```
# find Neel state in eigenbasis
def findNeelstate(length,particlenumber,table):
    """find positions of Neel states in table"""
    assert length==2*particlenumber, "need exactly half filling for Neel state"
    tot = 2**length-1
    N2 = tot//3   # integer representation of the Neel state 010101...
    N1 = tot-N2   # integer representation of the other Neel state 101010...
    return findIndex(table,N1), findIndex(table,N2)


def calc_LE(v,U,Neelpos,tint,tmax,dt): # return rate starting from Neel state
    t = np.arange(tint,tmax+dt/2,dt)
    """compute time evolution from Neel state"""
    vNeel=U[Neelpos[0]].conj()     # Neel state |1010> in eigenbasis |c1 c2 ...>
    LE = np.zeros(len(t))
    evol = np.exp(-1j*t[:,np.newaxis]*v[np.newaxis])
    """ <vNeel*U|U*exp(-iEt)*U|U*Neel>"""
    LE=np.abs(np.inner(evol*vNeel[np.newaxis,:],vNeel[np.newaxis,:]))
    return LE

# Run the program for both cases
t=np.arange(args.tint,args.tmax+args.dt/2,args.dt)

# calculate part the many-particle Hamiltonian
particlenumber=args.L/2
Psi=manyPsi(particlenumber,args.L)
table = findMagnetizationStates(args.L,particlenumber)
Neelpos = findNeelstate(args.L,particlenumber,table)
MPDW = construct_MPDW(Psi,args.L,args.W)
MPH = construct_MPH(args.V,args.L,table)
vmi,Umi = np.linalg.eigh(MPH)
Umi = Umi.transpose()


for i in range(args.dat):
    Store2=0
    for samp in range(int(args.sample)):
        MPDW = construct_MPDW(Psi,args.L,args.W)
        MPHfW = MPH + MPDW
        vmf,Umf = np.linalg.eigh(MPHfW)
        Store2 += calc_LE(vmf,Umi,Neelpos,args.tint,args.tmax,args.dt)

LE2=np.squeeze(Store2/args.sample)
RR2=-2*np.log(LE2)/args.L

for item in RR2:
    print(item)
```