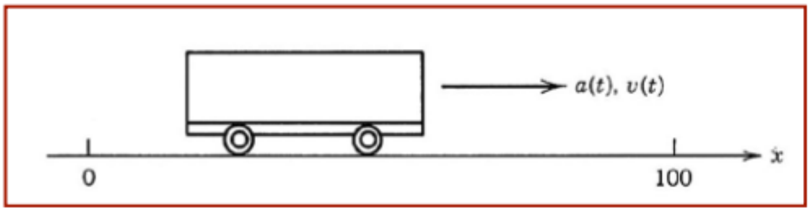
Cart on a track - simple linear form

Author: Sorin Moldoveanu
Affiliation: home
Status: under development

Introduction

We will try to find out the minimum (cosntant) force applied to a cart moving free on a track. In other words, use as little force as possible.
Example: we want to move the cart 100m in 10s. The initial velocity v is 0, the final velocity is free. The mass of the cart m is constant. The only control is the force f applied, namely the acceleration a.



In order to describe what happens, equations are:

$$\dot{x} = v$$
$$\dot{v} = f/m = a$$

Notations:
$x = x_1$
$v = x_2$
$a = u = k = f/m$

The function to minimize:
$J = q(x_{1f} - 100)^2 + r\int_0^{t_f} a^2 dt$

Working a little on J, considering $a = const$, the function comes to:
$J = q(50k - 100)^2 + 10rk^2$

In order to solve on a quantum computer, we put the function in a binary form, with a binary decision variable $x \in \{0, 1\}^n$ which indicate which control value to pick:
$J = q(50x^T K - 100)^2 + 10rx^T K^2$

subject to restriction to pick one control value only:
$x^T K = 1$

In the end, the objective is:
$min\left(q(50x^T K - 100)^2 + 10rx^T K^2 + p(1 - x^T K)^2\right)$

where K is a vector of possible control(acceleration) values (for this case considering a sigle time interval) and q, r, p are penalty terms (chose in such way in order to satisfy each objective)

The objective is mapped to an Issing Hamiltonian whose groundstate corresponds to the optimal solution. This is done using the following assignment:
$x \to (1 - Z_i)/2$ where Z_i is the Pauli **Z** operator that has eigenvalues = ± 1

The module for value and Pauli operator calculus is cargo.py.
We will implement the VQE quantum algorithm.
NOTE:

- still some work to do on this problem
- the analytical solutions are: K=0 (obvious) and K=2 for $q/r \to \infty$

In [1]:
```
#IMPORT
from qiskit import BasicAer
from qiskit_aqua import QuantumInstance
from qiskit_aqua import Operator, run_algorithm
from qiskit_aqua.input import EnergyInput
from qiskit_aqua.translators.ising import cart
from qiskit_aqua.algorithms import VQE, QAOA, ExactEigensolver
from qiskit_aqua.components.optimizers import COBYLA
from qiskit_aqua.components.variational_forms import RY
import numpy as np

#set for real device
#device = 'ibmq_16_melbourne'
from qiskit import IBMQ
IBMQ.load_accounts()
#backend = IBMQ.get_backend(device)

# define the problem
# variables
# n - number of acceleration elements to be taken into account
# K - acceleration vector; vector n x 1
# q - penalty
# p - penalty
# r - penalty

n = 5
q = 10**6
r = 1
p = 10
K=np.array([1.5,1.7,2,2.1,2.4])
e=np.ones(n)
E = np.matmul(np.asmatrix(e).T, np.asmatrix(e))

qubitOp, offset = cart.get_cart_qubitops(K, q, r, p)
algo_input = EnergyInput(qubitOp)
#print(offset, qubitOp)

#prepare some printing format
def index_to_selection(i, n):
    s = "{0:b}".format(i).rjust(n)
    x = np.array([1 if s[i]=='1' else 0 for i in reversed(range(n))])
    return x

def print_result(result):
    selection = cart.sample_most_likely(result['eigvecs'][0])
    #print('selection',selection)
    value = cart.cart_value(selection, K, q, r, p)
    print('\nvalue:',value)
    print('Optimal: selection {}, value {:.4f}'.format(selection, value))
    #print(selection,value)

    probabilities = np.abs(result['eigvecs'][0])**2
    i_sorted = reversed(np.argsort(probabilities))
    #i_sorted = reversed(np.argsort(value))
    print('\n----------------- Full result ---------------------')
    print('selection\tvalue\t\tprobability')
    print('---------------------------------------------------')
    for i in i_sorted:
        x = index_to_selection(i, n)
        #print('\nx=',x)
        value = cart.cart_value(x, K, q, r, p)
        probability = probabilities[i]
        print('%10s\t%.4f\t\t%.4f' %(x, value, probability))
        #if value>=0: print('%10s\t%.4f\t\t%.4f' %(x, value, probability))
        #if len(np.argwhere(x>0))==1: print('%10s\t%.4f\t\t%.4f' %(x, value,
probability))
```

In [2]:
```
#exact eigensolver - as reference
exact_eigensolver = ExactEigensolver(qubitOp, k=1)
result = exact_eigensolver.run()
#print(result)
#print(result['eigvecs'][0])
print('\n','EXACT_eigensolver')
print_result(result)
```

```
 EXACT_eigensolver

value: -10000000010.0
Optimal: selection [0 0 0 0 0], value -10000000010.0000

----------------- Full result ---------------------
selection       value           probability
---------------------------------------------------
[0 0 0 0 0]     -10000000010.0000               1.0000
[0 1 1 1 1]     -96100000347.8000               0.0000
[1 0 0 0 0]     -624999980.0000         0.0000
[0 1 0 0 0]     -224999976.0000         0.0000
[1 1 0 0 0]     -3599999997.0000                0.0000
[0 0 1 0 0]     30.0000         0.0000
[1 0 1 0 0]     -5625000000.0000                0.0000
[0 1 1 0 0]     -7225000004.0000                0.0000
[1 1 1 0 0]     -25600000085.0000               0.0000
[0 0 0 1 0]     -24999968.0000          0.0000
[1 0 0 1 0]     -6400000001.0000                0.0000
[0 1 0 1 0]     -8100000005.4000                0.0000
[1 1 0 1 0]     -27225000089.4000               0.0000
[0 0 1 1 0]     -11025000012.0000               0.0000
[1 0 1 1 0]     -32400000105.0000               0.0000
[0 1 1 1 0]     -36100000117.4000               0.0000
[1 1 1 1 1]     -148225000563.8000              0.0000
[0 0 0 0 1]     -399999962.0000         0.0000
[1 0 0 0 1]     -9025000004.0000                0.0000
[0 1 0 0 1]     -11025000009.6000               0.0000
[1 1 0 0 1]     -32400000102.6000               0.0000
[0 0 1 0 1]     -14400000018.0000               0.0000
[1 0 1 0 1]     -38025000120.0000               0.0000
[0 1 1 0 1]     -42025000133.6000               0.0000
[1 1 1 0 1]     -78400000286.6000               0.0000
[0 0 0 1 1]     -15625000020.8000               0.0000
[1 0 0 1 1]     -40000000125.8000               0.0000
[0 1 0 1 1]     -44100000139.8000               0.0000
[1 1 0 1 1]     -81225000295.8000               0.0000
[0 0 1 1 1]     -50625000160.8000               0.0000
[1 0 1 1 1]     -90000000325.8000               0.0000
[1 1 1 1 0]     -70225000261.4000               0.0000
```

In [3]:
```
#VQE
backend = BasicAer.get_backend('statevector_simulator')
seed = 50

cobyla = COBYLA()
cobyla.set_options(maxiter=250)
ry = RY(qubitOp.num_qubits, depth=3, entanglement='full')
vqe = VQE(qubitOp, ry, cobyla, 'matrix')
vqe.random_seed = seed

quantum_instance = QuantumInstance(backend=backend, seed=seed, seed_mapper=s
eed)

result = vqe.run(quantum_instance)
print('\n','VQE')
print_result(result)
```

```
 VQE

value: -10000000010.0
Optimal: selection [0 0 0 0 0], value -10000000010.0000

----------------- Full result ---------------------
selection	value		probability
---------------------------------------------------
[0 0 0 0 0]	-10000000010.0000		0.7741
[1 1 1 1 1]	-148225000563.8000		0.2107
[0 0 1 0 0]	30.0000		0.0047
[0 1 1 1 1]	-96100000347.8000		0.0033
[1 1 0 1 1]	-81225000295.8000		0.0031
[1 0 0 0 0]	-624999980.0000		0.0014
[1 0 1 0 0]	-5625000000.0000		0.0011
[0 1 0 0 0]	-224999976.0000		0.0007
[0 0 1 1 1]	-50625000160.8000		0.0002
[1 0 1 1 1]	-90000000325.8000		0.0001
[0 1 0 1 1]	-44100000139.8000		0.0001
[1 0 0 1 1]	-40000000125.8000		0.0001
[0 0 0 1 0]	-24999968.0000		0.0001
[1 1 1 0 1]	-78400000286.6000		0.0001
[1 1 1 0 0]	-25600000085.0000		0.0000
[0 0 0 1 1]	-15625000020.8000		0.0000
[0 1 1 0 1]	-42025000133.6000		0.0000
[1 1 1 1 0]	-70225000261.4000		0.0000
[1 1 0 0 0]	-3599999997.0000		0.0000
[0 1 0 0 1]	-11025000009.6000		0.0000
[1 0 0 1 0]	-6400000001.0000		0.0000
[0 0 0 0 1]	-399999962.0000		0.0000
[1 0 1 1 0]	-32400000105.0000		0.0000
[1 0 0 0 1]	-9025000004.0000		0.0000
[0 1 1 0 0]	-7225000004.0000		0.0000
[0 0 1 0 1]	-14400000018.0000		0.0000
[0 1 0 1 0]	-8100000005.4000		0.0000
[1 1 0 0 1]	-32400000102.6000		0.0000
[0 0 1 1 0]	-11025000012.0000		0.0000
[0 1 1 1 0]	-36100000117.4000		0.0000
[1 1 0 1 0]	-27225000089.4000		0.0000
[1 0 1 0 1]	-38025000120.0000		0.0000
```

In [4]:
```
#QAOA
backend = BasicAer.get_backend('statevector_simulator')
seed = 50

cobyla = COBYLA()
cobyla.set_options(maxiter=250)
qaoa = QAOA(qubitOp, cobyla, 3, 'matrix')
qaoa.random_seed = seed

quantum_instance = QuantumInstance(backend=backend, seed=seed, seed_mapper=s
eed)

result = qaoa.run(quantum_instance)
print('\n','QAOA')
print_result(result)
```

```
 QAOA

value: 5062944.2
Optimal: selection [0 0 1 1 1], value 5062944.2000

----------------- Full result --------------------
selection       value           probability
---------------------------------------------------
[0 0 1 1 1]     5062944.2000            0.1588
[0 0 0 1 1]     1562724.2000            0.1059
[0 0 1 0 0]     50.0000         0.0864
[1 1 1 1 0]     7023032.4000            0.0812
[0 1 0 1 0]     810151.4000             0.0697
[1 0 0 1 0]     640134.2000             0.0649
[0 0 1 0 1]     1440213.2000            0.0580
[0 0 0 0 0]     1000010.0000            0.0455
[1 0 1 1 1]     9000654.2000            0.0390
[0 1 0 0 1]     1102682.6000            0.0323
[0 1 0 0 0]     22533.8000              0.0281
[1 0 0 1 1]     4000374.2000            0.0276
[1 1 1 0 0]     2560267.8000            0.0227
[1 1 1 1 1]     14823450.0000           0.0217
[1 1 0 1 0]     2722780.4000            0.0202
[0 0 1 1 0]     1102680.2000            0.0168
[1 0 1 0 0]     562625.0000             0.0148
[1 1 1 0 1]     7840584.6000            0.0138
[1 0 0 0 1]     902664.2000             0.0128
[0 1 1 1 0]     3610343.4000            0.0127
[1 1 0 1 1]     8123102.0000            0.0102
[1 0 1 0 1]     3802860.2000            0.0095
[0 1 1 0 1]     4202886.6000            0.0091
[0 0 0 0 1]     40077.2000              0.0089
[1 0 1 1 0]     3240318.2000            0.0085
[0 1 1 1 1]     9610689.0000            0.0083
[0 1 1 0 0]     722641.8000             0.0073
[1 0 0 0 0]     62525.0000              0.0039
[0 1 0 1 1]     4410401.0000            0.0013
[0 0 0 1 0]     2556.2000               0.0001
[1 1 0 0 1]     3240320.6000            0.0000
[1 1 0 0 0]     360099.8000             0.0000
```

In [ ]: