In [2]:
```
import numpy as np
from numpy import pi
# importing Qiskit
from qiskit import QuantumCircuit
from qiskit import ClassicalRegister, QuantumRegister
import matplotlib.pyplot as plt
```

In [3]:
```
def sim_sv(qc,text):
    """Get the statevecor of the output of the circuit qc"""
    #import libs
    from qiskit_textbook.tools import vector2latex
    from qiskit import Aer, execute
    from qiskit.quantum_info import Statevector
    #code
    sv_sim = Aer.get_backend('statevector_simulator')
    sv = execute (qc, sv_sim).result().get_statevector()
    #vector2latex(sv, pretext = "|"+ text + "\\rangle =")
    return sv
```

In [4]:
```
def qft_rotations(circuit, n):
    """Performs qft on the first n qubits in circuit (without swaps)"""
    for i in range (n):
        circuit.h(i)
        m = 1
        for j in range (i+1,n):
            m = m +1;
            circuit.cu1(-2*pi/2**(m), i, j)
```

In [5]:
```
def swap_registers(circuit, n):
    for qubit in range(n//2):
        circuit.swap(qubit, n-qubit-1)
    return circuit
```

In [6]:
```
def makeQFTgate(N):
    """returns a gate of centred QFT, swaps qubits"""
    temp = QuantumCircuit (N)
    swap_registers(temp, N)
    qft_rotations(temp,N)
    temp.x(n-1);
    QFTC = temp.to_gate()
    QFTC.name = "QFT_" + str(N)
    return QFTC
```

In [7]:
```
def phase(n, phi):
    """returns the phase transformation gate for n qubits"""
    qc = QuantumCircuit(n+1);
    for i in range (n):
        qc.u1(phi/(2**(i+n-3)), n-1-i)
    for i in range (n):
        for j in range (i+1,n):
            qc.cx(n-1-j, n)
            qc.cx(n-1-i, n)
            qc.u1(phi* (2**(2-i-j)), n) #2-i-j or 4?
            qc.cx(n-1-i, n)
            qc.cx(n-1-j, n)
    phcirc = qc.to_gate()
```

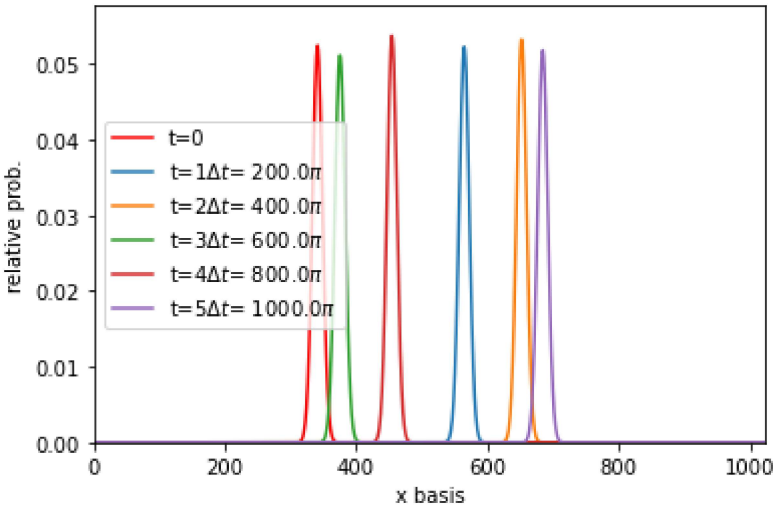
```
    phcirc.name = 'phase'+str(phi)
    return phcirc
```

In [16]:
```
def evolve (n,phi,initial_state, stepsnumber):
    """returns the state vector after evolving the initial_state for stepsnumber times"
    ## make the circuit
    n1 = n+1
    snaps = []
    m = 2**7
    #qubits
    for i in range (stepsnumber):
        q = QuantumRegister(n1)
        qc = QuantumCircuit(q)
        qc.barrier();
        #initialization
        magnitude = np.linalg.norm(initial_state)
        initial_state =initial_state/magnitude
        qc.initialize(initial_state, list(range(n)))
        qc.barrier();
        #gates
        QFT = makeQFTgate(n)
        phkinetic = phase (n,(i+1)*phi/m)
        QFTi = QFT.inverse()
        QFTi.name = "QFTi_" + str(n)
        phpotential = phase (n,((i+1)*16*phi/2/4)/m) #*16 to double the potential
        for j in range (m):
            #QFT
            qc.append(QFT,range(n))
            qc.barrier();
            #phase
            qc.append(phkinetic,range(n+1))
            qc.barrier();
            #inverse QFT
            qc.append(QFTi,list(range(n)))
            qc.barrier()
            #another phase for the harmonic potential
            qc.append(phpotential,range(n+1))
        finalstate = sim_sv(qc,'\psi')
        prob = np.square(abs(finalstate))
        snaps.append(prob)
    return snaps
```

In [17]:
```
def action ():
    """plots the the snapshots of the evolution process"""
    snaps = evolve(n,phi,initial_state, stepsnumber)
    plt.plot(np.square(abs(initial_state)),'r')
    leg = ["t=0"]
    plt.xlabel("x basis")
    plt.ylabel("relative prob.")
    for i in range (stepsnumber):
        plt.plot(snaps[i])
        leg.append("t="+ str(i+1)+ "$\Delta t$" + "= " + str(round((i+1)*phi/pi,2)) + "
    x=plt.axis([0, 2**n-1, 0, 1.1*max(np.square(abs(initial_state)))])
    plt.legend(leg)
```

In [19]:
```
##commented are descriptions of parameters, and different intitial states to play with
##parameters
n = 10; ## number of qubits
phi = 200*pi; ##time of evolution
#width= 14*2; ## 14*2 works so well centered (splitted) non moving gaussian without x g
```

```
            ## 15.2 + sqrt what works so well for coherent state oscillation with n =
            ## 15.2
width = 15.2/2; ## this is the correct width;
#initial_state = np.sqrt(np.exp(-(np.linspace(0,2**n-1, 2**n) - (2**n/2) +0.5)**2/(2*wi
## ground state of harmonic oscillator # actually not an error sqrt of gaussian is gaus

#initial_state = (np.exp(-(np.linspace(0,2**n-1, 2**n) )**2/(2*width**2))) + (np.exp(-(
## removed the sqrt what an error

#initial_state = np.sqrt(np.exp(-(np.linspace(0,2**n-1, 2**n) - (2**n/3) +0.5)**2/(2*15
# coherent state
initial_state = np.sqrt(np.exp(-(np.linspace(0,2**n-1, 2**n) - (2**n/3) +0.5)**2/(2*wid

##general energy state
#w=2 #harmonic
#mom = np.zeros(2**n);
#mom[w]=1
#mom [2**n-w]=1
#initial_state= np.fft.fft(mom,2**n)
magnitude = np.linalg.norm(initial_state)
initial_state =initial_state/magnitude


stepsnumber = 5;
action()

# put x back and recentered the guassian
```



In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]:

In [ ]: