infidelity_to_state)
```

```
In [ ]: infid_grape_no_diss = infidelity_to_state_vec(grape_no_diss.states)
```

```
In [ ]: '''fid_grape_diss = infidelity_to_state_vec(grape_diss.states)
        TypeError                                 Traceback (most recent call last)
        <ipython-input-101-fbabc06eff84> in <module>()
        ----> 1 fid_grape_diss = infidelity_to_state_vec(grape_diss.states)

        /anaconda3/envs/qutip-env/lib/python3.6/site-packages/numpy/lib/function_base.py in __
           2753             vargs.extend([kwargs[_n] for _n in names])
           2754
        -> 2755         return self._vectorize_call(func=func, args=vargs)
           2756
           2757     def _get_ufunc_and_otypes(self, func, args):

        /anaconda3/envs/qutip-env/lib/python3.6/site-packages/numpy/lib/function_base.py in _v
           2823             res = func()
           2824         else:
        -> 2825             ufunc, otypes = self._get_ufunc_and_otypes(func=func, args=args)
           2826
           2827             # Convert args to object arrays first

        /anaconda3/envs/qutip-env/lib/python3.6/site-packages/numpy/lib/function_base.py in _g
           2783
           2784             inputs = [arg.flat[0] for arg in args]
        -> 2785             outputs = func(*inputs)
           2786
           2787             # Performance note: profiling indicates that -- for simple

        <ipython-input-96-32321f8d8a33> in infidelity_to_state(psi)
              1 def infidelity_to_state(psi):
              2     r"""infidelity to state one"""
        ----> 3     fidelity = (abs(qone.overlap(psi)))**2
              4     infidelity = 1 - fidelity
              5     return infidelity

        /anaconda3/envs/qutip-env/lib/python3.6/site-packages/qutip/qobj.py in overlap(self, s
           1486                     return (self.data.H * state.data)[0, 0]
           1487
        -> 1488         raise TypeError("Can only calculate overlap for state vector Qobjs")
           1489
           1490     def eigenstates(self, sparse=False, sort='low',

        TypeError: Can only calculate overlap for state vector Qobjs


        '''
```

fid_grape_no_diss = infidelity_to_state_vec(grape_no_diss.states)

```
In [ ]: fidelity(sigmax(), sigmax())
```

```
In [ ]: zero_dm = ket2dm(qzero)
        zero_dm
```

```
In [ ]: one_dm = ket2dm(qone)
        one_dm
```

```
In [ ]: fidelity(zero_dm, zero_dm)
```

```
In [ ]: fidelity(zero_dm, one_dm)
```

```
In [ ]: fidelity(one_dm, zero_dm)
```

```
In [ ]: grape_no_diss.states[5]
```

```
In [ ]: grape_diss.states[5]
```

```
In [ ]: grape_diss.states[7]
```

```
In [ ]: fidelity(grape_diss.states[7], one_dm)
```

```
In [ ]: #0.0122.347E10**05j)
```

```
In [ ]: def infidelity_tejas(dm, d=2):
            r"""Brief description of the function"""
            infid = 1 - fidelity(one_dm, dm)
            return infid
```

```
In [ ]: infid_dm_vec = vectorize(infidelity_tejas)
```

```
In [ ]: infid_grape_diss = infid_dm_vec(grape_diss.states)
```

```
In [ ]: len(infid_grape_diss)
```

```
In [ ]: infid_grape_diss_optimized = infid_dm_vec(grape_diss_optimized.states)
```

```
In [ ]: len(infid_grape_diss_optimized)
```

2.10.3 plots

```
In [ ]: ax = axes()
        ax.plot(times, infid_grape_no_diss, label='grape_no_diss')
        ax.plot(times, infid_grape_diss, label='grape_diss')
        ax.plot(times, infid_grape_diss_optimized, label='grape_diss_optimized')
        #ax.plot(x, cos(x), ':b', label='cos(x)')'-g',
        #ax.axis('equal')
        ax.legend()
        ax.set(xlim=(times[0],times[-1]), ylim=(0, 1.1),
               xlabel='time', ylabel='Infidelity',
               title='Infidelity as a function of time ');
```

```
In [ ]: ax = axes()
        ax.plot(times, infid_grape_no_diss, label='grape_no_diss')
        ax.plot(times, infid_grape_diss, label='grape_diss')
        #ax.plot(times, infid_grape_diss_optimized, label='grape_diss_optimized')
        #ax.plot(x, cos(x), ':b', label='cos(x)')'-g',
        #ax.axis('equal')
        ax.legend()

        ax.set(xlim=(times[0],times[-1]), ylim=(0, 1.1),
               xlabel='time', ylabel='Infidelity',
               title='Infidelity as a function of time ');
```

```
In [ ]: ax = axes()
        ax.plot(times, infid_grape_no_diss, label='grape_no_diss')
        #ax.plot(times, infid_grape_diss, label='grape_diss')
        ax.plot(times, infid_grape_diss_optimized, label='grape_diss_optimized')
        #ax.plot(x, cos(x), ':b', label='cos(x)')'-g',
        #ax.axis('equal')
        ax.legend()

        ax.set(xlim=(times[0],times[-1]), ylim=(0, 1.1),
               xlabel='time', ylabel='Infidelity',
               title='Infidelity as a function of time ');
```

```
In [ ]: ax = axes()
        c_diff = result.u[-1, 0, : ] - xi_opt
        ax.plot(times, c_diff)
        c_max = amax(c_diff) + 0.1
        c_min = amin(c_diff) - 0.1

        ax.set(xlim=(0, total_time_evo), ylim=(c_min, c_max),
               xlabel='time', ylabel= 'diff',
               title='difference');
```

```
In [ ]: ax = axes()
        ax.plot(times, xi_opt)
        xi_max = amax(xi_opt) + 0.1
        xi_min = amin(xi_opt) - 0.1

        ax.set(xlim=(0, total_time_evo), ylim=(xi_min, xi_max),
               xlabel='time', ylabel= r'$g_{with diss}$',
               title='Control field in the presence of dissipation');
```

2.11 Versions

```
In [ ]: from qutip.ipynbtools import version_table

        version_table()
```

```
In [ ]: cnot()
```