# NASA Space Math

## Using Quantum Algorithms Simulator to solve Integrals

This project is using Qiskit.

Therefore, for more information about how Quantum Gates are implementing their functions using building blocks like Hadamard gates, etc., refer to this documentation.



In [1]:
```
from qiskit import QuantumCircuit, Aer, execute, transpile, execute, IBMQ
from qiskit_aer import AerSimulator

from qiskit.tools.monitor import job_monitor
from qiskit.circuit.library import WeightedAdder

from qiskit.circuit import Instruction, CircuitInstruction, Qubit, QuantumRegister, Clbi
from qiskit.circuit.library.standard_gates import IGate, XGate, CXGate, CCXGate, C3XGate
                                                  RXGate, RYGate, RZGate, HGate
from qiskit.exceptions import QiskitError

import matplotlib.pyplot as plt
%matplotlib inline
from qiskit.visualization import plot_histogram

import os
import sys
import math as m
import numpy as np
import pandas as pd
import sympy

from fractions import Fraction

from _functools import *
from traceback import format_exc
```

In [2]:
```
def initGates(circuit, qreq, nInputs, gateName='id'):
    '''Determine Input Value (Either 0 or 1)
    Initialization Gates
    First, zero it out at the beginning.
    All qubits start from ground state |0>. Create manipulable initialization gates as m
    The identity gate means that it remains the same state as previous, which, in this c
    Later, the Identity gate can be converted to an X Gate or NOT gate.'''

    circuit.data = [CircuitInstruction(operation=Instruction(name=gateName, num_qubits=1
                                       qubits=(Qubit(qreq, inputIndex),),
                                       clbits=()) for inputIndex in range(nInputs)]
```

In [3]:
```
def qAdd(weights, backend=AerSimulator(method='matrix_product_state')):
    aCirc1 = WeightedAdder(num_state_qubits=len(weights), weights=weights)

    nQubits = aCirc1.num_qubits
    qubits = aCirc1.qubits

    sumQubitIndices = [qubitIndex for qubitIndex in range(len(qubits)) if "'sum'" in str
```

```python
    nInputs = str(qubits).count("'state'")
    nOutputs = len(sumQubitIndices)

    q = QuantumRegister(nQubits, 'q')
    c = ClassicalRegister(nOutputs, 'c')

    aCirc = QuantumCircuit(q, c)

    initGates(aCirc, q, nInputs, gateName='x')

    aCirc.append(aCirc1, range(nQubits))
    aCirc.measure(sumQubitIndices, range(nOutputs))

    job = execute(aCirc, backend, shots=2000)
    result = job.result()
    counts = result.get_counts()
    count = list(counts)[0]

    return int(count, 2)
```

In [4]:
```python
def qintegrals(var, RawEqParts, debugPrint=False):
    Eqn_Integrated = []
    if debugPrint: factorsSet = []

    for RawEqPart in RawEqParts:
        coeffExp = RawEqPart.as_coeff_exponent(var)

        if debugPrint: print(coeffExp)

        coeffExpFrac = [Fraction(str(eqNum)).limit_denominator().as_integer_ratio() for

        # # Add 1 due to integration
        # Add by a value equal to the denominator of the 2nd/Divisor Fraction (e.g. if d

        factors = np.transpose(coeffExpFrac)
        denominator = factors[1, 1]

        # factors[:, 1][0] is the Numerator of the 2nd Fraction
        factors[:, 1][0] += denominator

        # # Fraction division is a multiplication by its reciprocal of the 2nd/Divisor F
        factors[:, 1] = factors[:, 1][::-1]
        result = []

        if debugPrint: factorsSet.append(factors)

        for A, B in factors:
            # Using Repeated Addition through WeightedAdder
            resultMul = qAdd([A]*B)
            result.append(resultMul)

        # New Coefficient for the integrated expression
        integCoef = Fraction(result[0], result[1])

        # New Exponent for the integrated expression
        # Add by a value equal to the denominator of the 2nd/Divisor Fraction (e.g. if d
        integExp = coeffExpFrac[1][0]+denominator
        integExp /= denominator

        Eqn_Integrated.append(integCoef*var**integExp)

        if debugPrint: print(factorsSet)

    return Eqn_Integrated
```

Calculations for: NASA Space Math 10Page120.pdf - Problem 2

PDF Document: https://spacemath.gsfc.nasa.gov/Calculus/10Page120.pdf

In [5]:
```
T = sympy.Symbol('T')
RawEqParts = [sympy.Mul(49), 42*T, 9*T**2]

Eqn_Integrated = qintegrals(T, RawEqParts)

print(Eqn_Integrated)
```

```
[49*T**1.0, 21*T**2.0, 3*T**3.0]
```

Calculations for: NASA Space Math 7Page48.pdf - Problem 1B

PDF Document: https://spacemath.gsfc.nasa.gov/Calculus/7Page48.pdf

In [6]:
```
T = sympy.Symbol('T')
RawEqParts = [float(coef)*T**exp for coef, exp in zip('1.49 12.30 41.94 76.00 77.55 42.2

Eqn_Integrated = qintegrals(T, RawEqParts)

print(Eqn_Integrated)
```

```
[149*T**9.0/900, 123*T**8.0/80, 2097*T**7.0/350, 38*T**6.0/3, 1551*T**5.0/100, 1057*T**
4.0/100, 473*T**3.0/150, 9*T**2.0/100, 9*T**1.0/10000]
```

Calculations for: NASA Space Math 10Page113.pdf - Problem 2

PDF Document: https://spacemath.gsfc.nasa.gov/Calculus/10Page113.pdf

In [7]:
```
m = sympy.Symbol('m')
RawEqParts = [0.025*m**-0.9]

Eqn_Integrated = qintegrals(m, RawEqParts)

print(Eqn_Integrated)
```

```
[m**0.1/4]
```