Task1

August 24, 2022

Task 1: The fast algorithm

A circular worldline for a particle can be written as

$\vec{x(t)} = R(cos(\frac{2pit}{T}), sin(\frac{2pit}{T})) = R(cos(wt), sin(wt))$

where $w = \frac{2pi}{T}$

Letting $x = Rcos(wt)$ and $y = Rsin(wt)$

$\Rightarrow \vec{x(t)} = (x, y)$

Deriving with respect to t

$\vec{v(t)} = (-\frac{2pi}{T}y, \frac{2pi}{T}x)$

We can rewrite it as

$

$$\begin{bmatrix} \dot{x} \\ \dot{y} \end{bmatrix} = \begin{bmatrix} 0 & -\frac{2pi}{T} \\ \frac{2pi}{T} & 0 \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} \tag{1}$$

$

Using the backward Euler method of integration, we can convert this to a linear system of equations

$

$$\begin{bmatrix} 1 & \frac{2pi\Delta t}{T} \\ -\frac{2pi\Delta t}{T} & 1 \end{bmatrix} \begin{bmatrix} x(t+\Delta t) \\ y(t+\Delta t) \end{bmatrix} = \begin{bmatrix} x(t) \\ y(t) \end{bmatrix} \tag{2}$$

$ $\cdots(1)$

A linear system problem (LSP) can be represented as the following

$A = $

where A is an N_bN_b Hermitian matrix and $\vec{x}$ and $\vec{b}$ are N_b-dimensional vectors. For simplicity, it is assumed $N_b = 2^{n_b}$. Otherwise, dummy equations can be added to convert the system satisfy this assumption. A and $\vec{b}$ are known and $\vec{x}$ is the unknown to be solved, i.e.

$ =A^{-1} $

Morrell Jr, H. J., & Wong, H. Y. (2021). Step-by-Step HHL Algorithm Walkthrough to Enhance the Understanding of Critical Quantum Computing Concepts. arXiv preprint arXiv:2108.09004.

if A is not Hermitian, then we can define $

$$\tilde{A} = \begin{bmatrix} 0 & A \\ A^{\dagger} & 0 \end{bmatrix} \tag{3}$$

$.

As $\tilde{A}$ is Hermitian, we can solve the equation $

$$\tilde{A}y = \begin{bmatrix} \vec{b} \\ 0 \end{bmatrix} \tag{4}$$

$ to get $

$$y = \begin{bmatrix} 0 \\ \vec{x} \end{bmatrix} \tag{5}$$

$

Harrow, A. W., Hassidim, A., & Lloyd, S. (2009). Quantum algorithm for linear systems of equations. Physical review letters, 103(15), 150502.

As Matrix $

$$\begin{bmatrix} 1 & \frac{2pi\Delta t}{T} \\ -\frac{2pi\Delta t}{T} & 1 \end{bmatrix} \tag{6}$$

$ in equation (1) is Skew-Hermitian we define

$

$$\begin{bmatrix} 0 & 0 & 1 & \frac{2pi\Delta t}{T} \\ 0 & 0 & -\frac{2pi\Delta t}{T} & 1 \\ 1 & -\frac{2pi\Delta t}{T} & 0 & 0 \\ \frac{2pi\Delta t}{T} & 1 & 0 & 0 \end{bmatrix} \tag{7}$$

$ which is now a Hermitian Matrix. For now we take the normalized problem where R=1 and $t_0 = 0$, so we have $

$$\begin{bmatrix} \vec{b} \\ 0 \end{bmatrix} = \begin{bmatrix} x(t_0) \\ y(t_0) \\ 0 \\ 0 \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \\ 0 \\ 0 \end{bmatrix} \tag{8}$$

$

The following code has been taken from https://qiskit.org/textbook/ch-applications/hhl_tutorial.html and modified to solve our problem

The interface for all algorithms to solve the linear system problem is `LinearSolver`. The problem to be solved is only specified when the `solve()` method is called:

`LinearSolver(...).solve(matrix, vector)`

The simplest implementation takes the matrix and the vector as NumPy arrays. Below we also create a `NumPyLinearSolver` (the classical algorithm) to validate our solutions.

[17]:
```
import numpy as np
from qiskit.algorithms.linear_solvers.numpy_linear_solver import␣
 ↪NumPyLinearSolver
from qiskit.algorithms.linear_solvers.hhl import HHL
T = 6
dt = 0.1
matrix = np.array([[0,0,1, (2*np.pi*dt)/T], [0,0,-(2*np.pi*dt)/T,1],[1,-(2*np.
 ↪pi*dt)/T,0,0],[(2*np.pi*dt)/T,1,0,0] ])
vector = np.array([1, 0, 0, 0])
naive_hhl_solution = HHL().solve(matrix, vector)
print("Hermitian Matrix for T=",T,"and delta t=",dt)
print(matrix)
print()
print("Initial position vector (b vector) at t=0")
print(vector)
```

```
Hermitian Matrix for T= 6 and delta t= 0.1
[[ 0.          0.          1.          0.10471976]
 [ 0.          0.         -0.10471976  1.        ]
 [ 1.         -0.10471976  0.          0.        ]
 [ 0.10471976  1.          0.          0.        ]]

Initial position vector (b vector) at t=0
[1 0 0 0]
```

For the classical solver we need to rescale the right hand side (i.e. `vector / np.linalg.norm(vector)`) to take into account the renormalisation that occurs once `vector` is encoded in a quantum state within HHL.

When np.linalg.norm() is called on an array-like input without any additional arguments, the default behavior is to compute the L2 norm on a flattened view of the array. This is the square root of the sum of squared elements and can be interpreted as the length of the vector in Euclidean space.

If you wanted the vector have a unit norm, you would simply divide every element by the norm.

[18]:
```
classical_solution = NumPyLinearSolver().solve(matrix, vector / np.linalg.
 ↪norm(vector))
```

Recall that the HHL algorithm can find a solution exponentially faster in the size of the system than their classical counterparts (i.e. logarithmic complexity instead of polynomial). However the cost for this exponential speedup is that we do not obtain the full solution vector. Instead, we obtain a quantum state representing the vector x and learning all the components of this vector would take a linear time in its dimension, diminishing any speedup obtained by the quantum algorithm.

Therefore, we can only compute functions from x (the so called observables) to learn information about the solution. This is reflected in the `LinearSolverResult` object returned by `solve()`, which contains the following properties - `state`: either the circuit that prepares the solution or the solution as a vector - `euclidean_norm`: the euclidean norm if the algorithm knows how to calculate it - `observable`: the (list of) calculated observable(s) - `circuit_results`: the observable results from the (list of) circuit(s)

Let's ignore `observable` and `circuit_results` for the time being and check the solutions we obtained before.

First, `classical_solution` was the result from a classical algorithm, so if we call `.state` it will return an array:

[19]:
```
print('classical state:', classical_solution.state)
```

```
classical state: [0.         0.         0.98915273 0.10358383]
```

Our other two examples were quantum algorithms, hence we can only access to the quantum state. This is achieved by returning the quantum circuit that prepares the solution state:

[20]:
```
print('naive state:')
print(naive_hhl_solution.state)
```

```
naive state:

q23286_0: 0               4     4
            circuit-303816
q23286_1: 1               5     5

q23287_0: 0     3     0
                                    QPE         QPE_dg
q23287_1: 1     2     1

q23287_2: 2     1 1/x 2

q23287_3: 3     0     3

  q23288: 4
```

Recall that the Euclidean norm for a vector $\mathbf{x} = (x_1, \ldots, x_N)$ is defined as $||\mathbf{x}|| = \sqrt{\sum_{i=1}^N x_i^2}$. Therefore, the probability of measuring 1 in the auxiliary qubit from Step is the squared norm of $\mathbf{x}$. This means that the HHL algorithm can always calculate the euclidean norm of the solution and we can compare the accuracy of the results:

[21]:
```
print('classical Euclidean norm:', classical_solution.euclidean_norm)
print('naive Euclidean norm:', naive_hhl_solution.euclidean_norm)
```

```
classical Euclidean norm: 0.9945615750429233
naive Euclidean norm: 0.9945615750429039
```

Comparing the solution vectors componentwise is more tricky, reflecting again the idea that we cannot obtain the full solution vector from the quantum algorithm. However, for educational purposes we can check that indeed the different solution vectors obtained are a good approximation at the vector component level as well.

To do so first we need to use `Statevector` from the `quantum_info` package and extract the right vector components, i.e. those corresponding to the ancillary qubit (bottom in the circuits) being 1 and the work qubits (the two middle in the circuits) being 0. Thus, we are interested in the states 1000 and 1001, corresponding to the first and second components of the solution vector respectively.

Following the results from Cao, Y., Daskin, A., Frankel, S., & Kais, S. (2012). Quantum circuit design for solving linear systems of equations. Molecular Physics, 110(15-16), 1675-1680. For the final state of the algorithm for solving the 4 × 4 system, we have to choose the correct decimal value that corresponds to the basis state of the seven-qubit system.

Basis State |i > = |Anc. >|Reg.C >|Reg.b >

[22]:
```
from qiskit.quantum_info import Statevector

naive_sv = Statevector(naive_hhl_solution.state).data
#tridi_sv = Statevector(tridi_solution.state).data

# Extract the right vector components. |1000000> corresponds to the index 64,␣
↪|1000001> corresponds to the index 65,|1000010> corresponds to the index 66␣
↪and |1000011> corresponds to the index 67
naive_full_vector = np.array([ naive_sv[64], naive_sv[65], naive_sv[66],␣
↪naive_sv[67]])

print('naive raw solution vector:', naive_full_vector)
```

```
naive raw solution vector: [9.01578195e-16-4.97636903e-15j
4.66014248e-15-3.70421887e-16j
 9.94561575e-01-1.57365885e-14j 1.04150245e-01-1.43101505e-14j]
```

At a first glance it might seem that this is wrong because the components are complex numbers instead of reals. However note that the imaginary part is very small, most likely due to computer accuracy, and can be disregarded in this case.

[23]:
```
naive_full_vector = np.real(naive_full_vector)
```

Next, we will divide the vectors by their respective norms to suppress any constants coming from the different parts of the circuits. The full solution vector can then be recovered by multiplying these normalised vectors by the respective Euclidean norms calculated above:

[24]:
```
print('full naive solution vector:', naive_hhl_solution.
↪euclidean_norm*naive_full_vector/np.linalg.norm(naive_full_vector))
#print('full tridi solution vector:', tridi_solution.
↪euclidean_norm*tridi_full_vector/np.linalg.norm(tridi_full_vector))
print('classical state:', classical_solution.state)
```

```
full naive solution vector: [8.96675029e-16 4.63479864e-15 9.89152727e-01
1.03583831e-01]
classical state: [0.         0.         0.98915273 0.10358383]
```

Remembering that for this problem we seek to find $

$$y = \begin{bmatrix} 0 \\ \vec{x} \end{bmatrix} \tag{9}$$

$, we just consider the components of $\vec{x}$

We are interested in the third and fourth components of the solution vector, which correspond to the components of $\vec{x}$. Thus, we are interested in the states |1000010> corresponding to the third component of the solution vector and |1000011> corresponding to the fourth component of the solution vector. |1000010> corresponds to the index 66 and |1000011> corresponds to the index 67

[25]:
```
naive_partial_vector = np.array([ 0, 0, naive_sv[66], naive_sv[67]])
naive_partial_vector = np.real(naive_partial_vector)  #Real part of␣
 ↪naive_partial_vector
```

[26]:
```
print('full naive solution vector:', "y=[",0,"          ",0,"          ␣
 ↪",naive_hhl_solution.euclidean_norm*naive_partial_vector[2]/np.linalg.
 ↪norm(naive_partial_vector),naive_hhl_solution.euclidean_norm*␣
 ↪naive_partial_vector[3]/np.linalg.norm(naive_partial_vector),"]") #print full␣
 ↪naive solution vector
print('from classical state:', "y=",classical_solution.state)
```

```
full naive solution vector: y=[ 0          0          0.989152726551841
0.10358383130045241 ]
from classical state: y= [0.         0.         0.98915273 0.10358383]
```

Calculate the circuit depth

[27]:
```
from qiskit import transpile
naive_depths = []
naive_hhl_solution = HHL().solve(matrix, vector)
naive_qc = transpile(naive_hhl_solution.state,basis_gates=['id', 'rz', 'sx',␣
 ↪'x', 'cx'])
naive_depths.append(naive_qc.depth())
```

[28]:
```
nb=2
sizes = [str(2**nb)+"x"+str(2**nb)]
columns = ['size of the system', 'quantum_solution depth']
data = np.array([sizes, naive_depths])
#print(columns[0],data[0])
#print(columns[1],data[1])
row_format ="{:>23}" * (len(columns) )
for team, row in zip(columns, data):
    print(row_format.format(team, *row))
```

```
     size of the system                    4x4
quantum_solution depth                   2623
```

Now we will perform the loop operation to obtain data for simulating the orbit of the James Webb telescope using the implicit Euler method of integration and the HHL algorithm.

[29]:
```
#Code to simulate the orbit of the James Webb telescope
xv1=[1] #initial position for quantum algorithm
```

```
yv1=[0]
cxv1 = [1] #initial position for classic algorithm
cyv1 = [0]
t1 = [] #list to keep track of time
vector = np.array([1, 0, 0, 0])  #vector b representing initial position at time␣
↪t=0
vector = vector/np.linalg.norm(vector)  #normaized vector b representing initial␣
↪position at time t=0
for i in range(181):
    T = 180  #180 days= 6 months (30 days per month)
    dt = 1  #every day we measure
    print('On day',i,'the results are') #print day of measurment
    print('normalized vector imput', vector)  #print normalized imput vector
    print('square of the norm of vector imput',np.linalg.norm(vector)**2) #print␣
↪square of the norm of vector imput(sum of squared components) to cheek if it␣
↪is a valid quantum state
    matrix = np.array([[0,0,1, (2*np.pi*dt)/T], [0,0,-(2*np.pi*dt)/
↪T,1],[1,-(2*np.pi*dt)/T,0,0],[(2*np.pi*dt)/T,1,0,0] ])  #Hermitian Matrix
    naive_hhl_solution = HHL().solve(matrix, vector)  #naive_hhl_solution from␣
↪HHL algorithm
    classical_solution = NumPyLinearSolver().solve(matrix, vector / np.linalg.
↪norm(vector)) #classical_solution from classical solution algorithm
    naive_sv = Statevector(naive_hhl_solution.state).data #state vector of␣
↪quantum solution
    #Extract the right vector components, i.e. those corresponding to the␣
↪ancillary qubit (bottom in the circuits) being 1 and the work qubits (the four␣
↪middle in the circuits) being 0,
    #Extract the right vector components. |1000000> corresponds to the index 64,␣
↪|1000001> corresponds to the index 65,|1000010> corresponds to the index 66␣
↪and |1000011> corresponds to the index 67
    naive_full_vector = np.array([ naive_sv[64], naive_sv[65], naive_sv[66],␣
↪naive_sv[67]])
    naive_full_vector = np.real(naive_full_vector)  #Real part of␣
↪naive_full_vector
    #We are interested in the third and fourth components of the solution␣
↪vector, which correspond to the components of \vec{x}
    #Thus, we are interested in the states |1000010> ,|1000011>, corresponding␣
↪to the third and fourth components of the solution vector respectively.␣
↪|1000010> corresponds to the index 66 and |1000011> corresponds to the index␣
↪67
    naive_partial_vector = np.array([ 0, 0, naive_sv[66], naive_sv[67]])
    naive_partial_vector = np.real(naive_partial_vector)  #Real part of␣
↪naive_full_vector
    x=naive_partial_vector[2]/np.linalg.norm(naive_partial_vector)  #normalized␣
↪x component
```

```
    y = naive_partial_vector[3]/np.linalg.norm(naive_partial_vector) ␣
↪#normalized y component
    vector=np.array([x,y, 0, 0])  #normaized vector b representing initial␣
↪position at time t=i*dt
    #print('full naive solution vector:', "y=[",0,"         ",0,"        ␣
↪",naive_hhl_solution.euclidean_norm*naive_full_vector[2]/np.linalg.
↪norm(naive_full_vector),naive_hhl_solution.euclidean_norm*␣
↪naive_full_vector[3]/np.linalg.norm(naive_full_vector),"]")
    #we will divide the vectors by their respective norms to suppress any␣
↪constants coming from the different parts of the circuits. The full solution␣
↪vector can then be recovered by multiplying these normalised vectors by the␣
↪respective Euclidean norms calculated above
    print('full naive solution vector:', "y=[",0,"         ",0,"        ␣
↪",naive_hhl_solution.euclidean_norm*naive_partial_vector[2]/np.linalg.
↪norm(naive_partial_vector),naive_hhl_solution.euclidean_norm*␣
↪naive_partial_vector[3]/np.linalg.norm(naive_partial_vector),"]") #print full␣
↪naive solution vector
    print('classical state:', "y=",classical_solution.state) #print classical␣
↪state
    print()
    xv1.append(naive_hhl_solution.euclidean_norm*naive_partial_vector[2]/np.
↪linalg.norm(naive_partial_vector))  #add the result from "x(t_i)" component␣
↪obtained from HHL algorithm to the list
    yv1.append(naive_hhl_solution.euclidean_norm*naive_partial_vector[3]/np.
↪linalg.norm(naive_partial_vector))  #add the result from "y(t_i)" component␣
↪obtained from HHL algorithm to the list
    cxv1.append(classical_solution.state[2])  #add the result from "x(t_i)"␣
↪component obtained from classical algorithm to the list
    cyv1.append(classical_solution.state[3])  #add the result from "y(t_i)"␣
↪component obtained from classical algorithm to the list
    t1.append(i) #savind time step
```

```
On day 0 the results are
normalized vector imput [1. 0. 0. 0.]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.9987830131823715
0.03486410418604186 ]
classical state: y= [0.         0.         0.99878301 0.0348641 ]

On day 1 the results are
normalized vector imput [0.99939132 0.03488534 0.         0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.9969588292189014
0.0696857663000798 ]
classical state: y= [0.         0.         0.99695883 0.06968577]

On day 2 the results are
```

```
normalized vector imput [0.99756603 0.06972821 0.         0.        ]
square of the norm of vector imput 0.9999999999999998
full naive solution vector: y=[ 0          0          0.9939209901395925
0.10442259593729666 ]
classical state: y= [0.         0.         0.99392099 0.1044226 ]

On day 3 the results are
normalized vector imput [0.99452634 0.10448619 0.         0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.9896731940800472
0.13903230596430186 ]
classical state: y= [0.         0.         0.98967319 0.13903231]

On day 4 the results are
normalized vector imput [0.99027595 0.13911698 0.         0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.9842206121258348
0.17347276399768854 ]
classical state: y= [0.         0.         0.98422061 0.17347276]

On day 5 the results are
normalized vector imput [0.98482005 0.17357842 0.         0.        ]
square of the norm of vector imput 0.9999999999999998
full naive solution vector: y=[ 0          0          0.9775698820174449
0.20770204369420378 ]
classical state: y= [0.         0.         0.97756988 0.20770204]

On day 6 the results are
normalized vector imput [0.97816527 0.20782854 0.         0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.9697291000697731
0.2416784757900855 ]
classical state: y= [0.         0.         0.9697291  0.24167848]

On day 7 the results are
normalized vector imput [0.97031971 0.24182567 0.         0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.9607078113160171
0.2753606988274437 ]
classical state: y= [0.         0.         0.96070781 0.2753607 ]

On day 8 the results are
normalized vector imput [0.96129293 0.27552841 0.         0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.9505169978879633
0.30870770950593845 ]
classical state: y= [0.         0.         0.950517   0.30870771]
```

```
On day 9 the results are
normalized vector imput [0.95109591 0.30889573 0.         0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.9391690656468062
0.3416789125984434 ]
classical state: y= [0.         0.         0.93916907 0.34167891]

On day 10 the results are
normalized vector imput [0.93974107 0.34188701 0.         0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.9266778290807833
0.3742341703699428 ]
classical state: y= [0.         0.         0.92667783 0.37423417]

On day 11 the results are
normalized vector imput [0.92724222 0.3744621  0.         0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.9130584944880022
0.40633385143949663 ]
classical state: y= [0.         0.         0.91305849 0.40633385]

On day 12 the results are
normalized vector imput [0.91361459 0.40658133 0.         0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.8983276414649514
0.43793887902579226 ]
classical state: y= [0.         0.         0.89832764 0.43793888]

On day 13 the results are
normalized vector imput [0.89887477 0.43820561 0.         0.        ]
square of the norm of vector imput 0.9999999999999996
full naive solution vector: y=[ 0          0          0.8825032027232027
0.46901077851755046 ]
classical state: y= [0.         0.         0.8825032  0.46901078]

On day 14 the results are
normalized vector imput [0.88304069 0.46929643 0.         0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.8656044422588927
0.4995117243108732 ]
classical state: y= [0.         0.         0.86560444 0.49951172]

On day 15 the results are
normalized vector imput [0.86613164 0.49981595 0.         0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.8476519319015551
0.5294045858565174 ]
classical state: y= [0.         0.         0.84765193 0.52940459]
```

```
On day 16 the results are
normalized vector imput [0.84816819 0.52972702 0.         0.        ]
square of the norm of vector imput 0.9999999999999998
full naive solution vector: y=[ 0          0          0.8286675262708592
0.5586529728610454 ]
classical state: y= [0.         0.         0.82866753 0.55865297]

On day 17 the results are
normalized vector imput [0.82917223 0.55899322 0.         0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.8086743361717249
0.58722127958681 ]
classical state: y= [0.         0.         0.80867434 0.58722128]

On day 18 the results are
normalized vector imput [0.80916686 0.58757893 0.         0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.7876967004602138
0.6150747281968603 ]
classical state: y= [0.         0.         0.7876967  0.61507473]

On day 19 the results are
normalized vector imput [0.78817645 0.61544934 0.         0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.7657601564144497
0.6421794110919974 ]
classical state: y= [0.         0.         0.76576016 0.64217941]

On day 20 the results are
normalized vector imput [0.76622654 0.64257053 0.         0.        ]
square of the norm of vector imput 0.9999999999999998
full naive solution vector: y=[ 0          0          0.7428914086466268
0.6685023321884367 ]
classical state: y= [0.         0.         0.74289141 0.66850233]

On day 21 the results are
normalized vector imput [0.74334387 0.66890948 0.         0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.7191182965939461
0.6940114470858307 ]
classical state: y= [0.         0.         0.7191183  0.69401145]

On day 22 the results are
normalized vector imput [0.71955628 0.69443413 0.         0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.694469760628085
0.7186757020767567 ]
```

```
classical state: y= [0.         0.         0.69446976 0.7186757 ]

On day 23 the results are
normalized vector imput [0.69489273 0.71911341 0.         0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.6689758068244125
0.7424650719501705 ]
classical state: y= [0.         0.         0.66897581 0.74246507]

On day 24 the results are
normalized vector imput [0.66938325 0.74291727 0.         0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.6426674704338882
0.7653505965428242 ]
classical state: y= [0.         0.         0.64266747 0.7653506 ]

On day 25 the results are
normalized vector imput [0.64305889 0.76581673 0.         0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.6155767781020566
0.787304415994118 ]
classical state: y= [0.         0.         0.61557678 0.78730442]

On day 26 the results are
normalized vector imput [0.61595169 0.78778392 0.         0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.5877367088811896
0.808299804661522 ]
classical state: y= [0.         0.         0.58773671 0.8082998 ]

On day 27 the results are
normalized vector imput [0.58809467 0.8087921  0.         0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.55918115408299
0.8283112036552217 ]
classical state: y= [0.         0.         0.55918115 0.8283112 ]

On day 28 the results are
normalized vector imput [0.55952172 0.82881569 0.         0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.5299448760207551
0.847314251952423 ]
classical state: y= [0.         0.         0.52994488 0.84731425]

On day 29 the results are
normalized vector imput [0.53026764 0.84783031 0.         0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.5000634656912207
0.8652858160534193 ]
classical state: y= [0.         0.         0.50006347 0.86528582]

On day 30 the results are
normalized vector imput [0.50036803 0.86581282 0.         0.        ]
square of the norm of vector imput 0.9999999999999998
full naive solution vector: y=[ 0          0          0.4695732994476077
0.8822040181433279 ]
classical state: y= [0.         0.         0.4695733  0.88220402]

On day 31 the results are
normalized vector imput [0.46985929 0.88274132 0.         0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.43851149471659223
0.8980482627252073 ]
classical state: y= [0.         0.         0.43851149 0.89804826]

On day 32 the results are
normalized vector imput [0.43877857 0.89859522 0.         0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.40691586481314274
0.9127992616921395 ]
classical state: y= [0.         0.         0.40691586 0.91279926]

On day 33 the results are
normalized vector imput [0.4071637 0.9133552 0.        0.       ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.37482487290819955
0.9264390578077414 ]
classical state: y= [0.         0.         0.37482487 0.92643906]

On day 34 the results are
normalized vector imput [0.37505316 0.9270033  0.         0.        ]
square of the norm of vector imput 0.9999999999999998
full naive solution vector: y=[ 0          0          0.34227758520525936
0.9389510465665414 ]
classical state: y= [0.         0.         0.34227759 0.93895105]

On day 35 the results are
normalized vector imput [0.34248605 0.93952291 0.         0.        ]
square of the norm of vector imput 0.9999999999999998
full naive solution vector: y=[ 0          0          0.3093136233828391
0.9503199964075922 ]
classical state: y= [0.         0.         0.30931362 0.95032   ]

On day 36 the results are
normalized vector imput [0.30950201 0.95089879 0.         0.        ]
square of the norm of vector imput 1.0
```

```
full naive solution vector: y=[ 0          0          0.27597311636074134
0.9605320672567298 ]
classical state: y= [0.         0.         0.27597312 0.96053207]

On day 37 the results are
normalized vector imput [0.2761412  0.96111708 0.         0.        ]
square of the norm of vector imput 0.9999999999999998
full naive solution vector: y=[ 0          0          0.24229665144882004
0.9695748273748822 ]
classical state: y= [0.         0.         0.24229665 0.96957483]

On day 38 the results are
normalized vector imput [0.24244422 0.97016535 0.         0.        ]
square of the norm of vector imput 0.9999999999999998
full naive solution vector: y=[ 0          0          0.20832522493772002
0.9774372684919445 ]
classical state: y= [0.         0.         0.20832522 0.97743727]

On day 39 the results are
normalized vector imput [0.20845211 0.97803258 0.         0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.1741001921917438
0.9841098192077654 ]
classical state: y= [0.         0.         0.17410019 0.98410982]

On day 40 the results are
normalized vector imput [0.17420623 0.98470919 0.         0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.13966321730459846
0.9895843566439771 ]
classical state: y= [0.         0.         0.13966322 0.98958436]

On day 41 the results are
normalized vector imput [0.13974828 0.99018706 0.         0.        ]
square of the norm of vector imput 0.9999999999999998
full naive solution vector: y=[ 0          0          0.1050562223793042
0.9938542163324213 ]
classical state: y= [0.         0.         0.10505622 0.99385422]

On day 42 the results are
normalized vector imput [0.10512021 0.99445952 0.         0.        ]
square of the norm of vector imput 0.9999999999999998
full naive solution vector: y=[ 0          0          0.07032133649401798
0.9969142003282133 ]
classical state: y= [0.         0.         0.07032134 0.9969142 ]

On day 43 the results are
normalized vector imput [0.07036417 0.99752137 0.         0.        ]
```

```
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.03550084441589055
0.998760583537466 ]
classical state: y= [0.         0.         0.03550084 0.99876058]

On day 44 the results are
normalized vector imput [0.03552247 0.99936888 0.         0.        ]
square of the norm of vector imput 0.9999999999999998
full naive solution vector: y=[ 0          0          0.0006371351253980351
0.999391118252096 ]
classical state: y= [0.00000000e+00 0.00000000e+00 6.37135125e-04
9.99391118e-01]

On day 45 the results are
normalized vector imput [6.37523172e-04 9.99999797e-01 0.00000000e+00
0.00000000e+00]
square of the norm of vector imput 0.9999999999999998
full naive solution vector: y=[ 0          0          -0.03422734978619487
0.998805036886057 ]
classical state: y= [ 0.          0.         -0.03422735  0.99880504]

On day 46 the results are
normalized vector imput [-0.0342482   0.99941336  0.          0.        ]
square of the norm of vector imput 0.9999999999999998
full naive solution vector: y=[ 0          0          -0.06905016778341551
0.9970030529097815 ]
classical state: y= [ 0.          0.         -0.06905017  0.99700305]

On day 47 the results are
normalized vector imput [-0.06909222  0.99761028  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.10378892705431349
0.9939873599816331 ]
classical state: y= [ 0.          0.         -0.10378893  0.99398736]

On day 48 the results are
normalized vector imput [-0.10385214  0.99459275  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.13840133811644328
0.989761629277435 ]
classical state: y= [ 0.          0.         -0.13840134  0.98976163]

On day 49 the results are
normalized vector imput [-0.13848563  0.99036444  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.17284526529827451
0.9843310050213374 ]
classical state: y= [ 0.          0.         -0.17284527  0.98433101]
```

```
On day 50 the results are
normalized vector imput [-0.17295054  0.98493051  0.          0.        ]
square of the norm of vector imput 0.9999999999999998
full naive solution vector: y=[ 0          0          -0.20707877803335942
0.9777020982234593 ]
classical state: y= [ 0.          0.         -0.20707878  0.9777021 ]

On day 51 the results are
normalized vector imput [-0.2072049   0.97829757  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.24106020190481575
0.9698829786319356 ]
classical state: y= [ 0.          0.         -0.2410602   0.96988298]

On day 52 the results are
normalized vector imput [-0.24120702  0.97047369  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.2747481693779801
0.9608831649091338 ]
classical state: y= [ 0.          0.         -0.27474817  0.96088316]

On day 53 the results are
normalized vector imput [-0.2749155   0.96146839  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.30810167015948026
0.9507136130440555 ]
classical state: y= [ 0.          0.         -0.30810167  0.95071361]

On day 54 the results are
normalized vector imput [-0.30828932  0.95129264  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.3410801011214164
0.9393867030149776 ]
classical state: y= [ 0.         0.        -0.3410801  0.9393867]

On day 55 the results are
normalized vector imput [-0.34128784  0.93995884  0.          0.        ]
square of the norm of vector imput 0.9999999999999998
full naive solution vector: y=[ 0          0          -0.3736433157298723
0.9269162237185902 ]
classical state: y= [ 0.          0.         -0.37364332  0.92691622]

On day 56 the results are
normalized vector imput [-0.37387088  0.92748076  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.40575167291759545
0.9133173561839923 ]
```

```
classical state: y= [ 0.          0.         -0.40575167  0.91331736]

On day 57 the results are
normalized vector imput [-0.4059988   0.91387361  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.4373660853413368
0.8986066550919483 ]
classical state: y= [ 0.          0.         -0.43736609  0.89860666]

On day 58 the results are
normalized vector imput [-0.43763246  0.89915395  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.46844806696511593
0.8828020286219284 ]
classical state: y= [ 0.          0.         -0.46844807  0.88280203]

On day 59 the results are
normalized vector imput [-0.46873337  0.8833397   0.          0.        ]
square of the norm of vector imput 0.9999999999999998
full naive solution vector: y=[ 0          0          -0.49895977991147783
0.8659227166514545 ]
classical state: y= [ 0.          0.         -0.49895978  0.86592272]

On day 60 the results are
normalized vector imput [-0.49926367  0.86645011  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.5288640805237081
0.8479892673342971 ]
classical state: y= [ 0.          0.         -0.52886408  0.84798927]

On day 61 the results are
normalized vector imput [-0.52918618  0.84850573  0.          0.        ]
square of the norm of vector imput 0.9999999999999998
full naive solution vector: y=[ 0          0          -0.5581245645829371
0.8290235120860272 ]
classical state: y= [ 0.          0.         -0.55812456  0.82902351]

On day 62 the results are
normalized vector imput [-0.55846449  0.82952843  0.          0.        ]
square of the norm of vector imput 0.9999999999999998
full naive solution vector: y=[ 0          0          -0.5867056116250797
0.8090485390073868 ]
classical state: y= [ 0.          0.         -0.58670561  0.80904854]

On day 63 the results are
normalized vector imput [-0.58706294  0.80954129  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.614572428303668
0.7880886647778145 ]
classical state: y= [ 0.          0.         -0.61457243  0.78808866]

On day 64 the results are
normalized vector imput [-0.61494673  0.78856865  0.          0.        ]
square of the norm of vector imput 0.9999999999999998
full naive solution vector: y=[ 0          0          -0.6416910907457869
0.7661694050533653 ]
classical state: y= [ 0.          0.         -0.64169109  0.76616941]

On day 65 the results are
normalized vector imput [-0.64208191  0.76663604  0.          0.        ]
square of the norm of vector imput 1.0000000000000004
full naive solution vector: y=[ 0          0          -0.6680285858495484
0.7433174434050356 ]
classical state: y= [ 0.          0.         -0.66802859  0.74331744]

On day 66 the results are
normalized vector imput [-0.66843545  0.74377016  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.6935528514728317
0.7195605988353243 ]
classical state: y= [ 0.          0.         -0.69355285  0.7195606 ]

On day 67 the results are
normalized vector imput [-0.69397526  0.71999885  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.7182328154643677
0.6949277919125676 ]
classical state: y= [ 0.          0.         -0.71823282  0.69492779]

On day 68 the results are
normalized vector imput [-0.71867025  0.69535104  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.7420384334896428
0.6694490095642696 ]
classical state: y= [ 0.          0.         -0.74203843  0.66944901]

On day 69 the results are
normalized vector imput [-0.74249037  0.66985674  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.7649407256056046
0.6431552685723082 ]
classical state: y= [ 0.          0.         -0.76494073  0.64315527]

On day 70 the results are
normalized vector imput [-0.76540661  0.64354698  0.          0.        ]
square of the norm of vector imput 1.0
```

```
full naive solution vector: y=[ 0          0          -0.7869118115395887
0.6160785778144136 ]
classical state: y= [ 0.          0.         -0.78691181  0.61607858]

On day 71 the results are
normalized vector imput [-0.78739108  0.6164538   0.          0.        ]
square of the norm of vector imput 0.9999999999999998
full naive solution vector: y=[ 0          0          -0.8079249446295835
0.5882518992979383 ]
classical state: y= [ 0.          0.         -0.80792494  0.5882519 ]

On day 72 the results are
normalized vector imput [-0.80841701  0.58861017  0.          0.        ]
square of the norm of vector imput 0.9999999999999998
full naive solution vector: y=[ 0          0          -0.827954544384473
0.5597091080333136 ]
classical state: y= [ 0.          0.         -0.82795454  0.55970911]

On day 73 the results are
normalized vector imput [-0.82845881  0.56005     0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.8469762276246326
0.5304849507960548 ]
classical state: y= [ 0.          0.         -0.84697623  0.53048495]

On day 74 the results are
normalized vector imput [-0.84749208  0.53080804  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.8649668381649742
0.5006150038275267 ]
classical state: y= [ 0.          0.         -0.86496684  0.500615  ]

On day 75 the results are
normalized vector imput [-0.86549365  0.5009199   0.          0.        ]
square of the norm of vector imput 1.0000000000000004
full naive solution vector: y=[ 0          0          -0.881904475004293
0.4701356295259397 ]
classical state: y= [ 0.          0.         -0.88190448  0.47013563]

On day 76 the results are
normalized vector imput [-0.8824416   0.47042197  0.          0.        ]
square of the norm of vector imput 0.9999999999999998
full naive solution vector: y=[ 0          0          -0.8977685189866134
0.43908393218032143 ]
classical state: y= [ 0.          0.         -0.89776852  0.43908393]

On day 77 the results are
normalized vector imput [-0.8983153   0.43935136  0.          0.        ]
```

```
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.9125396579020693
0.4074977128013398 ]
classical state: y= [ 0.         0.         -0.91253966  0.40749771]

On day 78 the results are
normalized vector imput [-0.91309544  0.4077459   0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.9261999099967677
0.37541542310396786 ]
classical state: y= [ 0.         0.         -0.92619991  0.37541542]

On day 79 the results are
normalized vector imput [-0.92676401  0.37564407  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.938732645862997
0.34287611869800116 ]
classical state: y= [ 0.         0.         -0.93873265  0.34287612]

On day 80 the results are
normalized vector imput [-0.93930438  0.34308495  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.9501226086831736
0.3099194115434359 ]
classical state: y= [ 0.         0.         -0.95012261  0.30991941]

On day 81 the results are
normalized vector imput [-0.95070128  0.31010817  0.          0.        ]
square of the norm of vector imput 0.9999999999999998
full naive solution vector: y=[ 0          0          -0.9603559328028287
0.27658542172855133 ]
classical state: y= [ 0.         0.         -0.96035593  0.27658542]

On day 82 the results are
normalized vector imput [-0.96094084  0.27675388  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.9694201606100626
0.24291472862942853 ]
classical state: y= [ 0.         0.         -0.96942016  0.24291473]

On day 83 the results are
normalized vector imput [-0.97001059  0.24306268  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.9773042577009248
0.20894832151035594 ]
classical state: y= [ 0.         0.         -0.97730426  0.20894832]

On day 84 the results are
```

```
normalized vector imput [-0.97789948  0.20907558  0.          0.        ]
square of the norm of vector imput 0.9999999999999998
full naive solution vector: y=[ 0          0          -0.9839986263122067
0.17472754962524165 ]
classical state: y= [ 0.          0.         -0.98399863  0.17472755]

On day 85 the results are
normalized vector imput [-0.98459793  0.17483397  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.9894951170053623
0.1402940718808027 ]
classical state: y= [ 0.          0.         -0.98949512  0.14029407]

On day 86 the results are
normalized vector imput [-0.99009777  0.14037952  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.9937870385872807
0.10568980612278649 ]
classical state: y= [ 0.          0.         -0.99378704  0.10568981]

On day 87 the results are
normalized vector imput [-0.9943923   0.10575418  0.          0.        ]
square of the norm of vector imput 0.9999999999999996
full naive solution vector: y=[ 0          0          -0.9968691662558649
0.0709568781069787 ]
classical state: y= [ 0.          0.         -0.99686917  0.07095688]

On day 88 the results are
normalized vector imput [-0.99747631  0.07100009  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.9987377479604718
0.0361375702171031 ]
classical state: y= [ 0.          0.         -0.99873775  0.03613757]

On day 89 the results are
normalized vector imput [-0.99934603  0.03615958  0.          0.        ]
square of the norm of vector imput 0.9999999999999998
full naive solution vector: y=[ 0          0          -0.9993905089695126
0.0012742699920553205 ]
classical state: y= [ 0.          0.         -0.99939051  0.00127427]

On day 90 the results are
normalized vector imput [-0.99999919  0.00127505  0.          0.        ]
square of the norm of vector imput 0.9999999999999998
full naive solution vector: y=[ 0          0          -0.9988266546395983
-0.03359058147487921 ]
classical state: y= [ 0.          0.         -0.99882665 -0.03359058]
```

```
On day 91 the results are
normalized vector imput [-0.99943499 -0.03361104  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.9970468713829209
-0.06841454120199858 ]
classical state: y= [ 0.          0.         -0.99704687 -0.06841454]

On day 92 the results are
normalized vector imput [-0.99765412 -0.06845621  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.9940533258316336
-0.10315521598745966 ]
classical state: y= [ 0.          0.         -0.99405333 -0.10315522]

On day 93 the results are
normalized vector imput [-0.99465875 -0.10321804  0.          0.        ]
square of the norm of vector imput 0.9999999999999998
full naive solution vector: y=[ 0          0          -0.9898496622002871
-0.1377703140169519 ]
classical state: y= [ 0.          0.         -0.98984966 -0.13777031]

On day 94 the results are
normalized vector imput [-0.99045253 -0.13785422  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.9844409978495199
-0.1722176963479454 ]
classical state: y= [ 0.         0.        -0.984441  -0.1722177]

On day 95 the results are
normalized vector imput [-0.98504057 -0.17232259  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.9778339170564022
-0.2064554282078414 ]
classical state: y= [ 0.          0.         -0.97783392 -0.20645543]

On day 96 the results are
normalized vector imput [-0.97842947 -0.20658117  0.          0.        ]
square of the norm of vector imput 0.9999999999999998
full naive solution vector: y=[ 0          0          -0.9700364629990195
-0.24044183004357506 ]
classical state: y= [ 0.          0.         -0.97003646 -0.24044183]

On day 97 the results are
normalized vector imput [-0.97062726 -0.24058827  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.9610581279650446
-0.274135528260522 ]
classical state: y= [ 0.          0.         -0.96105813 -0.27413553]
```

```
On day 98 the results are
normalized vector imput [-0.96164346 -0.27430249  0.          0.        ]
square of the norm of vector imput 0.9999999999999998
full naive solution vector: y=[ 0          0          -0.9509098417962384
-0.30749550558894767 ]
classical state: y= [ 0.          0.         -0.95090984 -0.30749551]

On day 99 the results are
normalized vector imput [-0.95148899 -0.30768279  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.9396039585829268
-0.3404811510166792 ]
classical state: y= [ 0.          0.         -0.93960396 -0.34048115]

On day 100 the results are
normalized vector imput [-0.94017622 -0.34068852  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.927154241624656
-0.37305230922721955 ]
classical state: y= [ 0.          0.         -0.92715424 -0.37305231]

On day 101 the results are
normalized vector imput [-0.92771892 -0.37327952  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.9135758466753352
-0.4051693294831126 ]
classical state: y= [ 0.          0.         -0.91357585 -0.40516933]

On day 102 the results are
normalized vector imput [-0.91413226 -0.4054161   0.          0.        ]
square of the norm of vector imput 0.9999999999999998
full naive solution vector: y=[ 0          0          -0.8988853034932796
-0.4367931138950612 ]
classical state: y= [ 0.          0.         -0.8988853  -0.43679311]

On day 103 the results are
normalized vector imput [-0.89943277 -0.43705914  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.8831004957185815
-0.4678851650180274 ]
classical state: y= [ 0.          0.         -0.8831005  -0.46788517]

On day 104 the results are
normalized vector imput [-0.88363835 -0.46817013  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.8662406391023266
-0.49840763271637545 ]
```

```
classical state: y= [ 0.          0.         -0.86624064 -0.49840763]

On day 105 the results are
normalized vector imput [-0.86676822 -0.49871119  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.8483262581141696
-0.5283233602410169 ]
classical state: y= [ 0.          0.         -0.84832626 -0.52832336]

On day 106 the results are
normalized vector imput [-0.84884293 -0.52864514  0.          0.        ]
square of the norm of vector imput 0.9999999999999998
full naive solution vector: y=[ 0          0          -0.8293791609567124
-0.5575959294624452 ]
classical state: y= [ 0.          0.         -0.82937916 -0.55759593]

On day 107 the results are
normalized vector imput [-0.82988429 -0.55793553  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.8094224130171307
-0.5861897052046154 ]
classical state: y= [ 0.          0.         -0.80942241 -0.58618971]

On day 108 the results are
normalized vector imput [-0.80991539 -0.58654672  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.7884803087883602
-0.6140698786256837 ]
classical state: y= [ 0.          0.         -0.78848031 -0.61406988]

On day 109 the results are
normalized vector imput [-0.78896053 -0.61444388  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.7665783422940112
-0.6412025095928067 ]
classical state: y= [ 0.          0.         -0.76657834 -0.64120251]

On day 110 the results are
normalized vector imput [-0.76704523 -0.64159303  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.7437431760530431
-0.6675545679994098 ]
classical state: y= [ 0.          0.         -0.74374318 -0.66755457]

On day 111 the results are
normalized vector imput [-0.74419615 -0.66796114  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.7200026086219441
-0.6930939739746305 ]
classical state: y= [ 0.          0.          -0.72000261 -0.69309397]

On day 112 the results are
normalized vector imput [-0.72044113 -0.6935161   0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.6953855407539516
-0.7177896369359771 ]
classical state: y= [ 0.          0.          -0.69538554 -0.71778964]

On day 113 the results are
normalized vector imput [-0.69580906 -0.71822681  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.6699219402165228
-0.7416114934376883 ]
classical state: y= [ 0.          0.          -0.66992194 -0.74161149]

On day 114 the results are
normalized vector imput [-0.67032996 -0.74206317  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.6436428053098407
-0.7645305437686745 ]
classical state: y= [ 0.          0.          -0.64364281 -0.76453054]

On day 115 the results are
normalized vector imput [-0.64403482 -0.76499618  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.6165801271308152
-0.7865188872555412 ]
classical state: y= [ 0.          0.          -0.61658013 -0.78651889]

On day 116 the results are
normalized vector imput [-0.61695565 -0.78699792  0.          0.        ]
square of the norm of vector imput 0.9999999999999998
full naive solution vector: y=[ 0          0          -0.5887668506284824
-0.807549756227667 ]
classical state: y= [ 0.          0.          -0.58876685 -0.80754976]

On day 117 the results are
normalized vector imput [-0.58912544 -0.80804159  0.          0.        ]
square of the norm of vector imput 0.9999999999999998
full naive solution vector: y=[ 0          0          -0.5602368344982329
-0.8275975486030311 ]
classical state: y= [ 0.          0.          -0.56023683 -0.82759755]

On day 118 the results are
normalized vector imput [-0.56057805 -0.8281016   0.          0.        ]
square of the norm of vector imput 0.9999999999999998
```

```
full naive solution vector: y=[ 0          0          -0.5310248099636794
-0.8466378590550888 ]
classical state: y= [ 0.          0.         -0.53102481 -0.84663786]

On day 119 the results are
normalized vector imput [-0.53134823 -0.8471535   0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.5011663384963513
-0.8646475087227771 ]
classical state: y= [ 0.          0.         -0.50116634 -0.86464751]

On day 120 the results are
normalized vector imput [-0.50147157 -0.86517412  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.4706977685246762
-0.8816045734274736 ]
classical state: y= [ 0.          0.         -0.47069777 -0.88160457]

On day 121 the results are
normalized vector imput [-0.47098445 -0.88214151  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.4396561911849495
-0.8974884103625457 ]
classical state: y= [ 0.          0.         -0.43965619 -0.89748841]

On day 122 the results are
normalized vector imput [-0.43992396 -0.89803503  0.          0.        ]
square of the norm of vector imput 0.9999999999999998
full naive solution vector: y=[ 0          0          -0.4080793951681776
-0.9122796832230339 ]
classical state: y= [ 0.          0.         -0.4080794  -0.91227968]

On day 123 the results are
normalized vector imput [-0.40832794 -0.91283531  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.37600582071773386
-0.9259603857448376 ]
classical state: y= [ 0.          0.         -0.37600582 -0.92596039]

On day 124 the results are
normalized vector imput [-0.37623483 -0.92652434  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.34347451283384733
-0.9385138636247727 ]
classical state: y= [ 0.          0.         -0.34347451 -0.93851386]

On day 125 the results are
normalized vector imput [-0.34368371 -0.93908546  0.          0.        ]
```

```
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.3105250737418855
-0.9499248347948076 ]
classical state: y= [ 0.          0.         -0.31052507 -0.94992483]

On day 126 the results are
normalized vector imput [-0.3107142  -0.95050339  0.          0.        ]
square of the norm of vector imput 0.9999999999999998
full naive solution vector: y=[ 0          0          -0.277197614682299
-0.9601794080258063 ]
classical state: y= [ 0.          0.         -0.27719761 -0.96017941]

On day 127 the results are
normalized vector imput [-0.27736644 -0.9607642   0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.2435327070809092
-0.9692650998381136 ]
classical state: y= [ 0.          0.         -0.24353271 -0.9692651 ]

On day 128 the results are
normalized vector imput [-0.24368103 -0.96985543  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.20957133315898396
-0.9771708496984085 ]
classical state: y= [ 0.          0.         -0.20957133 -0.97717085]

On day 129 the results are
normalized vector imput [-0.20969897 -0.97776599  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.17535483604323465
-0.9838870334843309 ]
classical state: y= [ 0.          0.         -0.17535484 -0.98388703]

On day 130 the results are
normalized vector imput [-0.17546164 -0.98448627  0.          0.        ]
square of the norm of vector imput 0.9999999999999998
full naive solution vector: y=[ 0          0          -0.14092486943645446
-0.9894054752004673 ]
classical state: y= [ 0.          0.         -0.14092487 -0.98940548]

On day 131 the results are
normalized vector imput [-0.1410107  -0.99000807  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.10632334691008433
-0.9937194569314725 ]
classical state: y= [ 0.          0.         -0.10632335 -0.99371946]

On day 132 the results are
```

```
normalized vector imput [-0.1063881  -0.99432468  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.07159239088041551
-0.9968237270201628 ]
classical state: y= [ 0.          0.         -0.07159239 -0.99682373]

On day 133 the results are
normalized vector imput [-0.07163599 -0.99743084  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.0367742813305599
-0.9987145064606627 ]
classical state: y= [ 0.          0.         -0.03677428 -0.99871451]

On day 134 the results are
normalized vector imput [-0.03679668 -0.99932277  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          -0.0019114043406055555
-0.9993894934988009 ]
classical state: y= [ 0.          0.         -0.0019114  -0.99938949]

On day 135 the results are
normalized vector imput [-0.00191257 -0.99999817  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.03295379951138939
-0.9988478664341759 ]
classical state: y= [ 0.          0.          0.0329538  -0.99884787]

On day 136 the results are
normalized vector imput [ 0.03297387 -0.99945621  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.06777888681474571
-0.997090284620454 ]
classical state: y= [ 0.          0.          0.06777889 -0.99709028]

On day 137 the results are
normalized vector imput [ 0.06782017 -0.99769756  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.10252146299495603
-0.9941188876626974 ]
classical state: y= [ 0.          0.          0.10252146 -0.99411889]

On day 138 the results are
normalized vector imput [ 0.1025839  -0.99472436  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.1371392339230322
-0.989937292812705 ]
classical state: y= [ 0.          0.          0.13713923 -0.98993729]
```

```
On day 139 the results are
normalized vector imput [ 0.13722276 -0.99054021  0.          0.        ]
square of the norm of vector imput 0.9999999999999998
full naive solution vector: y=[ 0          0          0.17159005740257263
-0.9845505905655266 ]
classical state: y= [ 0.          0.          0.17159006 -0.98455059]

On day 140 the results are
normalized vector imput [ 0.17169456 -0.98515023  0.          0.        ]
square of the norm of vector imput 0.9999999999999998
full naive solution vector: y=[ 0          0          0.2058319944718708
-0.9779653384625017 ]
classical state: y= [ 0.          0.          0.20583199 -0.97796534]

On day 141 the results are
normalized vector imput [ 0.20595736 -0.97856097  0.          0.        ]
square of the norm of vector imput 0.9999999999999998
full naive solution vector: y=[ 0          0          0.2398233604586192
-0.9701895531084013 ]
classical state: y= [ 0.          0.          0.23982336 -0.97018955]

On day 142 the results are
normalized vector imput [ 0.23996942 -0.97078045  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.27352277572504813
-0.9612327004123442 ]
classical state: y= [ 0.          0.          0.27352278 -0.9612327 ]

On day 143 the results are
normalized vector imput [ 0.27368936 -0.96181814  0.          0.        ]
square of the norm of vector imput 0.9999999999999998
full naive solution vector: y=[ 0          0          0.30688921604173136
-0.9511056840644133 ]
classical state: y= [ 0.          0.          0.30688922 -0.95110568]

On day 144 the results are
normalized vector imput [ 0.30707613 -0.95168495  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.33988206252873027
-0.9398208322619589 ]
classical state: y= [ 0.          0.          0.33988206 -0.93982083]

On day 145 the results are
normalized vector imput [ 0.34008907 -0.94039323  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.37246115110328365
-0.9273918827017854 ]
classical state: y= [ 0.          0.          0.37246115 -0.92739188]
```

```
On day 146 the results are
normalized vector imput [ 0.372688   -0.92795671  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.40458682137385005
-0.9138339658564617 ]
classical state: y= [ 0.          0.          0.40458682 -0.91383397]

On day 147 the results are
normalized vector imput [ 0.40483323 -0.91439054  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.43621996492097437
-0.8991635865551324 ]
classical state: y= [ 0.          0.          0.43621996 -0.89916359]

On day 148 the results are
normalized vector imput [ 0.43648564 -0.89971122  0.          0.        ]
square of the norm of vector imput 0.9999999999999998
full naive solution vector: y=[ 0          0          0.46732207290620553
-0.8833986038912405 ]
classical state: y= [ 0.          0.          0.46732207 -0.8833986 ]

On day 149 the results are
normalized vector imput [ 0.4676067  -0.88393664  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.49785528295111475
-0.8665582094816322 ]
classical state: y= [ 0.          0.          0.49785528 -0.86655821]

On day 150 the results are
normalized vector imput [ 0.4981585  -0.86708599  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.527782425229338
-0.8486629041034991 ]
classical state: y= [ 0.          0.          0.52778243 -0.8486629 ]

On day 151 the results are
normalized vector imput [ 0.52810387 -0.84917978  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.5570670677155359
-0.8297344727376099 ]
classical state: y= [ 0.          0.          0.55706707 -0.82973447]

On day 152 the results are
normalized vector imput [ 0.55740635 -0.83023982  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.5856735605361876
-0.8097959580482059 ]
```

```
classical state: y= [ 0.          0.          0.58567356 -0.80979596]

On day 153 the results are
normalized vector imput [ 0.58603026 -0.81028916  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.6135670793682196
-0.7888716323318383 ]
classical state: y= [ 0.          0.          0.61356708 -0.78887163]

On day 154 the results are
normalized vector imput [ 0.61394077 -0.78935209  0.          0.        ]
square of the norm of vector imput 0.9999999999999998
full naive solution vector: y=[ 0          0          0.6407136678326559
-0.7669869679693208 ]
classical state: y= [ 0.          0.          0.64071367 -0.76698697]

On day 155 the results are
normalized vector imput [ 0.64110389 -0.7674541   0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.6670802788316593
-0.7441686064167322 ]
classical state: y= [ 0.          0.          0.66708028 -0.74416861]

On day 156 the results are
normalized vector imput [ 0.66748656 -0.74462184  0.          0.        ]
square of the norm of vector imput 0.9999999999999998
full naive solution vector: y=[ 0          0          0.6926348147786631
-0.7204443257732518 ]
classical state: y= [ 0.          0.          0.69263481 -0.72044433]

On day 157 the results are
normalized vector imput [ 0.69305666 -0.72088311  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.7173461666725944
-0.6958430069652782 ]
classical state: y= [ 0.          0.          0.71734617 -0.69584301]

On day 158 the results are
normalized vector imput [ 0.71778307 -0.69626681  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.7411842519686547
-0.6703945985880346 ]
classical state: y= [ 0.          0.          0.74118425 -0.6703946 ]

On day 159 the results are
normalized vector imput [ 0.74163567 -0.6708029   0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.7641200511995148
-0.6441300804474175 ]
classical state: y= [ 0.          0.          0.76412005 -0.64413008]

On day 160 the results are
normalized vector imput [ 0.76458544 -0.64452239  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.7861256433023845
-0.6170814258465087 ]
classical state: y= [ 0.          0.          0.78612564 -0.61708143]

On day 161 the results are
normalized vector imput [ 0.78660443 -0.61745726  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.8071742396089129
-0.5892815626626384 ]
classical state: y= [ 0.          0.          0.80717424 -0.58928156]

On day 162 the results are
normalized vector imput [ 0.80766585 -0.58964046  0.          0.        ]
square of the norm of vector imput 0.9999999999999998
full naive solution vector: y=[ 0          0          0.8272402164565782
-0.560764333262397 ]
classical state: y= [ 0.          0.          0.82724022 -0.56076433]

On day 163 the results are
normalized vector imput [ 0.82774405 -0.56110587  0.          0.        ]
square of the norm of vector imput 0.9999999999999998
full naive solution vector: y=[ 0          0          0.8462991463818378
-0.5315644533033791 ]
classical state: y= [ 0.          0.          0.84629915 -0.53156445]

On day 164 the results are
normalized vector imput [ 0.84681458 -0.5318882   0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.8643278278570765
-0.5017174694728258 ]
classical state: y= [ 0.          0.          0.86432783 -0.50171747]

On day 165 the results are
normalized vector imput [ 0.86485425 -0.50202304  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.8813043135351598
-0.4712597162146044 ]
classical state: y= [ 0.          0.          0.88130431 -0.47125972]

On day 166 the results are
normalized vector imput [ 0.88184107 -0.47154674  0.          0.        ]
square of the norm of vector imput 1.0
```

```
full naive solution vector: y=[ 0          0          0.8972079369671953
-0.44022827149720667 ]
classical state: y= [ 0.          0.          0.89720794 -0.44022827]

On day 167 the results are
normalized vector imput [ 0.89775438 -0.44049639  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.9120193377609854
-0.40866091167661645 ]
classical state: y= [ 0.          0.          0.91201934 -0.40866091]

On day 168 the results are
normalized vector imput [ 0.9125748  -0.40890981  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.9257204851495414
-0.37659606550898417 ]
classical state: y= [ 0.          0.          0.92572049 -0.37659607]

On day 169 the results are
normalized vector imput [ 0.92628429 -0.37682543  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.9382946999409713
-0.3440727673691005 ]
classical state: y= [ 0.          0.          0.9382947  -0.34407277]

On day 170 the results are
normalized vector imput [ 0.93886617 -0.34428232  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.9497266748230165
-0.3111306097316134 ]
classical state: y= [ 0.          0.          0.94972667 -0.31113061]

On day 171 the results are
normalized vector imput [ 0.95030511 -0.3113201   0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.9600024929975156
-0.2778096949728412 ]
classical state: y= [ 0.          0.          0.96000249 -0.27780969]

On day 172 the results are
normalized vector imput [ 0.96058718 -0.27797889  0.          0.        ]
square of the norm of vector imput 0.9999999999999998
full naive solution vector: y=[ 0          0          0.9691096451221284
-0.24415058655185223 ]
classical state: y= [ 0.          0.          0.96910965 -0.24415059]

On day 173 the results are
normalized vector imput [ 0.96969988 -0.24429929  0.          0.        ]
```

```
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.9770370445386632
-0.2101942596302395 ]
classical state: y= [ 0.          0.          0.97703704 -0.21019426]

On day 174 the results are
normalized vector imput [ 0.97763211 -0.21032228  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.9837750407695156
-0.17598205119071095 ]
classical state: y= [ 0.          0.          0.98377504 -0.17598205]

On day 175 the results are
normalized vector imput [ 0.98437421 -0.17608923  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.989315431265735
-0.14155560971520942 ]
classical state: y= [ 0.          0.          0.98931543 -0.14155561]

On day 176 the results are
normalized vector imput [ 0.98991797 -0.14164182  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.9936514713924665
-0.10695684448383029 ]
classical state: y= [ 0.          0.          0.99365147 -0.10695684]

On day 177 the results are
normalized vector imput [ 0.99425665 -0.10702199  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.9967778826395767
-0.0722278745562513 ]
classical state: y= [ 0.          0.          0.99677788 -0.07222787]

On day 178 the results are
normalized vector imput [ 0.99738497 -0.07227186  0.          0.        ]
square of the norm of vector imput 0.9999999999999998
full naive solution vector: y=[ 0          0          0.9986908590474894
-0.03741097749779005 ]
classical state: y= [ 0.          0.          0.99869086 -0.03741098]

On day 179 the results are
normalized vector imput [ 0.99929911 -0.03743376  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.9993880718403941
-0.0025485379124977 ]
classical state: y= [ 0.          0.          0.99938807 -0.00254854]

On day 180 the results are
```

```
normalized vector imput [ 0.99999675 -0.00255009  0.          0.        ]
square of the norm of vector imput 1.0
full naive solution vector: y=[ 0          0          0.9988686722612032
0.03231700415404584 ]
classical state: y= [0.         0.         0.99886867 0.032317  ]
```

From the results of the HHL algorithm and the classical algorithm we see that the results of $

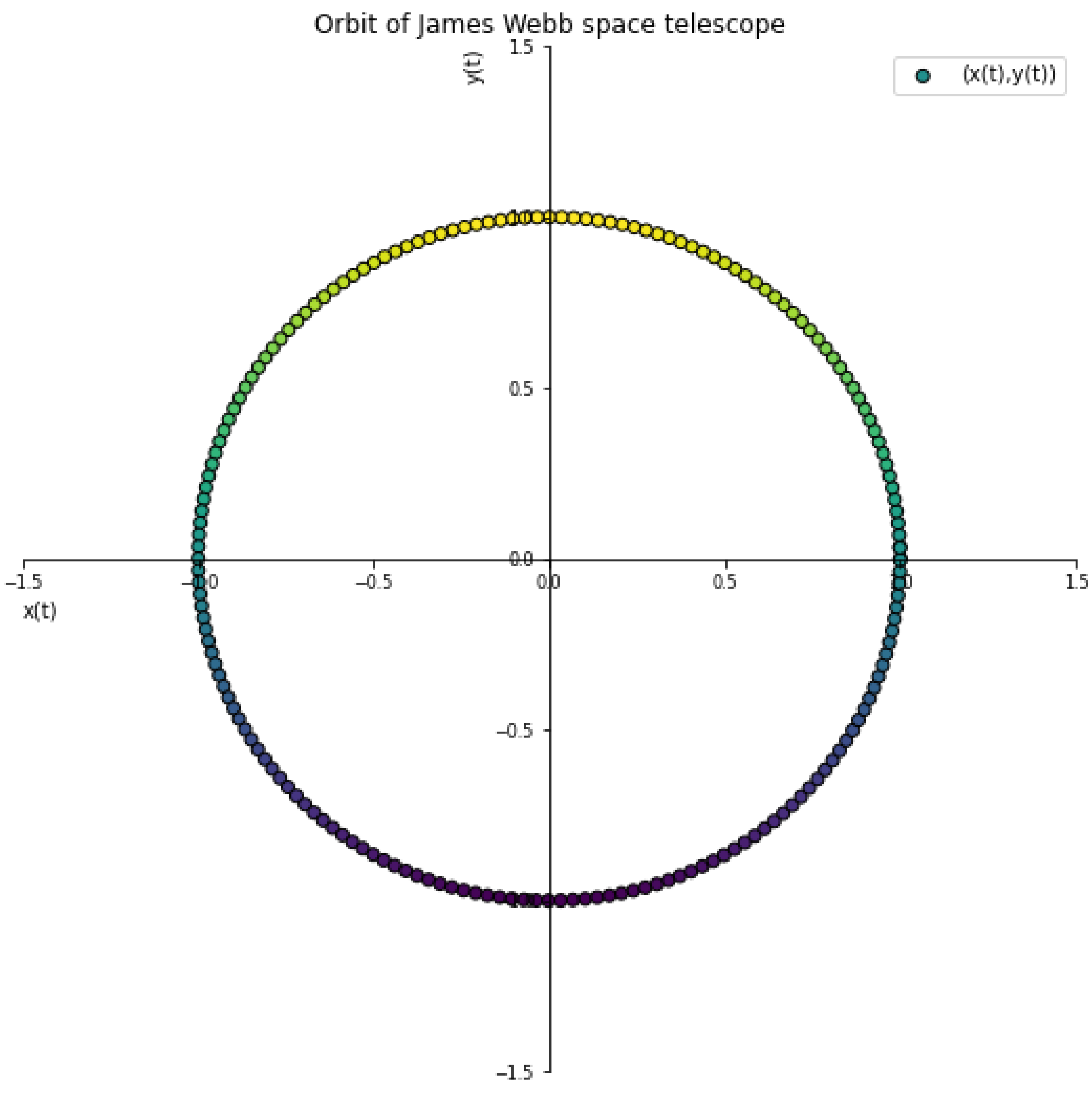
$$y = \begin{bmatrix} 0 \\ \vec{x} \end{bmatrix} \tag{10}$$

$ in each case are almost the same up to a global phase.

Plotting the results obtained form the HHL algorithm for the components of $\vec{x}$ on each time step we can simulate the orbit of the James Webb telescope using the implicit Euler method of integration on a quantum circuit.

[34]:
```python
from matplotlib import pyplot as plt

fig= plt.figure(figsize=(9, 9))
ax = fig.add_subplot(111)
ax.spines['left'].set_position('zero')
ax.spines['right'].set_color('none')
ax.spines['bottom'].set_position('zero')
ax.spines['top'].set_color('none')
plt.axis([-1.5,1.5,-1.5,1.5])
ax.xaxis.set_ticks_position('bottom')
ax.yaxis.set_ticks_position('left')
ax.set_ylabel('y(t)', loc='top')
ax.set_xlabel('x(t)', loc='left')
plt.xticks(fontsize=8)
plt.yticks(fontsize=8)
plt.title('Orbit of James Webb space telescope')
#plt.grid()
plt.scatter(xv1, yv1,c=cyv1, ec='k', label='(x(t),y(t))')
ax.legend()
# changing the size of figure to 2X2
```

[34]:
```
<matplotlib.legend.Legend at 0x7f23292cb550>
```



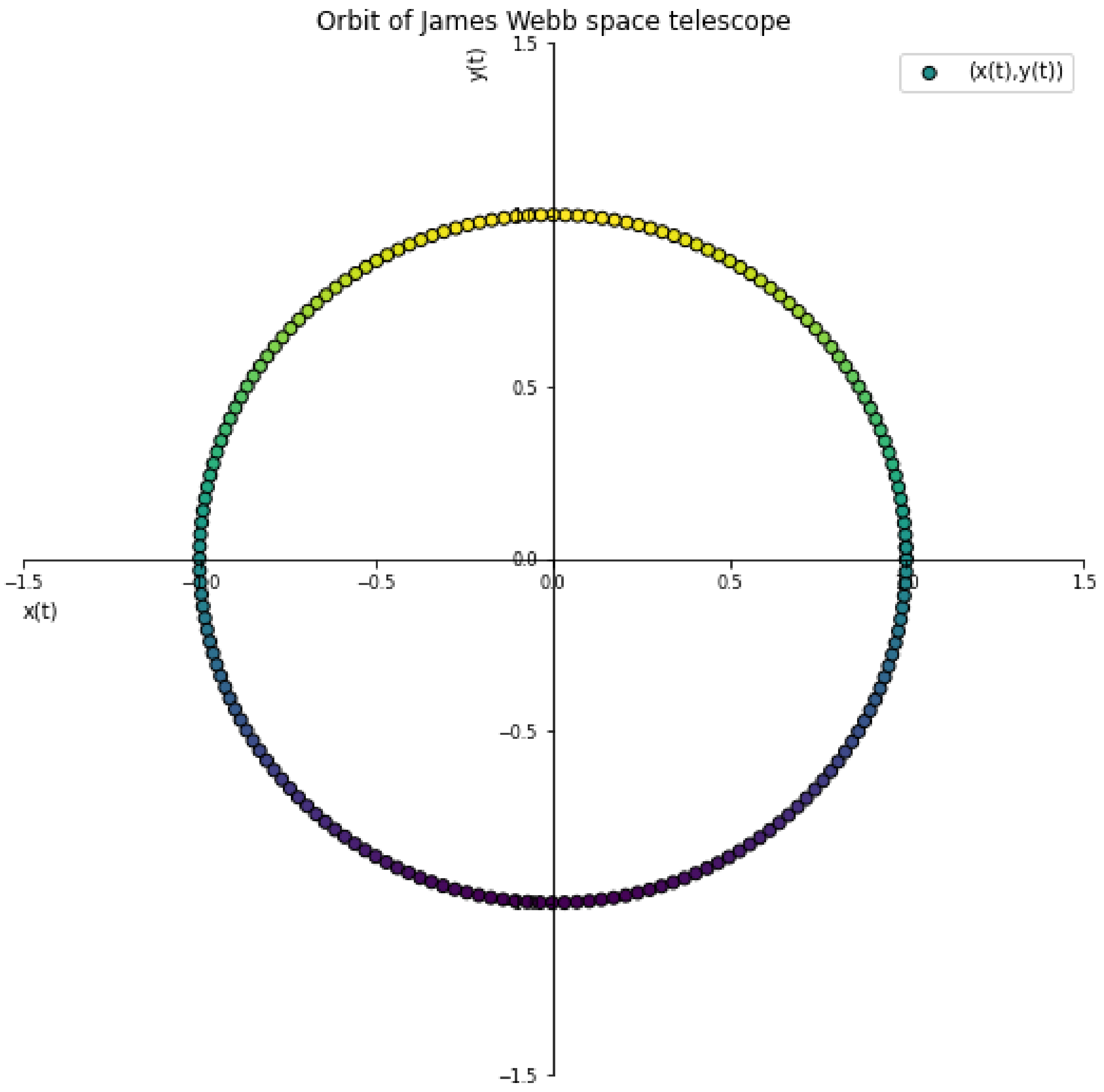
As in some cases we have a global phase in the full naive solution vector, we can cansider the following adjustments in order to avoid the global phase at the end figure.

[31]:
```
for i in range(len(cxv1)-1):
    x = np.sign(cxv1[i])
    y = np.sign(cyv1[i])
    x1 = np.sign(cxv1[i+1])
    y1 = np.sign(cyv1[i+1])
    if x>0:
        if y<0:
            if abs(cxv1[i]) < abs(cxv1[i+1]):
                if x != x1: cxv1[i+1] = -1*cxv1[i+1]
                if y != y1: cyv1[i+1] = -1*cyv1[i+1]
            elif (abs(cxv1[i]) < abs(cxv1[i+1])):
                if (y == y1): cyv1[i+1] = -1*cyv1[i+1]
                if (x != x1): cxv1[i+1] = -1*cxv1[i+1]

        elif y>0:
            if abs(cxv1[i]) > abs(cxv1[i+1]):
                if x != x1: cxv1[i+1] = -1*cxv1[i+1]
                if y != y1: cyv1[i+1] = -1*cyv1[i+1]
            elif (abs(cxv1[i]) < abs(cxv1[i+1])):
                if (y != y1): cyv1[i+1] = -1*cyv1[i+1]
                if (x == x1): cxv1[i+1] = -1*cxv1[i+1]

        elif y==0:
            if abs(cxv1[i]) > abs(cxv1[i+1]):
                if x != x1: cxv1[i+1] = -1*cxv1[i+1]
                if y1<0: cyv1[i+1] = -1*cyv1[i+1]


    elif x<0:
        if y<0:
            if abs(cxv1[i]) > abs(cxv1[i+1]):
                if x != x1: cxv1[i+1] = -1*cxv1[i+1]
                if y != y1: cyv1[i+1] = -1*cyv1[i+1]
            elif (abs(cxv1[i]) < abs(cxv1[i+1])):
                if (y != y1): cyv1[i+1] = -1*cyv1[i+1]
                if (x == x1): cxv1[i+1] = -1*cxv1[i+1]

        elif y>0:
            if abs(cxv1[i]) < abs(cxv1[i+1]):
                if x != x1: cxv1[i+1] = -1*cxv1[i+1]
                if y != y1: cyv1[i+1] = -1*cyv1[i+1]
            elif (abs(cxv1[i]) > abs(cxv1[i+1])):
                if (y == y1): cyv1[i+1] = -1*cyv1[i+1]
                if (x != x1): cxv1[i+1] = -1*cxv1[i+1]
```

[32]:
```
for i in range(len(xv1)-1):
    x = np.sign(xv1[i])
    y = np.sign(yv1[i])
    x1 = np.sign(xv1[i+1])
    y1 = np.sign(yv1[i+1])
    if x>0:
        if y<0:
            if abs(xv1[i]) < abs(xv1[i+1]):
                if x != x1: xv1[i+1] = -1*xv1[i+1]
                if y != y1: yv1[i+1] = -1*yv1[i+1]
            elif (abs(xv1[i]) < abs(xv1[i+1])):
                if (y == y1): yv1[i+1] = -1*yv1[i+1]
                if (x != x1): xv1[i+1] = -1*xv1[i+1]

        elif y>0:
            if abs(xv1[i]) > abs(xv1[i+1]):
                if x != x1: xv1[i+1] = -1*xv1[i+1]
                if y != y1: yv1[i+1] = -1*yv1[i+1]
            elif (abs(xv1[i]) < abs(xv1[i+1])):
                if (y != y1): yv1[i+1] = -1*yv1[i+1]
                if (x == x1): xv1[i+1] = -1*xv1[i+1]

        elif y==0:
            if abs(xv1[i]) > abs(xv1[i+1]):
                if x != x1: xv1[i+1] = -1*xv1[i+1]
                if y1<0: yv1[i+1] = -1*yv1[i+1]


    elif x<0:
        if y<0:
            if abs(xv1[i]) > abs(xv1[i+1]):
                if x != x1: xv1[i+1] = -1*xv1[i+1]
                if y != y1: yv1[i+1] = -1*yv1[i+1]
            elif (abs(xv1[i]) < abs(xv1[i+1])):
                if (y != y1): yv1[i+1] = -1*yv1[i+1]
                if (x == x1): xv1[i+1] = -1*xv1[i+1]

        elif y>0:
            if abs(xv1[i]) < abs(xv1[i+1]):
                if x != x1: xv1[i+1] = -1*xv1[i+1]
                if y != y1: yv1[i+1] = -1*yv1[i+1]
            elif (abs(xv1[i]) > abs(xv1[i+1])):
                if (y == y1): yv1[i+1] = -1*yv1[i+1]
                if (x != x1): xv1[i+1] = -1*xv1[i+1]
```

[33]:
```
from matplotlib import pyplot as plt

fig= plt.figure(figsize=(9, 9))
ax = fig.add_subplot(111)
ax.spines['left'].set_position('zero')
ax.spines['right'].set_color('none')
ax.spines['bottom'].set_position('zero')
ax.spines['top'].set_color('none')
plt.axis([-1.5,1.5,-1.5,1.5])
ax.xaxis.set_ticks_position('bottom')
ax.yaxis.set_ticks_position('left')
ax.set_ylabel('y(t)', loc='top')
ax.set_xlabel('x(t)', loc='left')
plt.xticks(fontsize=8)
plt.yticks(fontsize=8)
plt.title('Orbit of James Webb space telescope')
#plt.grid()
plt.scatter(xv1, yv1,c=cyv1, ec='k', label='(x(t),y(t))')
ax.legend()
# changing the size of figure to 2X2
```

```
[33]: <matplotlib.legend.Legend at 0x7f2328a84040>
```



As we have seen a quantum circuit can simulate the orbit of the James Webb telescope using the implicit Euler method of integration and the HHL algorithm. In order to do so, we have to consider valid quantum conditions. The matrix imput needs to be Hemitian and recalling that the vector imput is a Quantum state with real numbers it needs to satisfy the following conditions:

1) When a quantum system is measured, the probability of observing one state is the square of its value. The summation of amplitude squares must be 1 for a valid quantum state.

2) The overall probability must be 1 when we observe a quantum system. If we consider a quantum system as a vector, then the length of such vector should be 1.

We see that for some "full naive solution vectors" and "classical states" we have the same magnitude for their components but they differ by a - sign that comes from a "global phase" on the solution of the HHL algortithm (naive_hhl_solution), but we know that in quantum mechanics two vectors that differ by a global phase factor are considered equivalent. Nevertheless, as the solutions of the HHL algorithm can have a global phase, the new imput vector constructed from the solution will carry this global phase which also affects the new quantum and classical outcomes for the vector $\vec{x}$. Also we can get rid of this global phase on the "full naive solution vectors" by considering some adjustments before plotting our results.

We have plotted all the data for it's trayectory over the period of T=180 days= 6 months (30 days per month) performing a measurment every day. We found that the plotts of the $\vec{x} = (x(t), y(t))$ form a unitary circle, which we can consider as the trayectory of the James Webb telescope.

[ ]:

```
```