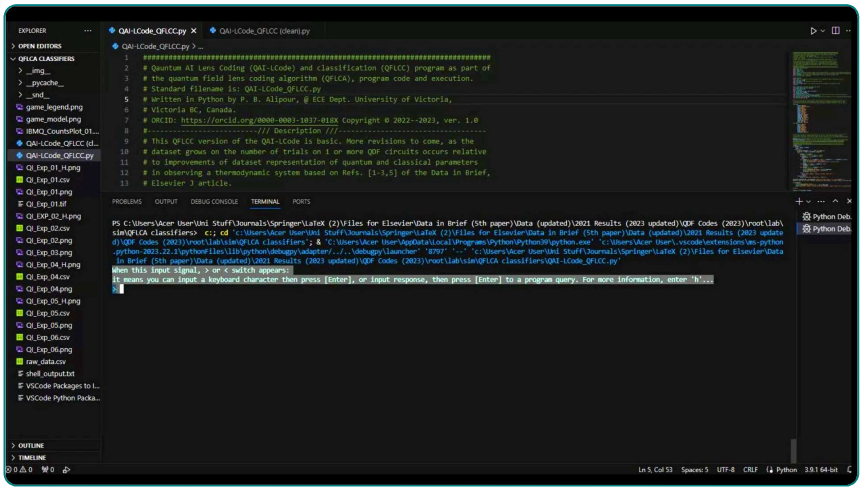
QFLCA Workflow and Code



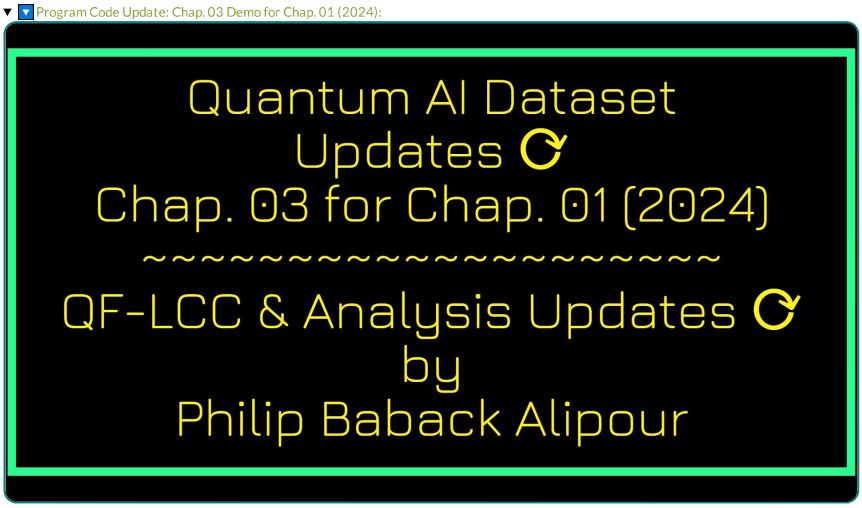
QFLCA Intro Demo: 337 MB mp4 file. ▽ | For Subtitles: SRT file. ▽

▼ ▼ i ␗ QFLCC program description (Start of Heading and Text) ␂ . . .

QFLCC as part of QFLCA: program description from QAI-LCode_QFLCC.py

```
1   ###############################################################################
2   # Quantum AI Lens Coding (QAI-LCode) and Classification (QFLCC) program as part of
3   # the quantum field lens coding algorithm (QFLCA), program code and execution.
4   # Standard filename is: QAI-LCode_QFLCC.py
5   # Written in Python by P. B. Alipour, @ ECE Dept. University of Victoria,
6   # Victoria BC, Canada.
7   # ORCID: https://orcid.org/0000-0003-1037-018X Copyright © 2022--2024, ver. 1.0
8   #--------------------------/// Description ///----------------------------------
9   # This QFLCC version of the QAI-LCode is basic. More revisions to come, as the
10  # dataset grows on the number of trials on 1 or more QDF circuits occurs relative
11  # to improvements of dataset representation of quantum and classical parameters
12  # in observing a thermodynamic system based on Refs. [1-3,5] of the Data in Brief,
13  # Elsevier J article.
14  ###############################################################################
```

▼ ▼ Program Code Update: Chap. 03 Demo for Chap. 01 (2024):



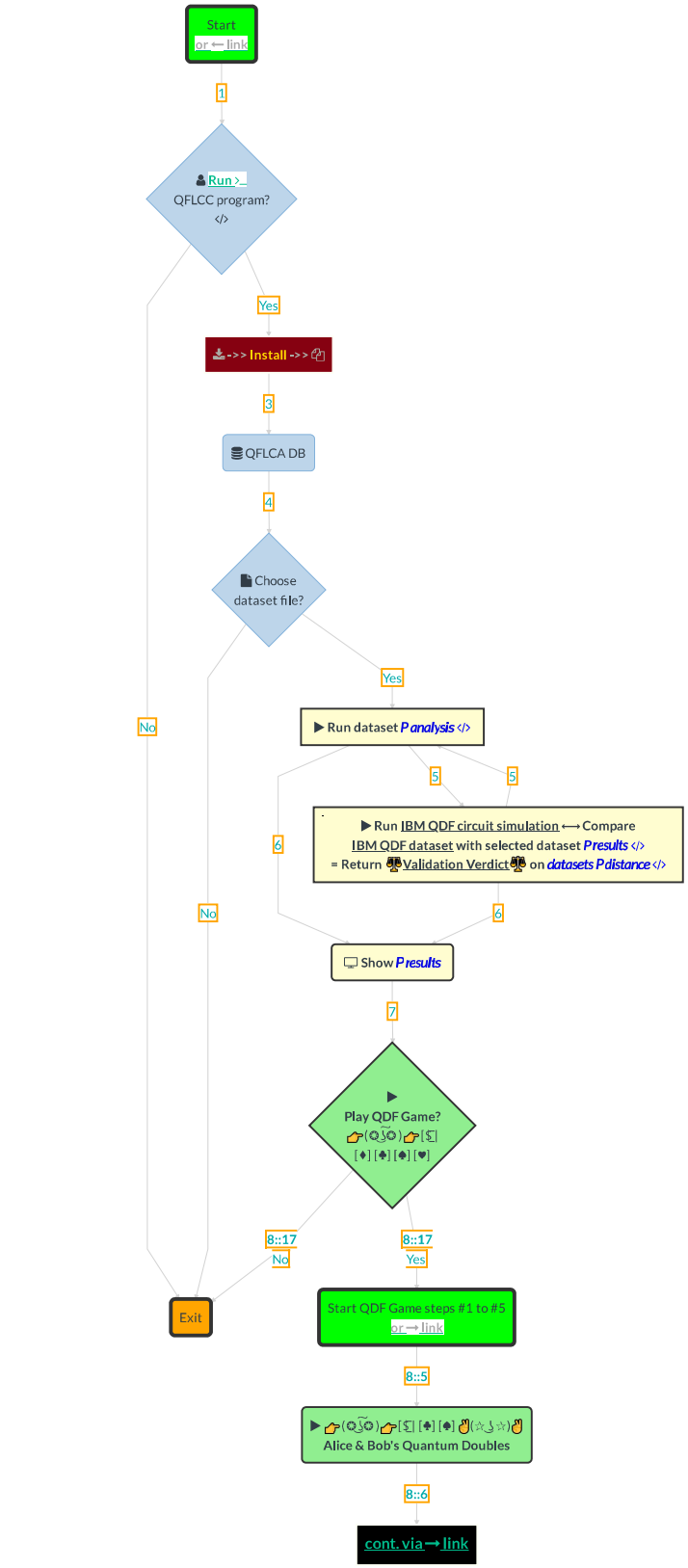
QFLCC 2024 Code Update: 616 MB mp4 file. ▽ | For Subtitles: SRT file. ▽

▼ ▼ i Expand/Collapse All. . .

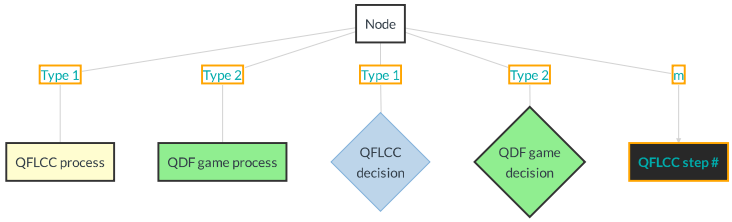
* Click ▼Expand All to view all code blocks from QAI-LCode_QFLCC.py on this page.
* Click ▶Collapse All to selectively view one or more code blocks from QAI-LCode_QFLCC.py on this page.

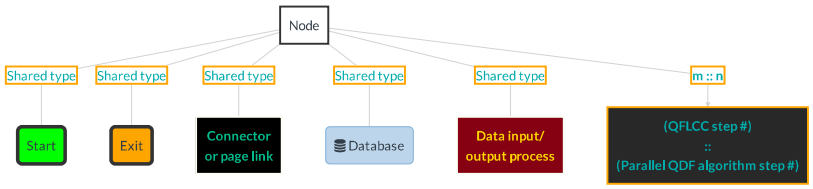
▼Expand All ▶Collapse All

Program execution flowchart



▼ Legend...





▼ ▼ i Flowchart description...

This 8-step diagram represents the flow of QFLCC steps of the complete QFLCA program (17 steps) after a successful installation and program run by the user. The final steps are presented in parallel, side-by-side, between programs/algorithms labeled as `m::n` e.g., [`8::17`] denotes `step 8 (from the QFLCC algorithm)::17 (as part of the QDF game algorithm)` after a successful dataset installation and analysis by the program. This program starts with dataset files installation, then the selection of one of the datasets for probability `P analysis`. From there, the IBM QDF circuit simulation module: QDF-LCode_IBMQ-2024.py is imported and run by the `QAI-LCode_QFLCC.py` program using Qiskit packages in an isolated (virtual) environment on your computer. The simulation is called via PAnalysis_model ( ) function within `QAI-LCode_QFLCC.py` combined with operations to compute `dataset P's`.

- Alternatively, run QDF-LCode_IBMQ-2024.ipynb file on `external servers` and `quantum computer providers` from Jupyterlab workspace within the IBMQ lab (more limited).

The simulation results are compared to reach a `validation verdict` between the circuit's dataset and the selected one to measure their `P distance(s)`. Finally, the program `prompts` `the user to continue` by playing the QDF game for `further validation of dataset results` (`P analysis`). The diagram continues on the relevant page linked here.

▼ ▼ i Expand/Collapse All...

```
* Click ▼Expand All to view all code blocks from QAI-LCode_QFLCC.py on this page.
* Click ▶Collapse All to selectively view a code block from QAI-LCode_QFLCC.py on this page.
```

▼Expand All ▶Collapse All

QDF Circuit Files for QFLCA

The following `QDF circuit source codes` generate the QFLCA dataset based on the `QDF circuit design and implementation`, or see Ref. [1] on the About page. The output after each program execution on IBMQ, otherwise, the QInspire platform, given the circuit's configuration from Ref. [1], is later processed as a set of datasets (in e.g., `*.csv`, `*.htm`) as input(s) by the `QAI-LCode_QFLCC.py` program. For more about the objectives of this program visit Project Objectives on the About page, or see below for the detailed codes as commented within the program's body. Future updates will include ways of processing and updating the circuit directly by a customized QDF circuit configuration option for the user. This will be available through `CLI` and/or `GUI`. For example, the QDF game program, flowchart, its CLI and GUI example are the preliminary representation of the `QFLCA project objectives`. The aim is to give the user the option to process new outputs as `user's datasets` by the `QFLCA program`. The program's objective will remain for it to determine and classify probabilities according to the user's `desired Hamiltonian` (target state as circuit's expected outcome), or see Ref. [1].

▼ ▼ i ≙ IBMQ source codes description and packages from filtered .py from raw .ipynb source code (see QDF-LCode_IBM-2024-raw.ipynb or QDF-LCode_IBM-2024.py to compile in VSC and your virtual qiskit environment). Also refer to code comments for installation notes:

IBMQ source code packages to draw the quantum circuit and count P's from QDF-LCode_IBM-2024.py

```
#-------------------------------------------------------------------------------------
# Quantum simulation of a quantum field lens coding algorithm with entanglement scaling
# between multi-well barrier of internal system B and external system A.
# Created by: Philip B. Alipour, Supervisor: T. A. Gulliver, at the University of Victoria,
# Dept. ECE, Victoria BC, Canada.
# Code updated based on Qiskit 2023--2024 changes specified in code comments below.
# Side-notes: You can also run code with the right packages installed in pipx or python.
# Examples of changes can be found on e.g.,
# https://docs.quantum.ibm.com/api/migration-guides/qiskit-1.0-installation and
# https://docs.quantum.ibm.com/api/migration-guides/qiskit-1.0-features
# follow the link to fix 'deprecated release' lines, use
# <pip install "qiskit-aer>=0.11.0">
# and <pip install qiskit-ibm-provider> after activating your qiskit virtual environment in
# shell.
#-------------------------------------------------------------------------------------
# Import the QISKit SDK... INstall Qiskit v.1 and update older ones < v.0.1 from above notes.
import termplotlib as tpl # To draw plots on circuit results during/after experiment.
import numpy as np
import sys
import re  # For search and spot regular expressions in {text, metadata, binary,...} I/O files.
from time import sleep
from qiskit import QuantumCircuit, ClassicalRegister, QuantumRegister
from qiskit import *
from qiskit_aer import Aer  # 2024 update... from its deprecated release.
from colorama import Fore, Back, Style  # For colored text messages.
#-------------------------------------------------------------------------------------
# Newly added package in 2023/2024, the U3 gate has been renamed the U gate.
# from https://quantum-computing.ibm.com/composer/docs/iqx/operations_glossary
# and for cu1 gate to recreate this gate, add the control modifier to the phase gate
# (formerly the U1 gate); (older version of Qiskit cu1, now cp)
# U3 should be renamed just U as it is an arbitrary single-qubit gate.
# CU1 should be renamed Cphase or CP depending on the above. For more, visit
# https://github.com/Qiskit/qiskit-terra/issues/4106
#-------------------------------------------------------------------------------------
from math import pi
from qiskit.compiler import transpile, assemble # In 2024, execute now is renamed to transpile.
from qiskit.visualization import *
from qiskit.providers.backend import Backend # Specify backend device for data processing
from qiskit_ibm_provider import IBMProvider  # 2024 update... from its deprecated release.
```

▼ ▼ i ≙ IBMQ source codes... cont.

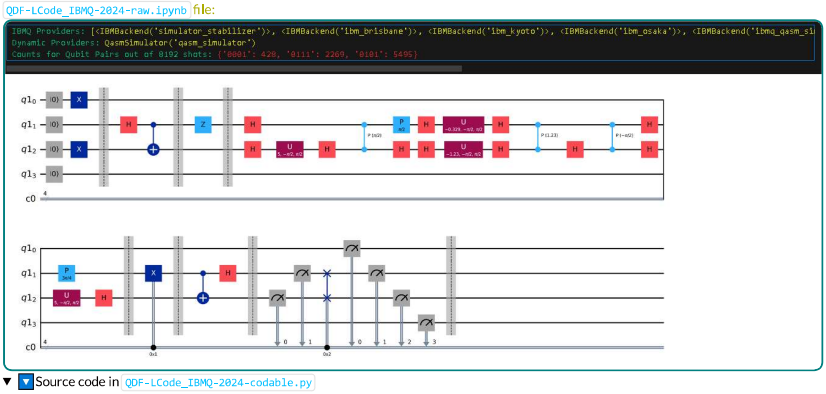
▼ ▼ qdf_circuit ( ) function code to compose, draw and run QDF measurements:

i QDF default circuit configuration and experiment to generate P results for the QFLCA program.

- Screenshot of the circuit simulation after executing the `qdf_circuit()` and `print(qc.draw())` functions:

- Executable under Windows OS: runs the IBMQ circuit simulator (compiled from the `QDF-LCode_IBM-2024.py` file) in the `qflcc classifiers\__pycache__` directory (click here or the following screenshot to download and run this program):

- Screenshot of the circuit simulation after executing the `qdf_circuit()` function based on the `QDF-LCode_IBMQ-2024-raw-codable.ipynb` or `QDF-LCode_IBMQ-2024-raw.ipynb` file:



- ▼ Source code in `QDF-LCode_IBMQ-2024-codable.py`

```
def qdf_circuit():  # Function to compose/build a QDF circuit.
 global qc    # Instantiate qc as a global variable to represent the quantum circuit.
 # Set your API Token. IBMQ.enable_account ('API Token') Create a Quantum Register with
 # 4 qubits.
 q = QuantumRegister(4)

 # Create a Classical Register with 4 bits.
 c = ClassicalRegister(4) # as _b in Fig. 6 of Ref. [1]

 # Create a Quantum Circuit.
 qc = QuantumCircuit(q, c)

 #-------------------------------------------------------------------------------------
 # Implementation of Superdense Coding
 #-------------------------------------------------------------------------------------
 # State initialization on 4 qubits in the z-basis in the |0> state.
 qc.reset(q[0])
 qc.reset(q[1])
 qc.reset(q[2])
 qc.reset(q[3])
 qc.x(q[0])
 qc.x(q[2])
 qc.barrier(range(4)) # Barrier to indicate physical barriers between systems A and B
                      # particles in communication.
 qc.h(q[1])
 qc.cx(q[1], q[2])
 qc.barrier(range(4))
 qc.z(q[1])   # Current state and encoded in the message through QFT.
 #qc.x(q[1])  # Current message encoded in the superdense coding scheme
 qc.barrier(range(4))

 # Time step = 1. Time evolution of potential energy part of the Ising model (spin
 # configuration tending to ground state or magnetization M>0) relative to kappaPhi
 # implementation.
 qc.h(q[2])
 qc.u(5,-pi/2,pi/2,q[2])
 qc.h(q[2])

 #-------------------------------------------------------------------------------------
 # Entanglement Encoder Implementation
 #-------------------------------------------------------------------------------------
 # Two-qubit Inverse Quantum Fourier Transform (QFT^-1)
 qc.h(q[1])
 qc.cp(pi/2, q[2], q[1])
 qc.h(q[2])

 # Time evolution of kinetic Ising model (kinetic energy part satisfying magnetization
 # value M=0)
 qc.p(pi/2, q[1])
 qc.h(q[1])
 qc.u(-(pi**2)/30, -pi/2, pi/2, q[1])
 qc.h(q[1])
 qc.h(q[2])
 qc.u(-(pi**2)/8, -pi/2, pi/2, q[2])
 qc.h(q[2])
 qc.cp((pi**2)/8, q[2], q[1])

 # Two-qubit Quantum Fourier Transform (QFT)
 qc.h (q[2])
 qc.cp (-pi/2, q[2], q[1])
 qc.h (q[1])

 # Time evolution of potential energy part
 qc.p(3*pi/4, q[1])
 qc.h (q[2])
 qc.u (5, -pi/2, pi/2, q[2])
 qc.h (q[2])
 qc.barrier(range(4))

 #-------------------------------------------------------------------------------------
 # Continuation of the superdense code algorithm. Remove IF statement for real quantum
 # computers when enabling one of the backends after 'qasm_simulator' below
 #-------------------------------------------------------------------------------------
 qc.x(q[1]).c_if(c, 1)  # IF statement , as if the prize is spotted via Eve or the
 # audience, ask Bob to decide to win the prize or a prize of lesser quality or energy
 # value; flipping condition for Alice to cloak the prize is 0 as opposed to 1 or 2 in
 # decimal. Remove IF statement when testing on a quantum computer and not a simulator.
 qc.barrier(range(4))
 qc.cx(q[1], q[2])
 qc.h(q[1])
 qc.barrier(range(4))
 qc.measure(q[2], c[0])  # Qubit 2 is in state |0>
 qc.measure(q[1], c[1])  # Qubit 1 is in state |1>

 # Remove IF statement for real quantum computers when enabling one of the backends after
 # 'qasm_simulator' below qc.swap(q[1],q[2]).c_if(c, 1) # SWAP gate is used if
 # condition c=1 or 01 in binary.
 qc.swap(q[1],q[2]).c_if(c, 2)   # SWAP gate is used if condition c=2 or 10 in binary
 qc.measure(q, c)  # Map the quantum measurement to the classical bits
 qc.barrier(range(4))

 global hline
 hline = "=============================================================================================================================================="
 print(f"{Fore.LIGHTGREEN_EX + hline}\
 \nQuantum simulation of a quantum field lens coding algorithm with entanglement scaling between a\
 \nmulti-well (barrier) interaction potential of internal system B interacting and external system A\
 from the method article: \
 \n{Fore.LIGHTCYAN_EX}https://www.sciencedirect.com/science/article/pii/S221501612300136X{Fore.LIGHTGREEN_EX} \
 \n*- QDF Circuit Built and Simulator generates its realtime datasets from\
 {Fore.LIGHTYELLOW_EX}{{ Qiskit Aer, IBMQ }} Qasm_Simulator{Fore.LIGHTGREEN_EX}\
 \n   Workspace and/or Virtual Environment on user's computer in Python. \
 \n*- Created by: Philip B. Alipour, Supervisor: T. A. Gulliver, at the University of Victoria,\
 \n   Dept. ECE, Victoria BC, Canada. \
 \n*- Code updated based on Qiskit 2023--2024 changes specified in code comments of this simulator. \
 \n*- {Fore.LIGHTRED_EX}\033[4m\033[1mSidenotes:\033[0m{Fore.LIGHTGREEN_EX} You can also run code with\
 the right packages installed in pipx or python.\
 \n   Examples of package installation changes and features can be found on: \
 \n   {Fore.LIGHTCYAN_EX}https://docs.quantum.ibm.com/api/migration-guides/qiskit-1.0-installation and \
 \n   https://docs.quantum.ibm.com/api/migration-guides/qiskit-1.0-features{Fore.LIGHTGREEN_EX} \
 \n{hline + Fore.YELLOW}")
 print(f"{Fore.LIGHTGREEN_EX}Press any key to continue...")
 os.system("pause >nul")
 print(f"\033[F{Fore.LIGHTMAGENTA_EX + hline}")
 print(Back.LIGHTGREEN_EX + Fore.YELLOW +
```

```
      "\033[1m<--- {{ Qiskit Aer, IBM }} QDF CIRCUIT SIMULATION BEGINS --->\033[0m" + Back.RESET)

#The following lines plot the P of measurements from the QDF circuit and draws the circuit.
global counts_exp   # Used to plot P results for N shots run on the QDF circuit.
#-------------------------------------------------------------------------------------
# Choose backend, number of shots and the plotting of histogram.
#-------------------------------------------------------------------------------------
#my_provider = IBMProvider() # Previously was IBMQ.load_account().
#print(Fore.LIGHTMAGENTA_EX + f"{hline + Fore.GREEN}\nIBMQ Providers:" + Fore.YELLOW, my_provider.backends())
#ibmq_pick = my_provider.get_backend('ibmq_qasm_simulator') # To run on without dynamic support.
dyn_pick = Aer.get_backend('qasm_simulator') # The device to run on dynamically... 2024 update.
print(Fore.LIGHTMAGENTA_EX + f"{hline + Fore.GREEN}\nDynamic Providers:"+ Fore.YELLOW, dyn_pick)
#ibmq_pick = provider.get_backend('simulator_statevector')
#ibmq_pick = my_provider.get_backend('ibmq_athens') # This machine was recently retired.
#ibmq_pick = my_provider.get_backend('ibmq_16_melbourne') # This machine was recently retired.
shots = 8192 # Number of shots to run the program (experiment); maximum is 8192 shots.
job_exp = dyn_pick.run(qc, shots = shots) # 2024 update... Qiskit package v.1.0 (see heading)
result = job_exp.result()
print(Fore.GREEN + f'Counts for Qubit Pairs out of {shots} shots:' + Fore.RED,
      result.get_counts(qc), Fore.LIGHTGREEN_EX)
counts_exp = (result.get_counts(qc))

#--- The above code snippet is the old version limited to counts ratio to maximum number
# of shots = 8192 resulting to plot probabilities in older Qiskit versions (as
# presented in our method article, Ref. [1]).
# New version to plot probabilities require further computation included for printing
# out the same probabilities based on Maximum of counts/total number of shots =
# a list of P(qubit pairs).
# The observed probabilities are computed by taking the respective counts and dividing
# by the total number of shots as follows.
#-------------------------------------------------------------------------------------
#-------------------------------------------------------------------------------------
# The following lines calculate P values and plot them for the 'counts_exp' above after
# being recorded as expected measurement outcome from the QDF circuit.
#-------------------------------------------------------------------------------------
#N[P(kappa Phi to Psi)]
P = 1 # The total probability of QDF circuit events relative to classical states or
      # denoting the system's Hamiltonian (total energy).

file = "ibm-qdf-shell_output.bin" # I/O bin or metadata file to binary file to read/write.
with open(file, 'w') as file_to_write:
   file_to_write.write(str(counts_exp))
   file_to_write.close() # End write counts_exp metadata to the file.

n_list = []  # Create an empty list for numbers to store.
bin_list = []  # Create an empty list for binaries to store.

with open(file, 'r') as file:

       # Read the contents of the file
       content = file.read()

       # Extract all the digits from the content and convert to int.
       N = re.findall(r'\d+', content)
       n_list = list(map(int, N))

       # Convert all the metadata digits to str to sort out binaries.
       bin_list = list(map(str, N))

       # Calculate quantum event p's for the total P.
       p1 = n_list[1]/shots # 1st p result is stored to the p_list.
       p2 = n_list[3]/shots # 2nd p result is stored to the p_list.
       p3 = n_list[5]/shots # 3rd p result is stored to the p_list.

# This prepares a list of event p's summing up to P in total from the shots.
# Counts for QDF pairwise qubits:
print(f'{Fore.YELLOW} Σ ({n_list[1]}, {n_list[3]}, {n_list[5]}) = {shots} shots ⟶  \
⟨P(b|ij)⟩ = {{ ({{n1, n2, n3}}/{shots})b|ij⟩ }} \
\n = {Fore.LIGHTCYAN_EX}{{ p1({bin_list[0]}) = {p1} }} {Fore.YELLOW}+{Fore.LIGHTCYAN_EX}\
{{ p2({bin_list[2]}) = {p2} }} {Fore.YELLOW}+{Fore.LIGHTCYAN_EX}\
{{ p3({bin_list[4]}) = {p3} }} {Fore.YELLOW}= {P}'), sleep(3)

# Set the default p color codes to 'white' as defined below in the p_color list until an if condition applies.
p_color = [Style.BRIGHT + Fore.WHITE, Style.BRIGHT + Fore.WHITE, Style.BRIGHT + Fore.WHITE]

# Set the high & low p's color-coded conditions of the p_color list against the total P color, set to LIGHTGREEN_EX.
if p1 < 0.5 and p1 >= 0.33:
    p_color[0] = Style.BRIGHT + Fore.WHITE
if p2 < 0.5 and p2 >= 0.33:
    p_color[1] = Style.BRIGHT + Fore.WHITE
if p3 < 0.5 and p3 >= 0.33:
    p_color[2] = Style.BRIGHT + Fore.WHITE

if p1 < 0.33 and p1 >= 0.25:
    p_color[0] = Style.DIM + Fore.YELLOW
if p2 < 0.33 and p2 >= 0.25:
    p_color[1] = Style.DIM + Fore.YELLOW
if p3 < 0.33 and p3 >= 0.25:
    p_color[2] = Style.DIM + Fore.YELLOW

if p1 < 0.25:
    p_color[0] = Style.DIM + Fore.RED
if p2 < 0.25:
    p_color[1] = Style.DIM +  Fore.RED
if p3 < 0.25:
    p_color[2] = Style.DIM + Fore.RED

if p1 > 0.5 and p1 < P:
    p_color[0] = Style.BRIGHT + Fore.LIGHTCYAN_EX
if p2 > 0.5 and p2 < P:
    p_color[1] = Style.BRIGHT + Fore.LIGHTCYAN_EX
if p3 > 0.5 and p3 < P:
    p_color[2] = Style.BRIGHT + Fore.LIGHTCYAN_EX

if ((p1 <= 0.5 and p1 >= 0.4)
    or (p2 <= 0.5 and p2 >= 0.4)
    or (p3 <= 0.5 and p3 >= 0.4)):  # This condition implies to superposition as discussed in Refs. [1, 2],
                                    # as the main discussion around state ⟨2⟩ of qubit pairs from a single
                                    # field vs. a QDF transformation.
    p_color[:] = Style.BRIGHT + Fore.LIGHTMAGENTA_EX # Any color in the p_list is set to that successful event p as magenta.
if p1 == P or p2 == P or p3 == P:  # This only occurs when a p ⟶ ⟨P_success⟩ = P = 1, as
                                   # 100% success probability of the expected measurement outcome.
                                   # See Ref. [2] publication for more details.
    p_color[:] = Style.BRIGHT + Fore.LIGHTGREEN_EX # Any color in the p_list is set to that successful event p as green.
```

```
# Plot QDF pairwise qubits p's relative to P:
print(f"{Fore.LIGHTMAGENTA_EX}{hline}\n Plot = [{Fore.LIGHTGREEN_EX}Experimented n of N counts\
{Fore.LIGHTYELLOW_EX} ∝ {Fore.LIGHTGREEN_EX} p of total circuit events P on pairwise\
qubits {Fore.LIGHTYELLOW_EX}= P(b|ij){Fore.LIGHTMAGENTA_EX}]; {Fore.LIGHTYELLOW_EX}|ij⟩ ≡ |q_i q_j⟩ ≡ |q_i q_j⟩{Fore.LIGHTMAGENTA_EX}.\
\n{hline}")

fig = tpl.figure()
fig.barh([P, p1, p2, p3], [Style.BRIGHT + Fore.LIGHTGREEN_EX+f"P(b|ij))", p_color[0]+f"p1(b|ij)) = p1({bin_list[0]})",
                           p_color[1]+f"p2(b|ij)) = p2({bin_list[2]})", p_color[2]+f"p3(b|ij)) = p3({bin_list[4]})"],
         force_ascii=False)
fig.show()

# Store the P's for simulation use by the QFLCC program for dataset analysis and QDF circuit predictions.
with open('ibm-qdf-stats.txt', 'w') as file_to_write:
   file_to_write.write(f"{P}, {p1}, {p2}, {p3}\nP, p({bin_list[0]}), p({bin_list[2]}), p({bin_list[4]})")
   file_to_write.close() # End write counts_exp metadata to the file.

print(Fore.LIGHTMAGENTA_EX +
      "\n\033[4m<<-- IBM QDF Circuit Measurement Results by Qiskit Aer Simulator Plotted Successfully! End of Task... -->>\033[0m\n"
      + Style.BRIGHT + Fore.LIGHTGREEN_EX + f"\033[1m")

print(f"{Fore.LIGHTGREEN_EX}Press any key to continue...")
os.system("pause >nul")
print(f"\033[F{Fore.LIGHTGREEN_EX + hline}")
```

▼ ▼ i ≙ IBMQ source codes... cont.

IBMQ cont. source code to draw the quantum circuit and count P's from QDF-LCode_IBM-2024.py

```
#-------------------------------------------------------------------------------------------------
# <----- Display histogram results of the QDF circuit event P's and then printing the circuit ----->
#-------------------------------------------------------------------------------------------------
qdf_circuit() # Call this function to plot P results given the number of shots.

#-------------------------------------------------------------------------------------------------
# Visualize the Circuit in the terminal by printing the circuit... This is in python. IBMQ kernel
# does not require print(), or simply enter qc.draw() in this case. See code from the
# 'QDF-LCode_IBMQ-2024-raw-codable.ipynb' codable file or its 'QDF-LCode_IBMQ-2024.ipynb' base file.
#-------------------------------------------------------------------------------------------------
print(qc.draw())

fcircuit = "ibm-qdf-circuit_output.bin" # The file that the printed circuit is saved.
with open(fcircuit, 'w', encoding='utf-8') as file_to_write:
    file_to_write.write(str(qc.draw()))
    file_to_write.close() # End writing the printed circuit.

print(Back.RESET + Fore.MAGENTA+"\n\033[4m<<-- QDF Circuit BUILT & RAN Successfully! End of Task... -->>\033[0m\n"
       + Fore.LIGHTGREEN_EX), sleep(3)
print(Back.LIGHTGREEN_EX + Fore.YELLOW
      + f"{hline}\n\033[1m<--- {{ Qiskit Aer, IBM }} QDF CIRCUIT SIMULATION CONCLUDED --->\n{hline}\033[0m" + Back.RESET)
#########--END OF PROGRAM--#########
```

▼ ▼ i ≙ QInspire source codes...

QInspire source code with QDF default circuit config. as QDF-LCode_QI.cq

```
Version 1.0
#-------------------------------------------------------------------------
# A quantum field lens coding algorithm by a conventional quantum circuit
# written in cQSAM by P. B. Alipour, run on Quantum Computers in QI, 2020-2023
#-------------------------------------------------------------------------
qubits 6 # Initialize a 6-qubit register (define the number of qubits)
.init
prep_z q[0:5] # State initialization on 6 qubits in the z-basis in the |0>
# state; Initialize inputs to some values
#-------------------------------------------------------------------------
# Qubit definitions for constructing an adder:
# q[0] --> |A>
# q[1] --> |B>
# q[2] --> CarryIn_SumOut
# q[3] --> CarryOut
#-------------------------------------------------------------------------
# Initialize inputs A=1, B=0 and CarryIn =1 to perform addition after
# decoding Alice's message about the prize position
{X q[0] | X q[2]}

#-------------------------------------------------------------------------
# Define the combined .superposition and .entanglement sub-circuit as
# .superpose_entangle for superdense coding
#-------------------------------------------------------------------------
.superpose_entangle(2) # Iterate up to i=2 times (third party encodes qubits);
# any i>2 times for any other scenario

H q[1] # Create a superposition of two states

cnot q[1], q[2] # Entangle two qubits via CNOT gate

CR q[1], q[4], 0.39 # Controlled phase shift (or T-gate equivalent)
# gate between qubits 1 and 4 with an angle of 0.39 radians = pi/8 degrees
H q[4]

.sender_encode_bits(1) # From superdense coding scheme , Alice sends two
# classical bits as one qubit to receiver
Z q[1]
{Rz q[4], -0.32 | Rz q[5], -1.23}
# (Rz q[5] is the controlled rotation on ancilla qubit 0, mapping to ((pi^2)/8
# + (pi^2)/30) rad or =< pi/2 degrees; The system is evolved to its excited states
# when the measurement on the ancilla qubit is in state |0>. Meanwhile , the system
# is purified to a state that is close to its ground state (steps of decompression
# discussed in the EE measurement section) as measurement takes place to observe
# its excited state |1>.

CR q[4], q[5], 0.39
X q[1]
# .middle_recorder(1) or Eve gathers information form Alice with increased certainty
# and sends info to Bob (see the QDF game model steps)

.receiver_decode_bits(1) # Bob as receiver
# decodes bits within two consecutive steps of the game; below Eve collects information
# from Alice relative to Bob as .middle_recorder(1) which defines a single step of
# kappa -based QDF transformation = a momentum (kinetic energy) operator.
H q[4]
CR q[1], q[4], 0.39
cnot q[1], q[2]
H q[1]
SWAP q[1], q[4] # SWAP gate on qubits 0 and 1 after the quantum field (QDF) transformation

measure_z q[1:2]

# Perform addition
.full_adder(1)
toffoli q[0],q[1],q[3]
cnot q[0],q[1]
toffoli q[1],q[2],q[3]
cnot q[1],q[2]
cnot q[0],q[1]

measure_z q[2, 4, 5]
# Measure the sum output from qubit 2 and the controlled rotation as momentum (kinetic
# energy) operator.
#-------------------------------------------------------------------------
# Listing 1. Quantum circuit code for the corresponding figure exported from QI
# platform written in QI at: https://www.quantum-inspire.com/, ran on a 26-qubit QX
# single-node simulator. By removing the relevant circuit components from this code,
# the superdense coding circuit for Fig. 4(a) can be reproduced to run on a 5-qubit
# processor, which is implemented between lines No. 5-7, 28-30, 37-49, 58-60.
#-------------------------------------------------------------------------
```

QAI-LCode_QFLCC Source Code

Downloadable source code is:

QAI-LCode_QFLCC file download

▼ Expand/Collapse All...

```
* Click Expand All to view all code blocks from QAI-LCode_QFLCC.py on this page.
* Click Collapse All to selectively view a code block from QAI-LCode_QFLCC.py on this page.
```

Expand All | Collapse All

QAI-LCode_QFLCC Module

▼ ⚠ Module description and code usage warning:

The current main module is the source file `QAI-LCode_QFLCC.py`. This module will be split into the `QDF-game.py` and `QAI-LCode_QFLCC.py` for import. The `QDF-game.py` is already within the file `QAI-LCode_QFLCC.py` file. The updated version to import the smaller `QAI-LCode_QFLCC.py` for the `QDF game` is under development as the next version of the code. QAI-LCode_QFLCC file download

- `ExitMyProgram`

  Bases: `Exception`
  - Exception used to exit program.

  ▼ Source code in `QAI-LCode_QFLCC.py`

```
104 class ExitMyProgram(Exception):
105     """- Exception used to exit program."""
```

`bg`

▼ Source code in `QAI-LCode_QFLCC.py`

```
class bg:
        black='\033[40m'
        red='\033[41m'
        green='\033[42m'
        orange='\033[43m'
        blue='\033[44m'
        purple='\033[45m'
        cyan='\033[46m'
        lightgrey='\033[47m'
        """- Colored background (bg) class for sectioning and highlighting the
        QF-LCC algorithm step, checkpoint, computer operation, error, or HALT."""
```

`lightgrey = '\x1b[47m'` `class-attribute` `instance-attribute`

- Colored background (bg) class for sectioning and highlighting the QF-LCC algorithm step, checkpoint, computer operation, error, or HALT.

`fg`

▼ Source code in `QAI-LCode_QFLCC.py`

```
class fg:
        black='\033[30m'
        silver="\033[0;38;5;7m"
        red='\033[31m'
        green='\033[32m'
        orange='\033[40m'
        blue='\033[34m'
        purple='\033[35m'
        cyan='\033[36m'
        lightgrey='\033[37m'
        darkgrey='\033[90m'
        lightred='\033[91m'
        lightgreen='\033[92m'
        yellow='\033[93m'
        lightblue='\033[94m'
        pink='\033[95m'
        lightcyan='\033[96m'

        """- Colored foreground (fg) class for sectioning and highlighting the
        QF-LCC algorithm step, checkpoint, computer operation, error, or HALT."""
```

`lightcyan = '\x1b[96m'` `class-attribute` `instance-attribute`

- Colored foreground (fg) class for sectioning and highlighting the QF-LCC algorithm step, checkpoint, computer operation, error, or HALT.

`build(debug)`

- Build production assets.

▼ Source code in `QAI-LCode_QFLCC.py`

```
@cli.command()
@click.option("-d", "--debug", help="Include debug output.")
def build(debug):
    """- Build production assets."""
```

`cli()`

Main entry point.

▼ Source code in `QAI-LCode_QFLCC.py`

```
@click.group()
def cli():
    """Main entry point."""
```

- ▼ Source code in `QAI-LCode_QFLCC.py`

```
    def PAnalysis_model():
    ##########################################################################################################################
    #--- This function compares the mapped results between QDF circuits (IBMQ <--> QI).
    #--- SAFE MODE QFLCC AND,
    #--- QDF Circuit Model from IBMQ article file: QDF-LCode_IBMQ-2024-codable.py,
    # for python library purposes imported as a module:
    #--- QDF-LCode_IBMQ-2024.py (same code content as a duplicate of QDF-LCode_IBMQ-2024-codable, but will be overwritten
    # by IBMQ folder content if you modify or update code. Only edit code within the QDF-LCode_IBMQ-2024-codable.py file!).
    ##########################################################################################################################
    global sim_state, dir_flag, entry_stage, hline
    sim_state = ['']
    hline = "=============================================================================================================================================="
    print(f'''{Fore.LIGHTMAGENTA_EX + hline}\n An ideal QDF circuit I/O model running realtime producing an IBMQ-based QDF dataset is from the \
    \n QDF-LCode_IBMQ-2024.py file. This circuit is compared to the analyzed dataset to show how close \
    \n the match is for a desired Hamiltonian and expected measurement outcome as a point of reference.\
    \n{hline}''')
    #-------------------------------------------------------------------------------------------
    # <----- Display histogram results of the QDF circuit event P's and then printing the circuit
    # as a model to display for QDF datasets of other circuits being analyzed here (next version will).
    # Future release will have current P results of QInspire or any other QDF circuit configure the
    # IBM-QDF circuit through a QAI mapping, accordingly (subject of QAI lab for QFLCA) ----->
    #-------------------------------------------------------------------------------------------
    P = 1 # The total probability of QDF circuit events relative to classical states or
          # denoting the system's Hamiltonian (total energy).
    with alive_bar(3, bar = 'blocks', manual=True) as bar:   #To print progress bar on these dataset and circuit simulation analyses.
       print(Fore.LIGHTMAGENTA_EX + Back.LIGHTGREEN_EX+ f"Please Wait!...:")
       print(f"*- Program is processing the analyzed dataset circuit P's after:"+Back.RESET), sleep(1)
       bar(0.5)
       print(f"{Back.LIGHTGREEN_EX}*- Import + Simulate the IBM QDF circuit by\
    {{ Qiskit Aer Simulator, IBMQ Provider }} [QUANTUM MODE ENVIRONMENT] on this computer."+Back.RESET), sleep(1)
       bar(0.7)
       print(f"{Back.LIGHTGREEN_EX}*- Dataset P results are compared on the imported circuit by the {{ QDF-LCode_IBMQ-2024.py }} module\
    on this computer [SAFE MODE ENVIRONMENT]."+ Back.RESET + Fore.LIGHTMAGENTA_EX), sleep(1)
       bar(1.)

    # Log the start of simulation.
    entry_stage = 0
    sim_state[0] = " QDF CIRCUIT SIMULATION BEGINS "
    sim_log()
    #-------------------------------------------------------------------------------------------------------------------------
    ibm_qdf_module = importlib.import_module("QDF-LCode_IBMQ-2024-codable") # Unconventional call from the targeted module.
    bar(1.)
    #-------------------------------------------------------------------------------------------------------------------------
    # Log the prompt of simulation to the user to proceed, restart or exit program.
    entry_stage = 1
    sim_state[0] = " QDF CIRCUIT SIMULATION_DATASET_ANALYSIS PROMPT "
    sim_log()
    #-----------------------------------------------------------------------------
    # Selected dataset and the simulated circuit's dataset validation verdict starts here,
    # if user enters 'n' or 'next'...
    # ----------------------------------------------------------------------------
    print(Fore.LIGHTGREEN_EX + f"*- Enter 'n' or 'next' to proceed calculating P's between the selected dataset and the simulated\
    IBM QDF circuit results.")
    print(f"© {Fore.LIGHTMAGENTA_EX}\x1B[1;4mThe datasets of the two QDF circuits are compared and validated with a verdict on how\
    close their projection of \n events (prediction) is, given their QDF circuit configuration!\033[0m")
    print(f"{Fore.LIGHTGREEN_EX}*- ✅ Enter 'dir' or 'sm dir' to select another dataset to analyze in Safe Mode (SM:>>).\
    SIMULATION WILL NOT RESTART!")
    print(f"{Fore.LIGHTGREEN_EX}*- ✅ Enter 'n' or 'next' to continue in Safe Mode (SM:>>).\
    QDF CIRCUITS SIMULATION_DATASET_ANALYSIS WILL START!")
    print(f"{Fore.LIGHTGREEN_EX}*- Enter any other key to change from SM:>> to regular prompt mode (>).")
    print(f"*- Enter 'r' in regular prompt mode (>) to 'restart' circuit simulation and dataset analysis. Enter 'h' for more options.")

    sm_input = input(f"SM:>> {fg.yellow}")
    if sm_input == "dir" or sm_input == "sm dir":
         dir_flag = 1 # Directory flag is set to 1 for directory access in safe mode.
         safeMode_dir()
         pass
    elif sm_input == "n" or sm_input == "next":
         dir_flag = 1
         print(f"{Fore.LIGHTGREEN_EX}SM:>>{fg.yellow} Next..."), sleep(1)
    else:
         dir_flag = 0  # Directory flag is set to 0 in case of regular prompt mode (>) or 'r' to restart the simulation.
         prompt()

    print(Back.LIGHTGREEN_EX + Fore.YELLOW + "\033[1m<--- QDF CIRCUITS SIMULATION_DATASET_ANALYSIS BEGINS --->\033[0m" + Back.RESET)

    ibmq_result = 'ibm-qdf-stats.txt'
    with open(ibmq_result, 'r') as file:
           # Read the contents of the file
           content = file.read()

           # Extract all the p values from the content as float.
           N = re.findall(r'[-+]?(?:\d*\.*\d+)', content)
           n_list = list(map(float, N))

           # Convert all the metadata digits to str to sort out binaries.
           bin_list = list(map(str, N))  # This is to identify qubit binary strings.

           # Calculate quantum event p's for the total P.
           P = n_list[0]  # The P result is stored to the p_list.
           p1 = n_list[1] # 1st p result is stored to the p_list.
           pq1 = bin_list[4]
           p2 = n_list[2] # 2nd p result is stored to the p_list.
           pq2 = bin_list[5]
           p3 = n_list[3] # 3rd p result is stored to the p_list.
           pq3 = bin_list[6]

    ibmq_p_max = max(p1, p2, p3)  # Identify the max value from the p list.
    max_p_index = n_list.index(ibmq_p_max) # Store the index value for p_max from the ibmq_result file.

    ibmq_p_min = min(p1, p2, p3)  # Identify the min value from the p list.
    min_p_index = n_list.index(ibmq_p_min) # Store the index value for p_min from the ibmq_result file.

    # Set the default p color codes to 'white' as defined below in the p_color list until an if condition applies.
    p_color = [Style.BRIGHT + Fore.WHITE, Style.BRIGHT + Fore.WHITE, Style.BRIGHT + Fore.WHITE,
               Style.BRIGHT + Fore.WHITE, Style.BRIGHT + Fore.WHITE]

    if round(p1, 2) <= round(ibmq_p_min, 2): # Classify/color code min(p1).
         p_color[0] = Style.BRIGHT + Fore.RED
    if round(p2, 2) <= round(ibmq_p_min, 2): # Classify/color code min(p2).
         p_color[1] = Style.BRIGHT + Fore.RED
    if round(p3, 2) <= round(ibmq_p_min, 2): # Classify/color code min(p3).
         p_color[2] = Style.BRIGHT + Fore.RED
    if round(p1, 2) >= round(ibmq_p_max, 2): # Classify/color code max(p1).
```

```
            p_color[0] = Style.BRIGHT + Fore.CYAN
        if round(p2, 2) >= round(ibmq_p_max, 2): # Classify/color code max(p2).
            p_color[1] = Style.BRIGHT + Fore.CYAN
        if round(p3, 2) >= round(ibmq_p_max, 2): # Classify/color code max(p3).
            p_color[2] = Style.BRIGHT + Fore.CYAN

        print(f"{Fore.LIGHTMAGENTA_EX}{hline}\nPlot = [{Fore.LIGHTGREEN_EX}P samples of the IBMQ model circuit,\
        if meets {Fore.LIGHTYELLOW_EX} ∝ {Fore.LIGHTGREEN_EX} p of the selected dataset file\
        {{ {Fore.LIGHTYELLOW_EX + fileresult[filenum-1] + Fore.LIGHTGREEN_EX} }}\non pairwise \
        qubits{Fore.LIGHTMAGENTA_EX}]...\nThen a strong vs weak p match, and the distance between QDF circuit events\
        are determined.{Fore.YELLOW}* {Fore.LIGHTGREEN_EX}\n{hline}{Fore.LIGHTGREEN_EX}\
        \n{Fore.YELLOW}* Computation model:{Fore.GREEN} κ Scalar Ψ0 Field Switch and Correlation Model, Ref. [1] of the DIB article.\
        \n  {Fore.LIGHTCYAN_EX}P(Ψ ↔  κ²Ψ) = ⟨P(Ψ ↔ Φ)⟩ = P(b|ij)); |ij⟩ ≡ |q_i q_j⟩ ≡ |qᵢqⱼ⟩. {Fore.GREEN}\
        \n{hline}"), sleep(0.5)

        print(f'''{Fore.LIGHTMAGENTA_EX}\n\033[4m<-- P Sampling between IBM QDF Circuit and the Selected QDF Circuit Dataset -->\033[0m''')
        if dir_flag == 1:  # Read and display the printed circuit when dataset dir is in SAFE MODE.
              with open("ibm-qdf-circuit_output.bin", 'r', encoding='utf-8') as file_to_read:
                    print(f'{fg.black}{bg.lightgrey}\n'+file_to_read.read(),"\n<-- IBM QDF Circuit Sample Printed in SAFE MODE --> "
                          + Back.BLUE + str(datetime.datetime.now()) + Back.RESET)
                    file_to_read.close() # End reading and displaying the printed circuit.
        print(f'''{Fore.LIGHTMAGENTA_EX}\n\033[4m<-- P Results Sampled from {{ {ibmq_result} }} file are: -->\033[0m''')
        with alive_bar(3, bar = 'blocks' , manual=True) as bar:
         model_fig = tpl.figure() # Now plot histogram with horizontal bars for the computed probabilities.
         model_fig.barh([P, p1, p2, p3], [Style.BRIGHT + Fore.LIGHTGREEN_EX+"P(Total)", p_color[0]+f"ibmq qdf p({bin_list[4]})",
                                          p_color[1]+f"ibmq qdf p({bin_list[5]})", p_color[2]+f"ibmq qdf p({bin_list[6]})"],
                force_ascii=False), sleep(1)
         model_fig.show(), sleep(1)
         print('')
         bar(1)
         dir_flag = 0 # Reset flag until recalled.
         qdf_bit_pairs = ['',''] # To register which qdf_bit_pair has the max or min p.
         #-------------------------------------------------------------------------------------
         # Now classify and print which sampled qdf_bit_pair set from the
         # ibmq_result file is max_p and which min_p.
         if min_p_index == 1:
          qdf_bit_pairs[0] = pq1
         if min_p_index == 2:
          qdf_bit_pairs[0] = pq2
         if min_p_index == 3:
          qdf_bit_pairs[0] = pq3
         if max_p_index == 1:
          qdf_bit_pairs[1] = pq1
         if max_p_index == 2:
          qdf_bit_pairs[1] = pq2
         if max_p_index == 3:
          qdf_bit_pairs[1] = pq3

        print(Fore.YELLOW+f'{hline}')
        print(fg.purple+f'*- min(P) from {{ {ibmq_result} }} is performed by the QDF bit pairs\
        {{ {fg.red+ qdf_bit_pairs[0] +fg.purple} }} as {{ ibm qdf }} sample set {fg.yellow}#{min_p_index} = {ibmq_p_min}')
        print(fg.purple+f'*- max(P) from {{ {ibmq_result} }} is performed by QDF bit pairs\
        {{ {fg.lightgreen+ qdf_bit_pairs[1] +fg.purple} }} as {{ ibm qdf }} sample set {fg.yellow}#{max_p_index} = {ibmq_p_max}')
        print(Fore.YELLOW+f'{hline}')
        #-------------------------------------------------------------------------------------------------------
        # Recall and store which binary string had the minimum P from the table of the selected dataset (dataframe).
        global df_min, df_min_bin
        if  (sample_data.min(axis=0)[1] < 1):
                 df_min=df.loc[(df['probability'] <= sample_data.min(axis=0)[1]), :].binary_string[:]
                 df_min_bin = df_min.to_string(index=False, header=False)
        #-------------------------------------------------------------------------------------------------------
        deltaMatch_max = (1 - abs(ibmq_p_max - float(df3Mem)) ) # Calculate Δp of max(P) Match.
        deltaMatch_min = (1 - abs(ibmq_p_min - sample_data.min(axis=0)[1]) ) # Calculate Δp of min(P) Match.
        delta_p_min = abs(ibmq_p_min - sample_data.min(axis=0)[1]) # Calculate Δp of min(P).
        delta_p_max = abs(ibmq_p_max - float(df3Mem)) # Calculate Δp of max(P).

        Valid_Verdict = ['Weak',f'{{∞ , ⁿ/ₐ}}', 'Avg.', 'Strong', 'ₙₐ']  # Validation Verdict of a strong
                                                                        # vs. weak correlation match between
                                                                        # the selected file dataset p's and the IBM QDF
                                                                        # circuit event p's...
        # Note: A verdict of ₙₐ denotes missing information or unknown about the max or min of the p's
        # (calculable if config is corrected or complemented).

        if round(deltaMatch_max, 2) >= 0 and round(deltaMatch_max, 2) < 0.5: # Classify/color code max Δp match.
            p_color[3] = Style.BRIGHT + Fore.RED
            Valid_Verdict = Fore.LIGHTRED_EX + Valid_Verdict[0]

        if round(deltaMatch_max, 2) >= 0.5 and round(deltaMatch_max, 2) <= 0.55: # Classify/color code max Δp match.
            p_color[3] = Style.NORMAL + fg.silver
            Valid_Verdict = Fore.LIGHTWHITE_EX + Valid_Verdict[1]

        if round(deltaMatch_min, 2) >= 0 and round(deltaMatch_min, 2) < 1/3: # Classify/color code min Δp match.
            p_color[4] = Style.BRIGHT + Fore.RED
            Valid_Verdict = Fore.LIGHTRED_EX + Valid_Verdict[0]

        if (round(deltaMatch_min, 2) <= 1/3 and
            round(deltaMatch_max, 2) <= 1/3) or (round(deltaMatch_min, 2) <= 1/2 and
                                                 round(deltaMatch_max, 2) <= 1/2): # Classify/color code min-max Δp match.
            p_color[3] = Style.NORMAL + fg.silver
            p_color[4] = p_color[3]
            Valid_Verdict = Fore.LIGHTWHITE_EX + Valid_Verdict[1]

        if round(deltaMatch_max, 2) > 0.55 and round(deltaMatch_max, 2) < 0.66: # Classify/color code max Δp match.
            p_color[3] = Style.BRIGHT + Fore.LIGHTYELLOW_EX
            Valid_Verdict = Fore.LIGHTYELLOW_EX + Valid_Verdict[2]

        if (round(deltaMatch_max, 2) >= 0.66 and
            round(deltaMatch_max, 2) < 0.9) and (round(deltaMatch_min, 2) >= 0.66 and
                                                 round(deltaMatch_max, 2) < 0.9): # Classify/color code min-max Δp match.
            p_color[3] = Style.BRIGHT + Fore.LIGHTMAGENTA_EX
            p_color[4] = p_color[3]
            Valid_Verdict = Fore.LIGHTCYAN_EX + 'Above ' + Valid_Verdict[2]

        if (round(deltaMatch_max, 2) >= 0.9 and
            round(deltaMatch_max, 2) <= 1) and (round(deltaMatch_min, 2) >= 0.9 and
                                                round(deltaMatch_max, 2) <= 1): # Classify/color code min-max Δp match.
            p_color[3] = Style.BRIGHT + Fore.LIGHTGREEN_EX
            p_color[4] = p_color[3]
            Valid_Verdict = Fore.LIGHTGREEN_EX + Valid_Verdict[3]

        if (round(deltaMatch_min, 2) <= 1/3 and
            round(deltaMatch_max, 2) <= 1/2) or (round(deltaMatch_min, 2) <= 1/2 and
                                                 round(deltaMatch_max, 2) <= 1/2): # Classify/color code min-max Δp match.
            p_color[3] = Style.NORMAL + fg.silver
            p_color[4] = p_color[3]
```

```
            Valid_Verdict = fg.silver + Valid_Verdict[4] # See note for a ₙᵤₗₗ verdict!

    PAnalysis_fig = tpl.figure() # Now plot histogram with horizontal bars for the computed probabilities of the two datasets.
    PAnalysis_fig.barh([P, float(df3Mem), float(dfcompMem), sample_data.min(axis=0)[1], ibmq_p_min, ibmq_p_max,
                        delta_p_min, delta_p_max, deltaMatch_min, deltaMatch_max],
        [Style.BRIGHT + Fore.LIGHTGREEN_EX + f"P(\x1B[4mTotal\x1B[0m{Style.BRIGHT + Fore.LIGHTGREEN_EX})",
         f'{Style.BRIGHT+Fore.WHITE}{{ {fileresult[filenum-1]} }} max(P(\x1B[4m{df2bin}\x1B[0m{Style.BRIGHT+Fore.WHITE}))',
         f'{Style.RESET_ALL + Fore.WHITE}{{ {fileresult[filenum-1]} }} comp(P(\x1B[4m{bitcomp}\x1B[0m{Style.DIM+Fore.WHITE}))',
         Style.BRIGHT + Fore.WHITE + f"{{ {fileresult[filenum-1]} }} min(P(\x1B[4m{df_min_bin}\x1B[0m{Style.BRIGHT + Fore.WHITE}))",
         f"{Style.BRIGHT +  Fore.YELLOW}{{ ibm qdf }} min((P(\x1B[4m{qdf_bit_pairs[0]}\x1B[0m{Style.BRIGHT + Fore.YELLOW})) ",
         f"{Style.BRIGHT +  Fore.YELLOW}{{ ibm qdf }} max((P(\x1B[4m{qdf_bit_pairs[1]}\x1B[0m{Style.BRIGHT + Fore.YELLOW})) ",
         p_color[4] + "Δp of \x1B[4mmin(P)\x1B[0m" + p_color[4],
         p_color[3] + f"Δp of \x1B[4mmax(P)\x1B[0m" + p_color[3],
         p_color[4]+ "Δp of\x1B[4m min(P) match\x1B[0m" + p_color[4],
         p_color[3]+ "Δp of\x1B[4m max(P) match\x1B[0m" + p_color[3]], force_ascii=False), sleep(1)
    PAnalysis_fig.show()
    bar(1.)

    print(f"{Fore.LIGHTYELLOW_EX+hline+ Fore.LIGHTMAGENTA_EX}\n*- Δp Data: ☣  \033[4mValidation Verdict\033[0m ☣{fg.yellow}  \
    between the two dataset samples from {fg.cyan}{{ {fileresult[filenum-1]} , {ibmq_result} }}{fg.yellow} \
    \n   is a/an: \
    \033[4m{Valid_Verdict + ' Match {{ '+ str(round(deltaMatch_min, 2)) +' , '+ str(round(deltaMatch_max, 2)) +' }}'}\033[0m\
    {Fore.LIGHTYELLOW_EX}")

    DeltaP_verdict = Valid_Verdict
    Valid_Verdict = ['Weak',f'{{∞ , ⁿ/ₐ}}', 'Avg.', 'Strong', 'ₙᵤₗₗ']  # Reset Valid_Verdict list elements upon
                                                                      # the previously rendered verdict.
    #-------------------------------------------------------------------------------------
    # Validate circuit config. based on recorded P's associated to their qubit pair binaries (strings)
    s1 = df_min_bin
    s2 = qdf_bit_pairs[0]
    s3 = df2bin
    s4 = qdf_bit_pairs[1]
    vv_circuits =[False]
    vv_bins = [False]

    # Validation Verdict on the two circuits qubit string sets over min and max P's as compared for a match.
    if re.match(s2, s1):
        vv_bins = fg.lightgreen+ Valid_Verdict[3]
        qdf_s_match = f"{{ {s1} , {s2} }}"
    elif re.match(s4, s3):
        vv_bins = fg.lightgreen + Valid_Verdict[3]
        qdf_s_match = f"{{ {s3} , {s4} }}"
    else:
        vv_bins = fg.red + Valid_Verdict[4] # See note for a ₙᵤₗₗ verdict!

    # Validation Verdict on the two circuits as compared for a config. match.
    if (vv_bins == fg.lightgreen + Valid_Verdict[3]) and (DeltaP_verdict == Fore.LIGHTGREEN_EX + Valid_Verdict[3]):
        vv_circuits = fg.green + Valid_Verdict[3]
    if (vv_bins == fg.lightgreen + Valid_Verdict[3]) and (DeltaP_verdict == Fore.LIGHTGREEN_EX + Valid_Verdict[1]):
        vv_circuits = fg.yellow + Valid_Verdict[2]
    if (vv_bins == fg.lightgreen + Valid_Verdict[3]) and (DeltaP_verdict == Fore.LIGHTCYAN_EX + 'Above ' + Valid_Verdict[2]):
        vv_circuits = Fore.LIGHTCYAN_EX + 'Above ' + Valid_Verdict[2]
    if (vv_bins == fg.red + Valid_Verdict[4]) and (DeltaP_verdict == Fore.LIGHTGREEN_EX + Valid_Verdict[3]):
        vv_circuits = fg.yellow + Valid_Verdict[2]
    if ((vv_bins == fg.red +
         Valid_Verdict[4]) or (vv_bins == fg.lightgreen + Valid_Verdict[3])) and (DeltaP_verdict == fg.silver + Valid_Verdict[4]):
         vv_circuits = f'{{ {fg.lightred + Valid_Verdict[4]} , {False} }}' # A ₙᵤₗₗ or false match verdict! See also next line/condition.
    if (float(df3Mem) <= 1/3 and sample_data.min(axis=0)[1]<= 1/3) and (ibmq_p_max >= 1/2):
        vv_circuits = f'{fg.lightred}{{ {Valid_Verdict[4]} , {fg.yellow}{Valid_Verdict[1]}{fg.lightred} }}' # A ₙᵤₗₗ match verdict! In short,
         # a state transition matrix with all elements between certain circuit components return a 0 state or null match and
         # state 2 for one of the two circuits! See P's preliminary analysis for the measured QDF circuit.
         # This circuit event outcome can also be interpreted as a rejected match between the two circuits, based on their configuration.
    if (vv_bins == fg.red + Valid_Verdict[4]) and (Valid_Verdict == Fore.LIGHTYELLOW_EX + Valid_Verdict[2]):
        vv_circuits = fg.red + Valid_Verdict[2]
    #-------------------------------------------------------------------------------------
    print(f"\033[1m{Fore.LIGHTYELLOW_EX+hline+ Fore.LIGHTMAGENTA_EX}\n*- QDF Qubit Sets: ☣  \033[4mValidation Verdict\033[0m ☣{fg.yellow}  \
    between the two dataset samples from {fg.cyan}{{ {fileresult[filenum-1]} , {ibmq_result} }}{fg.yellow} \
    \n   is a/an: \033[4m{vv_bins + ' Match ' + qdf_s_match}\033[0m {Fore.LIGHTYELLOW_EX}")

    print(f"\033[1m{Fore.LIGHTYELLOW_EX+hline+ Fore.LIGHTMAGENTA_EX}\n*- QDF Circuits: ☣  \033[4mValidation Verdict\033[0m ☣{fg.yellow}  \
    between the two QDF circuit configurations from {fg.cyan}{{ {fileresult[filenum-1]} , {ibmq_result} }}{fg.yellow} \
    \n   is a/an: \033[4m{vv_circuits} Match \033[0m {Fore.LIGHTYELLOW_EX}")

    # Log validation verdict results on simulation and dataset analysis.
    entry_stage = 2
    sim_state[0] = f"Δp Data: {Valid_Verdict}, QDF Qubit Sets: {vv_bins + ' Match ' + qdf_s_match},  QDF Circuits: {vv_circuits} Match"
    sim_log()

    print(f"{hline + Fore.LIGHTMAGENTA_EX}\n *- The IBM QDF circuit can be reconfigured for worst and best case Hamiltonian scenarios,\
    \n    given the expected QDF measurement outcomes from the collected QFLCA datasets for a QFLCC.\
    \n\x1b[38;5;226m Simulation completed on: \033[1;32m{fg.yellow+bg.blue+str(datetime.datetime.now())+Back.RESET} \
    \n{hline + Fore.RESET}"), sleep(4)

    dir_flag = 0
    print(Back.LIGHTGREEN_EX + Fore.YELLOW + "\033[1m<--- QDF CIRCUITS SIMULATION_DATASET_ANALYSIS CONCLUDED --->\033[0m"
          + Fore.LIGHTGREEN_EX + Back.RESET)

    # Log the end of simulation and dataset analysis.
    entry_stage = 3
    sim_state[0] = " QDF CIRCUITS SIMULATION_DATASET_ANALYSIS CONCLUDED "
    sim_log()
```

▼ ▼ i Expand/Collapse All...

* Click ▼Expand All to view all code blocks from QAI-LCode_QFLCC.py on this page.
* Click ▶Collapse All to selectively view a code block from QAI-LCode_QFLCC.py on this page.

▼Expand All ▶Collapse All

QAI-LCode_QFLCC Functions and Calls

▼ ▼ i ⚠ QFLCC intro, animation, help, website, logging, sound and exit functions...

i To prompt and help user to select and enter an option to change e.g., sound volume, select file(s) to install and exit program when asked by the user. Function() list from top to bottom as ordered:

▼ ▼ site_doc ( ) function website code:

i To show the QFLCA website about its project files and program codes.

- ▼ ▼ Source code in `QAI-LCode_QFLCC.py`

```
147     def site_doc():
148     ###############################################################
149     # To load and read QFLCA project documentation website.
150     ###############################################################
151         global site_dir, site_name
152         site_file = ["index.html", "about.html", "QAI-LCode_QFLCC-reference.html",
153                      "QDF-game-reference.html"] # This html file is stored in the same site folder to read.
154         site_dir = "site" # This site folder contains project's documentation html files to read.
155         winsound.PlaySound("SystemQuestion", winsound.SND_ALIAS) # Play sound.
156         if site_name == "home-site":
157             webbrowser.open_new_tab(os.path.join(site_dir, site_file[0]))
158         elif  site_name == "about-site":
159               webbrowser.open_new_tab(os.path.join(site_dir, site_file[1]))
160         elif  site_name == "qflcc-site":
161               webbrowser.open_new_tab(os.path.join(site_dir, site_file[2]))
162         elif  site_name == "game-site":
163               webbrowser.open_new_tab(os.path.join(site_dir, site_file[3]))
```

▼ clear_line ( ) function animation code:

To jump back within the CLI and reenter images stored from an array in a sequence.

- ▼ Source code in `QAI-LCode_QFLCC.py`

```
259     def clear_line(wid):
260         global spaces
261         LINE_UP = '\033[1A'
262         LINE_CLEAR = '\x1b[2K'
263         for spaces in range(wid):
264           print(LINE_UP, end=LINE_CLEAR)
```

▼ intro_anim ( ) function animation code:

To load and play 1D QDF circuit images in the terminal from an array in a sequence.

- ▼ Source code in `QAI-LCode_QFLCC.py`

```
    def intro_anim():
         global chars
    ######################################
    # Calling this function will run
    # an animated intro image displayed
    # on terminal!
    ######################################
         # Enable the following commented code lines if you switch to chafa.loader..
         # The active code lines use both chafa.loader and
         # the Pillow (PIL) package definitions/functions. Only comment out when indicated..
         #-------------------------------------------------------------------------------
         #FONT_HEIGHT = 12*6  #24 # Default value for image height >= image width.
         #FONT_WIDTH  = 26*2  #11 # Default value for image height >= image width.
         #FONT_RATIO = (FONT_WIDTH / FONT_HEIGHT) # A scalar ratio to keep dimensions fixed by product multiplication.

         config = CanvasConfig()

         #config.height = 9 #60 # Default value for a square image.
         #config.width  = 160 #60 # Default value for a square image.

         # Set the canvas cell geometry.
         #config.cell_height = FONT_HEIGHT
         #config.cell_width  = FONT_WIDTH

         # Create a list of image objects
         img_list=  ["./__img__/qdf-circuit-sprite/1D-circuit-00.png", #32-bit depth images
                     "./__img__/qdf-circuit-sprite/1D-circuit-a.png", "./__img__/qdf-circuit-sprite/1D-circuit-b.png",
                     "./__img__/qdf-circuit-sprite/1D-circuit-c.png", "./__img__/qdf-circuit-sprite/1D-circuit-d.png",
                     "./__img__/qdf-circuit-sprite/1D-circuit-d.png"]

         # Loop through the gif image objects for a while.
         for j in range(0, 6):
          for img in img_list:
              print("\033[1A", end="\x1b[2K", flush= True), sleep(0.001)
              #image = Loader(img_list[j])   # Open image with chafa.loader or Loader.
              image = Image.open(img_list[j]) # Open image with PIL. Not used by Loader.

              wid, hgt = image.size
              mode = image.mode  # Detect image mode on your system.

              # When PIL package definitions are used. Comment out if using chafa.loader.
              if (img == "./__img__/1D-QDFcircuit-sprite/1D-circuit-02.png" or
                  img == "./__img__/1D-QDFcircuit-sprite/1D-circuit-03.png"):
                   config.height = (hgt/8.8)
                   config.width  = wid/8.8
              else:
                config.height = hgt/7.8
                config.width  = wid/6.6

              #config.height = (hgt/6.4) - 1
              #config.width  = wid/7.2
              # Enable one of the switches to configure canvas to draw a high resolution image (for chafa.loader).
              #-------------------------------------------------------------------------------------------------------
              #config.pixel_mode = chafa.PixelMode.CHAFA_PIXEL_MODE_SIXELS  # To display a high resolution image
              #config.pixel_mode = chafa.PixelType.CHAFA_PIXEL_RGBA8_UNASSOCIATED  # To display according to
              # terminal's output controls use config.pixel_mode = chafa.PixelMode.CHAFA_PIXEL_MODE_KITTY
              # to display a high resolution image.
              #config.pixel_mode = chafa.PixelMode.CHAFA_PIXEL_MODE_KITTY
              #config.pixel_mode = chafa.PixelMode.CHAFA_PIXEL_MODE_SYMBOLS  # To add symbols to image
              #config.pixel_mode = chafa.PixelType.CHAFA_PIXEL_BGRA8_UNASSOCIATED
              #-------------------------------------------------------------------------------------------------------

              # Configure the ideal canvas geometry based on our FONT_RATIO, when using chafa.loader or PIL.
              #config.calc_canvas_geometry(
              #image.width,
              #image.height,
              #FONT_RATIO
              #)

              # Set symbol map for visibility:
              # add 0's and 1's and quantum states to the map, relevant to the topic of this QFLCA program
              symbol_map = chafa.SymbolMap()
              symbol_map.add_by_range("a", "f")

              bands  = len(image.getbands())

              # Put image into correct format
              pixels = image.tobytes()
              canvas = Canvas(config)

              # Draw the intro image or graphics, when using chafa.loader.
              '''canvas.draw_all_pixels(
              image.pixel_type,
              image.get_pixels(),
              image.width, image.height,
              image.rowstride
                   )'''

              # Draw the intro image or graphics, when loading image using PIL.
              canvas.draw_all_pixels(
              PixelType.CHAFA_PIXEL_RGBA8_PREMULTIPLIED, # Other example is .CHAFA_PIXEL_RGBA8_UNASSOCIATED,
                                                         # given the image mode.
              pixels,
              image.width, image.height,
              image.width * bands
                   )
              #-----------------------------------------------
              #  Drawing Chars on Canvas.
              #-----------------------------------------------
              QP_rand = random.random()  # Returns a random number between 0 and 1.
              CP_rand = random.randint(0, 1) # Returns a random number denoting the classical state.
              Int_rand= random.randint(0, 3) # Returns a random number denoting the quantum state.

              for pixel in canvas[4,::10]:
                   pixel.char = "1"

              for pixel in canvas[5,::10]:
                   pixel.char = "0"

              #i = 0  # For drawing a sequence of chars or numbers on canvas.
              #for pixel in canvas[:-3:-1, 3]:
              #     pixel.char = str(i)[0]
              #     i += 10
```

```
            print(canvas[3,55].char)
            canvas[3,55].char = "H" # Denotes the H gate in the partial QDF circuit.
            ## Superposition state is obtained by Hadamard gate (H) by
            ## H|0⟩ =(|0⟩ +|1⟩ )/sqrt(2) and H|1⟩ =(|0⟩ -|1⟩ )/sqrt(2).

            s = str(round(QP_rand, 2)) # Round the random value to two decimal points and convert to string.
            c = 7 # Column cell number to draw a char.

            if img == "./__img__/1D-QDFcircuit-sprite/1D-circuit-05.png":
                 s = "0.66"  # This is the expected P value >= 2/3 for a strong QDF prediction.

            for r in s[0:3]:
                 canvas[c,55].char = "0"
                 canvas[c,56].char = "."
                 canvas[c,57].char = r # Draw the first three chars on canvas.

            canvas[3,82].char = "X"  # Denotes the X gate in the partial QDF circuit.
            canvas[c,82].char = CP_rand

            #-------------------------------------------------------------------------
            # The following is for terminal output and image info printouts per
            # a cleared terminal line.
            #-------------------------------------------------------------------------
            # clear_line(config.height) # Disable this if you want to see all frames in a sequenced list.
            clear_line(config.width) # Disable this if you want to see all frames in a sequenced list.

            output = canvas.print()  # Default option.

            #-------------------------------------------------------------------------
            # Write and Print images on the terminal.
            #-------------------------------------------------------------------------
            # Write picture
            print(output.decode(), Back.RESET), sleep(0.05)  # without decoding is raw data of the image.

            #if QP_rand > 0.5:   # Qubit P values printed in color for the corresponding circuit gates.
            #     print(fg.green + s, Back.RESET)
            #elif QP_rand <= 0.5 and img != "./__img__/1D-QDFcircuit-sprite/1D-circuit-05.png":
            #     print(fg.red + s, Back.RESET)
            #elif QP_rand <= 0.5 and QP_rand > 0.48: # Indicates P values approximating superposition.
            #     print(fg.purple + s, Back.RESET)
            #elif img == "./__img__/1D-QDFcircuit-sprite/1D-circuit-05.png":
            #     print(fg.green + s, Back.RESET)

            # Print animated P's of gates in the 1D QDF circuit after logging all the animation frames.
            file='intro-shell_output.txt' # The file to read/write from the terminal.
            chars.append(s)
            chars = list(dict.fromkeys(chars)) # Removes duplicate values from recurring image frames.

            with open(file, 'w') as file_to_write:
                 # Write information about the original image sequence and final image frame.
                 file_to_write.write(str(chars) + f"\n * Last animated step image: {img}" +
                                     f"\n * Size: {wid} x {hgt} pixels" + f"\n * Mode: {mode}\n"
                                     + str(datetime.datetime.now())) # date output entry to the file.
```

▼ intro_anim_print ( ) function animation end code:

Prints the stats information on the intro animation frames after successfully concluded.

- ▼ Source code in QAI-LCode_QFLCC.py

```
def intro_anim_print():
#-------------------------------------------
# This function prints the intro_anim results
# from its log file upon animation conclusion.
#-------------------------------------------
  with open("intro-shell_output.txt", "r") as input_file:
    for i in range(1):
         head = next(input_file).strip()
         print(fg.black+ bg.orange+"Animated P's of the QDF circuit, frame-by-frame = "+Back.RESET)
         print(fg.yellow, head)

    lines = input_file.readlines()[:-1]

    # Clean up the newlines in the lines of data.
    lines = [line.rstrip('\n') for line in lines]

    # Split each line at the "," character.
    lines = [line.split(',') for line in lines]
    print(fg.lightgreen, lines, bg.green +fg.yellow + Back.RESET)
    print(bg.blue + Fore.BLACK + str(datetime.datetime.now()) + Back.RESET)
  input_file.close()

  print(bg.red+fg.yellow +
        f"<<........ 1D QDF circuit animation as an update to the QFLCA-QFLCC v.1.0, has successfully concluded ........>>")
  print(f"<<........ Next... QFLCA Program's Prompt & Command Begins! ........>>", Back.RESET)
  print(Fore.LIGHTYELLOW_EX + f"=========================================================================", Back.RESET)
```

▼ QFLCC_program ( ) function code:

To end the current QFLCC program step engaged by the user.

- ▼ Source code in QAI-LCode_QFLCC.py

```
def QFLCC_program():
    while True:
        if random.randint(1, 1000) == 500:
            raise ExitMyProgram
        """- QFLCC program steps prior are more complicated than shown here. This is to
        end the current QFLCC program step engaged by the user."""
```

▼ main_exit ( ) function code:

Run the ending of the current QFLCC program step.

- ▼ Source code in QAI-LCode_QFLCC.py

```
def main_exit():
    ########################################################
    # Run the ending of the current QFLCC program step
    # and catch the ExitMyProgram exception.
    ########################################################
    global entry_stage, sim_state # Any exit entry of the program is reset back to 0.

    try:
        QFLCC_program()
    except ExitMyProgram:
         print(fg.yellow+Back.RED+"The program is terminated manually!")
         # Log the exit of the program.
         entry_stage = -1 # This value resets entry to 0 via its
                          # default increment += 1.
         entry_stage += 1
         file_ = open('shell_output.txt', 'a')
         now = datetime.datetime.now() # Date the I/O file entry
         subprocess.run("echo {}- Checkpoint logged on {} for all programs: \
///-PROGRAM TERMINATED-----///".format(entry_stage, now.strftime("%Y-%m-%d %H:%M:%S")),
shell=True, stdout=file_)
         raise SystemExit
```

▼ set_volume ( ) function code:

Sets volume according to desktop's master volume.

- ▼ Source code in QAI-LCode_QFLCC.py

```
def set_vol(new_volume):
    ############################################################
    # The volume function sets volume according to your OS.
    # This option is available as 'v' or 'volume' during program.
    ############################################################
        p.press('volumedown', presses = 50) # Sets volume to zero.
        time.sleep(0.5) # Using time.sleep to space the presses.
        x = math.floor(new_volume / 2) # Setting the amount of presses required.
        p.press('volumeup', presses = x) # Setting the volume.
        winsound.PlaySound("SystemQuestion", winsound.SND_ALIAS) # Sound test.
        """- End of volume settings."""
```

▼ prompt ( ) function code:

Prompting code for a user input.

- ▼ Source code in QAI-LCode_QFLCC.py

```
def prompt():
###################################
# Prompting code for a user input #
###################################
    global __help__, promptIn, site_name, entry_stage # File state/flag is 0 or 1
    # to see if dataset files are present and to be copied to the current directory.
    global dir_flag # Directory flag for dataset directory access in safe mode when set to 1.
    __help__ = 'h'

    while True:
        print(fg.lightgreen + "\r< ", end=""), sleep(1.1)
        print("\r> ", end=""+Fore.RESET), sleep(1.1)
        try:
             promptIn = str(input())
             # Respond to the user's choice or command.
             if promptIn == 'n' or promptIn == 'next':
                print("Next...\n")
                break
             elif promptIn == 'v' or promptIn == 'volume':
                vol = float(input("Input sound volume (° ͜ʖ °)🔊  between [🔇 = 0 for muted,\
and 🔊 = 100 for loudest]:"))
                set_vol(vol)
                prompt() # Restart.
                if vol < 0:
                       print(fg.red +"Out of range or wrong value entered! Readjusted to muted or 0!")
                       prompt() # Restart.
                if vol > 100:
                       print(fg.red +"Out of range or wrong value entered! Readjusted to maximum or 100!")
                       prompt() # Restart.
                break
             elif promptIn == 'b' or promptIn == 'begin' or promptIn == 'r' or promptIn == 'restart':
                 print('Restarting program...☝(° ͜ʖ °)☝')
                 subprocess.run(["python", "QAI-LCode_QFLCC.py"])  # Restart program.
                 break
             elif promptIn == 'cls' or promptIn == 'clear':
                 os.system('cls') # Clear screen.
             elif promptIn == 'website' or promptIn == 'site' or promptIn == 'about' or promptIn == 'web':
                 site_name = "about-site" # This html file is stored in the same site folder to read.
                 site_doc() # Load website.
             elif (promptIn == 'dir' or promptIn
                   == 'sm dir') and (dir_flag == 1): # Active only in safe mode directory environment when set to 1.
                 safeMode_dir() # Run safe mode directory environment during simulation and dataset anlysis.
                 dir_flag == 0 # Reset flag until next time requested by the user to access dataset files.
                 break
             elif (promptIn == 'dir' or promptIn == 'sm dir') and (dir_flag == 0):
                     print(f"🚫 This command is supported only in the QFLCA's Safe Mode (SM:>>) environment!\
\n\x1B[4m-Enter 'dir' at the dataset analysis & simulation stage after QFLCC directory installation!\x1B[0m"
+ fg.lightgreen)
                     continue
             elif promptIn == __help__ or promptIn == 'help':
                 userHelp()  # Show help.
                 continue
             elif promptIn == 'e' or promptIn == 'exit':
                 # Play Windows exit sound.
                 winsound.PlaySound("SystemExit", winsound.SND_ALIAS)
                 print('Exiting program...👋 (° ͜ʖ °)👋  Goodbye!')
                 entry_stage = 0
                 main_exit() # Terminate program.
             else:
                print('Input command or response. For help, enter \'h\'... ')
                continue
        except ValueError:
             main_exit()
             print("Invalid input. Please enter a response.")
```

▼ userHelp ( ) function code:

Helping tips code for the user.

- Source code in `QAI-LCode_QFLCC.py`

```
    def userHelp():
         # Play Windows question sound and display recalled tips to the user.
         winsound.PlaySound("SystemExclamation", winsound.SND_ALIAS)
         print('Input tips: \n-Enter \'n\' key for the \'next\' message or input. \n-Enter \'h\' or \'help\' \
    to display these tips. \n-Enter a file number when an uploaded file list is displayed to view result. \
    \n-Enter \'dir\' or \'sm dir\' at the dataset analysis & simulation stage after QFLCC directory installation.*\
    \n*-This command is supported only in the QFLCA\'s Safe Mode (SM:>>) environment!\
    \n-Enter \'site\' or \'web\' to view the \'website\' \'about\' this program or project. \
    \n-Enter \'b\' or \'r\' to \'restart\' or \'begin\' program. \n-Enter \'v\' or \'volume\' for sound volume \
    change during program. \n-Enter \'cls\' or \'clear\' to clear screen. \n-Enter \'e\' to \'exit\' the program.')
```

Expand/Collapse All...

```
* Click Expand All to view all code blocks from QAI-LCode_QFLCC.py on this page.
* Click Collapse All to selectively view a code block from QAI-LCode_QFLCC.py on this page.
```

Expand All | Collapse All

QFLCA/QFLCC dataset files installation, P analysis and results for a selected file...

To prompt user, log steps, file installation, dataset file content analysis (P's), results and errors. Function() list from top to bottom as ordered:

QFLCC prompting user code to call prompt ()...

```
QFLCC prompting user code: from QAI-LCode_QFLCC.py
```

```
    ###################################
    # Prompting step starts from here
    ###################################
    print(Back.LIGHTBLACK_EX + fg.lightgreen+ f"When this input signal, > or < switch appears: ")
    print("it means you can input a keyboard character then press [Enter], or input response, then \
    press [Enter] to a program query. For more information, enter \'h\'...")

    def userHelp():
         # Play Windows question sound and display recalled tips to the user.
         winsound.PlaySound("SystemExclamation", winsound.SND_ALIAS)
         print('Input tips: \n-Enter \'n\' key for the \'next\' message or input. \n-Enter \'h\' or \'help\' \
    to display these tips. \n-Enter a file number when an uploaded file list is displayed to view result. \
    \n-Enter \'dir\' or \'sm dir\' at the dataset analysis & simulation stage after QFLCC directory installation.*\
    \n*-This command is supported only in the QFLCA\'s Safe Mode (SM:>>) environment!\
    \n-Enter \'site\' or \'web\' to view the \'website\' \'about\' this program or project. \
    \n-Enter \'b\' or \'r\' to \'restart\' or \'begin\' program. \n-Enter \'v\' or \'volume\' for sound volume \
    change during program. \n-Enter \'cls\' or \'clear\' to clear screen. \n-Enter \'e\' to \'exit\' the program.')

    process = Thread(target = prompt)  # Invoke prompt() function by creating and executing threads
    process.start()
    #__space__ = input("Input something:")
    #if input():
    process.join() # Wait for the thread to terminate

    if entry_stage == 0:  # This makes sure when main_exit is called, program exit is executed without bypassing it.
         main_exit()

    """- QFLCC code continues."""
    #-------------------------------------------------------------------------------------------
    # Code Use and Copyright Information
    #-------------------------------------------------------------------------------------------
    __ORCID__ = "ORCID: https://orcid.org/0000-0003-1037-018X"
    __copyright__ = "Copyright © 2022--2024"
    __license__ = "Creative Commons (CC BY-NC-ND)"
    __author__ = "Philip B. Alipour"
    __location__ = "@ ECE Dept. University of Victoria, Victoria BC, Canada"
    __version__ = "ver. 1.0"
    __description__ = "This QFLCC version of the QAI-LCode is basic. More revisions to come, as the \
    dataset grows on the number of trials on 1 or more QDF circuits occurs relative to improvements \
    of dataset representation of quantum and classical parameters in observing a thermodynamic system \
    based on Refs. [1-3,5] of the Data in Brief, Elsevier J article."
    print(Back.BLUE + fg.lightgreen + __description__, "\n")
    print(Back.RESET+"Copyright information in Python code:\n")
    print("\033[0m"+__license__ , __author__, __location__+',', fg.blue+"\x1B[4m"+__ORCID__+"\x1B[0m",
          __copyright__, __version__, fg.pink+bg.lightgrey)
    sleep(5)

    ########################################
    # Main dataset file directory installation
    # and listing steps starts here.
    ########################################
    from tqdm import tqdm  # This is for progressbar display.

    ibmq_f=0 # File flag set to 0 by default for IBMQ files if as not IN
    qi_f=0   # File flag set to 0 by default for QInspire files as not IN
    dir_flag = 0 # Directory flag is set to 1 for dataset directory access in safe mode.
                 # Installing files is building the directory, so it is set to 0 here.

    print(os.getcwd())

    ppath = Path(os.getcwd()).parent # This points to the parent path of current directory.
    cpath = Path(os.getcwd())  # This is the current path.
    print(ppath.absolute())
    IBMQ_dir = os.path.join(ppath, 'IBMQ\\test\\')
    QI_dir = os.path.join(ppath, 'QI\\test\\')
    QI_files = os.listdir(QI_dir) # Fetching the list of all the files.
    IBMQ_files = os.listdir(IBMQ_dir) # Fetching the list of all the files.
    print('Importing IBMQ/QI data files: ', IBMQ_files, QI_files)  # Prints from e.g., C:/here/my_dir

    pool = ThreadPoolExecutor(max_workers)

    file_ = open('shell_output.txt', 'w+')
    subprocess.run("echo ///--- QFLCC Shell Log_file START ---///", shell=True, stdout=file_)

    now = datetime.datetime.now() # Date each I/O file entry.

    try:
       # with MultithreadedCopier(max_threads=16) as copier: # this is to achieve e.g., ~35x speedup on a 32 core machine.
       # with ThreadPool(4) as copier:
        with pool as executor:
                for file_name1 in QI_files:
                 future = executor.submit(shutil.copy, QI_dir + file_name1, cpath)  # future = pool.submit(my_task, argument)
                                                                                    # does not block.
                 qi_f=1
                for file_name1 in IBMQ_files:
                    future = executor.submit(shutil.copy, IBMQ_dir+file_name1, cpath)
                    ibmq_f=1
                value = future.result() # This one blocks.
                print(Back.RED + fg.black
                      + f'Thread ---> <{value}> <--- got {future}. Maximum threads initiated: {int(max_workers)}')
                pool.shutdown()

        print('All IBMQ/QI data files imported successfully!')
        #----------------------------------
        # File directory listing code
        #----------------------------------
    except Exception as e:
        for i in tqdm(range(0, 10), ncols = 100, desc =Back.RESET+"Progress: "+Fore.RED): # Progress level of a copy operation.
             time.sleep(.05)
        print('1 or more IBMQ/QI data files import failed!')
        qi_f=0
        ibmq_f=0
        print(e)

    print(Back.RESET+'||||||||||||||||||||||||||||||||||||||||') #, fileresult, sep="\n")
    print(Fore.YELLOW + 'You may view any of the files listed below by clicking on the file or [Ctrl + Mouse_click] option:')

    res = os.listdir()
    fileresult = []

    for (idx, st) in enumerate(res, 1):
        print(bg.green+'This is file # {} for {}'.format(idx, st.split('/')[-1]))
        time.sleep(.05)
        fileresult.append(st)
        for i in tqdm(range(0, 1), ncols=100, desc =Back.RESET+f"Progress: {st}"+fg.lightcyan + Style.BRIGHT): # Progress level of copy operation.
         # For any error related to this line on colors, install this: pip install tqdm==4.59.
         time.sleep(.05)
         subprocess.run("echo {}- Checkpoint logged on {} for file # {} \
    as {}".format(idx, now.strftime("%Y-%m-%d %H:%M:%S"), idx, fileresult[idx-1]), shell=True, stdout=file_)

    #res=os.listdir().index() # Enable this line (with its print next) to have clickable files to load and view
                             # on screen (not in terminal).

    print(Style.RESET_ALL + fg.orange + bg.cyan +"||||||||||||||||||||||||||||||||||||||||")
    #print(*res, '|||||||||||||||||||||||||||||||||||||||||||||||||', f'The current directory total listed files = {len(res)}', sep="\n")
    print('||||||||||||||||||||||||||||||||||||||||||||',
          f'The current directory total listed files = {len(res)}', sep="\n"), sleep(2)
    print("||||||||||||||||||||||||||||||||||||||||||||||||"+Back.RESET)

    #-------------------------------------
```

```
# File directory listing code cont.
#-------------------------------------
entries=[]
```

▼ file_reader ( ) function code:

ℹ File directory listing code to read specific files in the installed directory: .csv, .py, .txt, .json, .html. For analysis, .csv is the main dataset to read and analyze.

- ▼ Source code in QAI-LCode_QFLCC.py

```
def file_reader():
#-------------------------------------------------------------------
# File directory listing code cont. is by calling this function.
#-------------------------------------------------------------------
 global entries, selected_path, selection
 while True:
  entries = []
  # Build a list of files and folders in the working directory.
  current_folder = Path.cwd()
  # If we're not already in the root, the first entry is the parent folder.
  if current_folder.parent != current_folder:
    entries.append(Path('..'))
  entries += current_folder.glob('*')
  # Print a numbered list of files and folders
  print(fg.cyan+f'Listing contents of {current_folder}')
  for i, path in enumerate(entries):
    print(f'{i}: {path.name}')
  # Get user input. Valid input is the word 'quit' or a number that corresponds
  # to an index in the list of files and folders
  print(fg.lightgreen + "Select a file # to see its contents in form of text before analysis, or enter \
\'n\' for next to analyze file contents:")
  print(fg.lightgreen + "\r< ", end=""), sleep(1.1)
  print("\r> ", end=""+Fore.RESET), sleep(1.1)
  user_input = input('')
  if user_input == 'n' or user_input =='next':
    break
  elif user_input == '' or user_input =='h':
     userHelp() # Give user prompting tips.
     prompt() # Prompt user.
  elif user_input == 'v' or user_input == 'volume':
                vol = float(input("Input sound volume (° ͜ʖ °)👂 between [🔇 = 0 for muted,\
 and 🔊 = 100 for loudest]:"))
                set_vol(vol)
                file_reader() # Restart.
                if vol < 0:
                       print(fg.red +"Out of range or wrong value entered! Readjusted to muted or 0!")
                       file_reader() # Restart.
                if vol > 100:
                       print(fg.red +"Out of range or wrong value entered! Readjusted to maximum or 100!")
                       file_reader() # Restart.
  elif user_input == 'b' or user_input == 'begin' or user_input == 'r' or user_input == 'restart':
                  print('Restarting program...☝(° ͜ʖ °)☝')
                  subprocess.run(["python", "QAI-LCode_QFLCC.py"])  # Restart program.
                  break
  elif user_input == 'cls' or user_input == 'clear':
                  os.system('cls') # Clear screen.
  elif user_input == 'e' or user_input == 'exit':
                  # Play Windows exit sound.
                  winsound.PlaySound("SystemExit", winsound.SND_ALIAS)
                  print('Exiting program...✋ (° ͜ʖ °)✋  Goodbye!')
                  print(fg.yellow+Back.RED+"The program is terminated manually!"+Back.RESET)
                  sys.exit(1) # For abnormal termination, prefetch exception and force this exit.
  try:
    selection = int(user_input)
  except ValueError:
    print(fg.orange+'Invalid user input (please enter a number)'), sleep(1.1)
    continue
  if selection >= len(entries):
    print(fg.red+'Invalid user input (invalid number)'), sleep(2.1)
    continue
  # If the selected path is a folder, then change the working directory to
  # that folder.
  selected_path = Path(entries[selection])
  if selected_path.is_dir():
    os.chdir(selected_path)
    continue
  # Otherwise, try to read the file contents as if they are text. If this fails
  # print a warning message.
  try:
    file_contents = selected_path.read_text("UTF-8")
  except UnicodeDecodeError:
    print(Fore.LIGHTYELLOW_EX + f'{selected_path.name} is not a text or csv file'), sleep(2.1)
    continue
  # Otherwise print the file contents
  print(fg.black+bg.lightgrey + file_contents +Back.RESET)
  print(fg.lightgreen + "\r< ", end=""), sleep(1.1)
  print("\r> ", end=""+Fore.RESET), sleep(1.1)
  user_input = input('')
```

▼ filenum_call ( ) function code:

ℹ For dataset selection as a file to choose for analysis of its contents or recorded P's.

- ▼ Source code in QAI-LCode_QFLCC.py

```python
def filenum_call():
#---------------------------------
# This function takes in user's
# file number request.
#---------------------------------
     global filenum, idx, entries, fileresult

     # Take file index value from user
     print(fg.orange +bg.cyan +"||||||||||||||||||||||||||||||||||||||||")
     print(f'The current directory total listed files = {len(entries)-1}', sep="\n")
     print("||||||||||||||||||||||||||||||||||||||||"+Back.RESET)
     filenum=input(fg.yellow+'Enter the relevant file # as {*.csv, *.txt, *.png} for analysis from the list: ')
     filenum=int(filenum)  # Typecast (convert) string datatype into integer.
     idx=filenum
     print("\n")
     print(Fore.GREEN + 'Selected file number is {} for {}'.format(idx, fileresult[idx-1]))

     if dir_flag == 1: # if Safe Mode is enabled then maintain a restricted yet error tolerant
                       # safe mode environment for the user to select a dataset file form directory.
        print(Back.LIGHTGREEN_EX + Fore.YELLOW +
            "\033[1m<--- SAFE MODE DATASET ANALYSIS ENABLED --->\033[0m" + Back.RESET), sleep(2)
        print(f"{fg.lightgreen}Press any key to continue...")
        sys.stdout.write('SM:>>...')
        sys.stdout.flush()
        os.system("pause >nul")
```

▼ csv_analyzer ( ) function code:

For dataset analysis, here as .csv to analyze its recorded P's.

- ▼ Source code in `QAI-LCode_QFLCC.py`

```python
def csv_analyzer():
#-------------------------------------
# Open the QFLCA "*.csv" for analysis.
#-------------------------------------
   global pngfile

   with open(fileresult[idx-1], 'r') as x:
         pngfile=Path(f'{fileresult[idx - 1]}').stem
         sample_data = list(csv.reader(x, delimiter=","))
         print(Fore.RED + Back.YELLOW +
               f'Sample_data initiated... selected file for analysis from index value = 0 to {len(entries)-1} is: {idx}', sep="\n")
         print(Fore.CYAN + Back.RESET + '\033[1;4m' + '- User menu options are limited to Basic QFLCC program options.'+ '\033[0m')
         print(Fore.LIGHTCYAN_EX + '\033[1;4m' + '- Table data list:' + '\033[0m'+ Fore.RED + Back.CYAN + Fore.RED)
         subprocess.call([f'{pngfile}.png'], shell=True) # After stem from the selected csv filename, show the relevant
                                                         # histogram and circuit with the csv file being analyzed.
         subprocess.call([f'{pngfile}_H.png'], shell=True)
         sample_data = np.array(sample_data) # Import the P data into an array.
         print(sample_data)

         for row in sample_data:
               y=row[0]  # Store array's element value in y before converting csv left column data to a float value.
               print(row[0] + Back.GREEN)
```

▼ file_ext_call ( ) function code:

Call this function to setup and reconfigure r/w switches on the selected dataset file (extension-based).

- ▼ Source code in `QAI-LCode_QFLCC.py`

```python
def file_ext_call():
  #-------------------------------------------------------------------------
  # Call this function to setup and reconfigure r/w switches
  # on the selected dataset file (extension based).
  # * See group of file extensions to operate r/w on prior to this function.
  #-------------------------------------------------------------------------
  global filesFlag
  ###########################################################
  # Code reconfig options instead of the while loop:
  # r = root, d = directories, f = files
  # for r, d, f in os.walk(thisdir):
  #   for dirfile in f:
  #     if dirfile.endswith(ext0) and (file_extension=='.txt'
  #        or file_extension=='.docx'):
  # ... as follows.
  ###########################################################
  while filesFlag == 0:  # File state/flag is 1 in case of dataset files are present/copied in the current directory
    #for item in range(1, 2):
     if (file_extension=='.txt' or file_extension=='.docx' or file_extension=='.bin') and filesFlag==0:
          # print(os.path.join(r, dirfile))
          print(Fore.LIGHTYELLOW_EX + f'1- A {file_extension} file chosen to analyze:', thisfile)
          print('2- Temporarily read text file content available...')
          # print(os.path.join(r, dirfile))
          # csv_analyzer()
          file = open(thisfile, 'r', encoding='utf-8')
          content= file.read()
          print(fg.black+bg.lightgrey + content+ Back.RESET)
          print(Fore.LIGHTYELLOW_EX + 'Select another file for r-w analysis:...'), sleep(2.1)
          file_reader()
          print("Textual data to read...")
          filenum_call()
          filesFlag==0
          break
     elif (file_extension=='.png' or file_extension=='.jpg') and filesFlag==0:
          print(Fore.LIGHTYELLOW_EX + f'1- A {file_extension} file chosen to analyze:', thisfile)
          print('2- Temporarily image file content available...')
          # print(os.path.join(r, dirfile))
          pngfile=Path(f'{thisfile}').stem
          with open(thisfile, 'r') as x:
               subprocess.call(f'{pngfile}.png', shell=True)
               subprocess.call(f'{pngfile}.jpg', shell=True)
          file_reader()
          filenum_call()
          print("Image file to view...")
          filesFlag==0
          break
     elif file_extension=='.csv':
          print("*.csv being read for analysis...")
          csv_analyzer()
          filesFlag==1
          break
```

▼ **next ( ) function** code:

ℹ To flush out the buffered data on continuous 'n' character keystrokes on keyboard reaching the next program step.

- ▼ Source code in `QAI-LCode_QFLCC.py`

```
824     def __next__():
825         while True:
826              if keyboard.is_pressed("n"):
827                   print('Next...')
828                   sys.stdout.flush()
829                   break
830              '''This flushed out the buffered keystrokes of character 'n' from memory
831                 when key pressed by user continuously up to after reaching the 'Next...'
832                 printout message displayed on screen...'''
```

▼ PAnalysis ( ) function code:

ℹ To analyze the selected dataset by user, conduct its basic analysis of P's from a QDF circuit measurement.

- ▼ Source code in `QAI-LCode_QFLCC.py`

```
872     def PAnalysis():
873     ######################################
874     # P analysis of the selected dataset. #
875     ######################################
876           global df2, df3
877           if  (sample_data.max(axis=0)[1] < 0.5 and sample_data.max(axis=0)[1] > 0.2):  # Conditions to
878               # assign values to e.g., a weak classical min(ΔP) = max(P) of a quantum outcome range (from
879               # the qubit dataset). Complement here is min(P) of the dataset if the user inputs for the opposition.
880                df2=df.loc[((df['probability'] < 0.5) & (df['probability'] > 0.27)), :].binary_string[:]
881                df3=df.loc[((df['probability'] < 0.5) & (df['probability'] > 0.27)), :].probability[:] # Classify
882                # in the given included range of n:m
883           elif (sample_data.max(axis=0)[1] < 1 and sample_data.max(axis=0)[1] >= 0.5):  # Conditions to assign values
884                 # to e.g., a strong classical max(ΔP) = min(P) of a quantum outcome range (from the qubit dataset).
885                 # Complement here is max(P) of the dataset if the user inputs for the opposition.
886                df2=df.loc[((df['probability'] < 1) & (df['probability'] >= 0.5)), :].binary_string[:]
887                df3=df.loc[((df['probability'] < 1) & (df['probability'] >= 0.5)), :].probability[:]
```

▼ qdf_PAnalysis ( ) function code:

ℹ To analyze the selected dataset by user, display its QDF analysis results of P's from a QDF circuit measurement.

- ▼ Source code in `QAI-LCode_QFLCC.py`

```
def qdf_PAnalysis():
 global qdf, bitpair, df2bin, df3Mem, dfcomp, bitcomp # Main global variables to compute classical bit and qubit P's.
 if (sample_data.max(axis=0)[1] < 1):
    PAnalysis()
    print(Fore.GREEN + Back.RESET + 'Strong Prediction binary string is: ' + Fore.YELLOW,
          df2.to_string(index=False, header=False) + Fore.GREEN + ', with a P value of'
          + Fore.YELLOW, df3.to_string(index=False, header= False))
    df3Mem =df3.to_string(index=False, header= False)
    ###
    df2bin= df2.to_string(index=False, header=False)

    qdf0=[df2bin[i:i+2] for i in range(math.floor(len(df2bin)/2)-1, math.ceil(len(df2bin)), 2)] # Use this formula
    # to satisfy conditions of relevant outer binary string relative to central bit-pair according to Eq. (20) or
    # Ref. [1] of the current Elsevier J. paper (given your/the QDF circuit design)...
    qdf1=[df2bin[i:i+2] for i in range(math.floor(len(df2bin)/2), math.ceil(len(df2bin)), 2)]
    qdf2=[df2bin[i:i+2] for i in range(math.floor(len(df2bin)/2)-1, math.ceil(len(df2bin)), math.floor(len(df2bin)/2))]

    qdf= qdf0 or qdf1 or qdf2 # Dynamic assignment of either df2bin range.

    if len(df2bin) <= 6 and qi_f == 1: # This if statement is designed to analyze QI dataset. One or more if statements
        # needed to be added for IBMQ or different circuit/dataset configurations with their recorded binary strength
        # Lengths in their tables (their *.csv structure).
        print(Fore.GREEN + 'Paired qubit (left list element) relative to classical bit output (right list element): '
              + Fore.YELLOW, qdf0)
        qdf= qdf0  # Assign resultant value of qdf0 to qdf.
    elif len(df2bin) > 6 and qi_f == 1:
        print(Fore.GREEN + 'Paired qubit (left list element) relative to classical bit output (right list element): '
              + Fore.YELLOW, qdf1)
        qdf= qdf1  # Assign resultant value of qdf1 to qdf.
    else:  # Depending on QDF circuit design and configuration, more elif statements can be added to suit a bit-pair
           # count and string foci as the one designed under IBMQ platform compared to QI (or QInspire) platform.
        print(Fore.GREEN + 'Paired qubit (left list element) relative to classical bit output (right list element): '
              + Fore.YELLOW, qdf2) # Focus on the maximum from the middle qubit pair, if we put i+3, is to denote
              # a pair + 1 qubit denoting up to three particles involved e.g. a photon Alice, Bob and the prize entangled.
              # Iterate over a range of bins that are multiples of 2 (i.e. the size of the split substrings) for row in sample_data:
        qdf= qdf2  # Assign resultant value of qdf2 to qdf.

    if dir_flag == 0:
        prompt()
    else:  # Maintain a restricted yet error tolerant safe mode environment for the user to select a dataset file form directory.
        print(Back.LIGHTGREEN_EX + Fore.YELLOW +
            "\033[1m<--- SAFE MODE DATASET ANALYSIS CONTINUES --->\033[0m" + Back.RESET), sleep(2)
        print(f"{fg.lightgreen}Press any key to continue...")
        sys.stdout.write('SM:>>...')
        sys.stdout.flush()
        os.system("pause >nul")
    ###############################################################################################
    # This part of the function prints the expected measurement output for different cases
    # on qubit pairs.
    ###############################################################################################
    print(Fore.RED + f'\x1B[1;4mNotes:\033[0m\n' + Fore.CYAN + f'-* When shown in histograms generated from quantum circuit, \
the rightmost bit is for the measured qubit with the lowest index (q[0]). The leftmost bit is for the qubit with the highest index \
as the most significant qubit.\n-* Measurement focuses on QDF circuit bit pairs given the circuit configuration \
== focus on the maximum from the middle qubit pair in a list of measurement results. The program loop iterates over a range of \
bins that are multiples of 2 (i.e. the size of the split substrings) for row in sample_data...'+ Fore.YELLOW)
    df2list = qdf0 or qdf1 or qdf2
    qubit_print=''
    if '01' in df2list[:-1]: # Print the verdict on the one element prior last.
          print(fg.yellow+'{P_|01>b(01)} = 1 == ES')
          qubit_print='{P_|01>b(01)} = 1 == ES'
    if '00' in df2list[:-1]:
          print(fg.yellow+'{P_|00>b(00)} = 0 == GS')
          qubit_print='{P_|00>b(00)} = 0 == GS'
    if '10' in df2list[:-1]:
          print(fg.yellow
                +f"{'{P_|10>b(10)} = 2 == ES within GS (QPT) or prize with lesser E value == some E loss or gain for Bob == uncertain or certain by swap gate when
          qubit_print=f"{'{P_|10>b(10)} = 2 == ES within GS (QPT) or prize with lesser E value == some E loss or gain for Bob == uncertain or certain by swap gate
    if '11' in df2list[:-1]:
          print(fg.yellow+'{P_|11>b(11)} = 3 == superposition or prize state entangled with Alice or Bob')
          qubit_print='{P_|11>b(11)} = 3 == superposition or prize state entangled with Alice or Bob'
    if '0b' in df2list[:]: #or '1b' in df2list[:-1]:
          print(bg.orange + fg.lightgreen, qubit_print, Back.RESET + fg.lightgreen + "The binary string "+ Fore.YELLOW
                + "\'..0b\' (classical bit)" + fg.lightgreen + " ends upon the least significant qubit (rightmost qubit).")
    if '1b' in df2list[:]:
          print(bg.orange + fg.lightgreen, qubit_print, Back.RESET + fg.lightgreen + "The binary string "+ Fore.YELLOW
                + "\'..1b\' (classical bit)" + fg.lightgreen + " ends upon the least significant qubit (rightmost qubit).")

    dfcomp = 1-df3  # Calculation of the P complement for the bitpair.
    np.seterr(all="ignore") # Suppress irrelevant/outdated floating point numpy errors.

    for bit in df2:
        if bit == "1":
            bit == "0"
        else:
            bit == "1"
    bitcomp = bit.replace("1", "2").replace("0", "1").replace("2", "0")

    if len(df2bin)>0 and qi_f==1:
        # The following lines cover the bitpair property between the qdf0 to qdf1 or qdfn as assigned above (as replaced bits).
        index = 0
        if qdf[index] == "01":
                bitpair = ["10"]
        if qdf[index] == "10": # We use the if statement rather than elif due to absolute conditions of the bitpair readout
                               # from the dataset, according to above if-statements and for-loop.
                bitpair = ["01"]
        if qdf[index] == "11":
                bitpair = ["11"]
        if qdf[index] == "00":
                bitpair = ["00"]
        if qdf[index] == "0b": # If classical state is = 0 in binary, result is conditioned to classical bit = 0.
                bitpair == "0"
        if qdf[index] =="1b": # If classical state is = 1 in binary, result is conditioned to classical bit = 1.
                bitpair == "1"

        print(fg.green+"Complement of the binary string is:"+fg.yellow, bitcomp, fg.green+", and the focused pair in it "
              +fg.yellow, bitpair, fg.green+"has a P' value of"+ Fore.LIGHTRED_EX, dfcomp.to_string(index=False, header=False))
 else:
    print(fg.red+'Erroneous or Tied P values!')
    subprocess.run(["python", "QAI-LCode_QFLCC.py"])  # Restart program.

 """ Complex code alternative for the line:
 if len(df2bin)>0 and qi_f==1:
         bitpair = [bitcomp[i:i+2] for i in range(math.floor(len(bit)/2)-2, math.floor(len(bit)/2), 2)] # This line covers the
                   # bitpair property between the qdf0 to qdf1 or qdfn as assigned above.
```

```python
            index = 0
            #bitpair = [qdf[index]] # Invert the bitpair result from the past do to compare with the last bitpair result when needed.
            #while index <= len(bitpair):
    elif len(df2bin)<=6 and qi_f==1:
            bitpair = [bitcomp[i:i+2] for i in range(math.floor(len(bit)/2)-1, math.floor(len(bit)/2), 2)] # This line covers the
                      # bitpair property between the qdf0 to qdf1 or qdfn as assigned above.
            print(fg.green+"Complement of the binary string is: "+fg.yellow, bitcomp, fg.green+", and the focused pair in it "
                   +fg.yellow, bitpair, fg.green+"has a P' value of"+ Fore.LIGHTRED_EX, dfcomp.to_string(index=False, header=False)) """

    if dir_flag == 1: # Maintain a restricted yet error tolerant safe mode environment for the user to select a dataset file form directory.
        print(Back.LIGHTGREEN_EX + Fore.YELLOW +
              "\033[1m<--- SAFE MODE DATASET ANALYSIS CONCLUDED --->\033[0m" + Back.RESET+ fg.lightgreen), sleep(2)
```

▼ safeMode_dir( ) function code:

To analyze the selected dataset by user within the Safe Mode (SF:>>) simulation environment. The limited commands compare the selected dataset results with the realtime simulated IBM QDF circuit results (dataset).

- ▼ Source code in `QAI-LCode_QFLCC.py`

```python
def safeMode_dir():
  ###############################################################################
  # Directory function restricted to select a different file by the user
  # for analysis in a restricted yet safe mode environment.
  ###############################################################################
  global dir_flag # Flag is set to 1 for dataset directory access in safe mode.
  global dfcompMem # Recall and reuse as global from PAnalysis() results.
  if dir_flag == 1:
       print(Back.LIGHTGREEN_EX + Fore.YELLOW +
             "\033[1m<--- SAFE MODE DIR ENVIRONMENT ENABLED --->\033[0m" + Back.RESET), sleep(2)
       filesFlag = 0
       file_reader()  # Execution flow of functions with dir and file flag settings
                      # must run sequentially...
       filenum_call()
       #filesFlag = 0
       filesFlag = 0
       with alive_bar(4, bar = 'blocks', manual=True) as bar:  # Print progress bar per Safe Mode step.
                                                               # Enables multiprocessing when needed.
        csv_analyzer()
        bar(.2)
        file_ext_call()
        bar(.3)
        dataframe()
        bar(.7)
        qdf_PAnalysis()
        bar(.9)
        dfcompMem = float(dfcomp.to_string(index=False, header=False)) # Convert the string representation
             # of string to float for future use/comparison/validation.
        #filesFlag==1 # This file state denotes all dataset files are present within this directory.
             # No need to overwrite in case of program restart or in safe mode (SM)...
        bar(1)
        ###  Progress concluded on the selected file by user and its preliminary analysis. ####
       #dir_flag == 0 # Reset flag until next time requested by the user to access dataset files.
       #prompt()
       #PAnalysis_model() # Continue IBM QDF circuit simulation comparison of its dataset with the selected one.
  else:
       print(Back.LIGHTGREEN_EX + Fore.YELLOW +
             "\033[1m<--- SAFE MODE DIR ENVIRONMENT DISABLED --->\033[0m" + Back.RESET), sleep(2)
       """- End of safeMode_dir function calls """
```

▼ PAnalysis_model( ) function code:

To run IBM QDF circuit simulation and analyze the selected dataset by user, comparing dataset results with the simulated IBM QDF circuit results (dataset). Dependencies: imports the QDF-LCode_IBMQ-2024.py module (or its codable version).

- Screenshots of the circuit simulation and user's selected dataset analysis after executing the `PAnalysis_model()` function:

```
*- Enter 'n' or 'next' to proceed calculating P's between the selected dataset and the simulated IBM QDF circuit results.
ⓘ  The datasets of the two QDF circuits are compared and validated with a verdict on how close their projection of
 events (prediction) is, given their QDF circuit configuration!
*-  ✅ Enter 'dir' or 'sm dir' to select another dataset to analyze in Safe Mode (SM:>>). SIMULATION WILL NOT RESTART!
*-  ✅ Enter 'n' or 'next' to continue in Safe Mode (SM:>>). QDF CIRCUITS SIMULATION_DATASET_ANALYSIS WILL START!
*- Enter any other key to change from SM:>> to regular prompt mode (>).
*- Enter 'r' in regular prompt mode (>) to 'restart' circuit simulation and dataset analysis. Enter 'h' for more options.
SM:>> n
SM:>> Next...
<--- QDF CIRCUITS SIMULATION_DATASET_ANALYSIS BEGINS --->
================================================================================================================================
Plot = [P samples of the IBMQ model circuit, if meets  ∝  p of the selected dataset file { QI_Exp_02.csv }
on pairwise  qubits]...
Then a strong vs weak p match, and the distance between QDF circuit events are determined.*
================================================================================================================================
* Computation model: ĸ Scalar ΨΦ Field Switch and Correlation Model, Ref. [1] of the DIB article.
  P(Ψ ⟷ ĸ²Ψ) = {P(Ψ ⟷Φ)} = P(b|ij)); |ij⟩ = |q_i q_j⟩ = |q₁q₂⟩.
================================================================================================================================

<-- P Sampling between IBM QDF Circuit and the Selected QDF Circuit Dataset -->

<-- P Results Sampled from { ibm-qdf-stats.txt } file are: -->
on 0: P(Total)          [1.0000000000000]
      ibmq qdf p(0101)  [0.6763916015625]
      ibmq qdf p(0111)  [0.2738037109375]
      ibmq qdf p(0001)  [0.0498046875000]
on 0:
|████████████████████████████████████████| 100% [3/3] in 2.2s (1.46/s)
================================================================================================================================
*- min(P) from { ibm-qdf-stats.txt } is performed by the QDF bit pairs { 0001 } as { ibm qdf } sample set #3 = 0.0498046875
*- max(P) from { ibm-qdf-stats.txt } is performed by QDF bit pairs { 0101 } as { ibm qdf } sample set #1 = 0.6763916015625
================================================================================================================================
P(Total)                        [1.0000000000000]
{ QI_Exp_02.csv } max(P(010100b))  [0.9716800000000]
{ QI_Exp_02.csv } comp(P(101011b))  [0.0283200000000]
{ QI_Exp_02.csv } min(P(010010b))  [0.0283203125000]
{ ibm qdf } min((P(0001))       [0.0498046875000]
{ ibm qdf } max((P(0101))       [0.6763916015625]
Ap of min(P)                    [0.0214843750000]
Ap of max(P)                    [0.2952880859375]
Ap of min(P) match              [0.9785156250000]
Ap of max(P) match              [0.7047116015625]
================================================================================================================================
*- Ap Data: ⚖ Validation Verdict ⚖  between the two dataset samples from { QI_Exp_02.csv , ibm-qdf-stats.txt }
   is a/an: Above Avg. Match {( 0.98 , 0.7 )}
================================================================================================================================
*- QDF Qubit Sets: ⚖ Validation Verdict ⚖  between the two dataset samples from { QI_Exp_02.csv , ibm-qdf-stats.txt }
   is a/an: Strong Match { 010100b , 0101 }
================================================================================================================================
*- QDF Circuits: ⚖ Validation Verdict ⚖  between the two QDF circuit configurations from { QI_Exp_02.csv , ibm-qdf-stats.txt }
   is a/an: Above Avg. Match
================================================================================================================================
 *- The IBM QDF circuit can be reconfigured for worst and best case Hamiltonian scenarios,
    given the expected QDF measurement outcomes from the collected QFLCA datasets for a QFLCC.
 Simulation completed on: 2024-03-30 17:41:18.391752
================================================================================================================================
<--- QDF CIRCUITS SIMULATION_DATASET_ANALYSIS CONCLUDED --->
> exit
Exiting program...🖖(° ͜ʖ °)🖖 Goodbye!
The program is terminated manually!
```


```
Ln 1092, Col 2   Spaces: 5   UTF-8   CRLF   {} Python   3.11.8 64-bit
```

- ▼ Source code in `QAI-LCode_QFLCC.py`

```python
def PAnalysis_model():
    ##########################################################################################################################
    #--- This function compares the mapped results between QDF circuits (IBMQ <--> QI).
    #--- SAFE MODE QFLCC AND,
    #--- QDF Circuit Model from IBMQ article file: QDF-LCode_IBMQ-2024-codable.py,
    # for python library purposes imported as a module:
    #--- QDF-LCode_IBMQ-2024.py (same code content as a duplicate of QDF-LCode_IBMQ-2024-codable, but will be overwritten
    # by IBMQ folder content if you modify or update code. Only edit code within the QDF-LCode_IBMQ-2024-codable.py file!).
    ##########################################################################################################################
    global sim_state, dir_flag, entry_stage, hline
    sim_state = ['']
    hline = "=========================================================================================================================================="
    print(f'''{Fore.LIGHTMAGENTA_EX + hline}\n An ideal QDF circuit I/O model running realtime producing an IBMQ-based QDF dataset is from the \
    \n QDF-LCode_IBMQ-2024.py file. This circuit is compared to the analyzed dataset to show how close \
    \n the match is for a desired Hamiltonian and expected measurement outcome as a point of reference.\
    \n{hline}''')
    #-------------------------------------------------------------------------------------------
    # <----- Display histogram results of the QDF circuit event P's and then printing the circuit
    # as a model to display for QDF datasets of other circuits being analyzed here (next version will).
    # Future release will have current P results of QInspire or any other QDF circuit configure the
    # IBM-QDF circuit through a QAI mapping, accordingly (subject of QAI lab for QFLCA) ----->
    #-------------------------------------------------------------------------------------------
    P = 1 # The total probability of QDF circuit events relative to classical states or
          # denoting the system's Hamiltonian (total energy).
    with alive_bar(3, bar = 'blocks', manual=True) as bar:   #To print progress bar on these dataset and circuit simulation analyses.
       print(Fore.LIGHTMAGENTA_EX + Back.LIGHTGREEN_EX+ f"Please Wait!...:")
       print(f"*- Program is processing the analyzed dataset circuit P's after:"+Back.RESET), sleep(1)
       bar(0.5)
       print(f"{Back.LIGHTGREEN_EX}*- Import + Simulate the IBM QDF circuit by\
    {{ Qiskit Aer Simulator, IBMQ Provider }} [QUANTUM MODE ENVIRONMENT] on this computer."+Back.RESET), sleep(1)
       bar(0.7)
       print(f"{Back.LIGHTGREEN_EX}*- Dataset P results are compared on the imported circuit by the {{ QDF-LCode_IBMQ-2024.py }} module\
    on this computer [SAFE MODE ENVIRONMENT]."+ Back.RESET + Fore.LIGHTMAGENTA_EX), sleep(1)
       bar(1.)

    # Log the start of simulation.
    entry_stage = 0
    sim_state[0] = " QDF CIRCUIT SIMULATION BEGINS "
    sim_log()
    #-------------------------------------------------------------------------------------------------------------------------
    ibm_qdf_module = importlib.import_module("QDF-LCode_IBMQ-2024-codable") # Unconventional call from the targeted module.
    bar(1.)
    #-------------------------------------------------------------------------------------------------------------------------
    # Log the prompt of simulation to the user to proceed, restart or exit program.
    entry_stage = 1
    sim_state[0] = " QDF CIRCUIT SIMULATION_DATASET_ANALYSIS PROMPT "
    sim_log()
    #-----------------------------------------------------------------------------
    # Selected dataset and the simulated circuit's dataset validation verdict starts here,
    # if user enters 'n' or 'next'...
    # ----------------------------------------------------------------------------
    print(Fore.LIGHTGREEN_EX + f"*- Enter 'n' or 'next' to proceed calculating P's between the selected dataset and the simulated\
    IBM QDF circuit results.")
    print(f"© {Fore.LIGHTMAGENTA_EX}\x1B[1;4mThe datasets of the two QDF circuits are compared and validated with a verdict on how\
    close their projection of \n events (prediction) is, given their QDF circuit configuration!\033[0m")
    print(f"{Fore.LIGHTGREEN_EX}*- ✅ Enter 'dir' or 'sm dir' to select another dataset to analyze in Safe Mode (SM:>>).\
    SIMULATION WILL NOT RESTART!")
    print(f"{Fore.LIGHTGREEN_EX}*- ✅ Enter 'n' or 'next' to continue in Safe Mode (SM:>>).\
    QDF CIRCUITS SIMULATION_DATASET_ANALYSIS WILL START!")
    print(f"{Fore.LIGHTGREEN_EX}*- Enter any other key to change from SM:>> to regular prompt mode (>).")
    print(f"*- Enter 'r' in regular prompt mode (>) to 'restart' circuit simulation and dataset analysis. Enter 'h' for more options.")

    sm_input = input(f"SM:>> {fg.yellow}")
    if sm_input == "dir" or sm_input == "sm dir":
         dir_flag = 1 # Directory flag is set to 1 for directory access in safe mode.
         safeMode_dir()
         pass
    elif sm_input == "n" or sm_input == "next":
         dir_flag = 1
         print(f"{Fore.LIGHTGREEN_EX}SM:>>{fg.yellow} Next..."), sleep(1)
    else:
         dir_flag = 0  # Directory flag is set to 0 in case of regular prompt mode (>) or 'r' to restart the simulation.
         prompt()

    print(Back.LIGHTGREEN_EX + Fore.YELLOW + "\033[1m<--- QDF CIRCUITS SIMULATION_DATASET_ANALYSIS BEGINS --->\033[0m" + Back.RESET)

    ibmq_result = 'ibm-qdf-stats.txt'
    with open(ibmq_result, 'r') as file:
           # Read the contents of the file
           content = file.read()

           # Extract all the p values from the content as float.
           N = re.findall(r'[-+]?(?:\d*\.*\d+)', content)
           n_list = list(map(float, N))

           # Convert all the metadata digits to str to sort out binaries.
           bin_list = list(map(str, N))  # This is to identify qubit binary strings.

           # Calculate quantum event p's for the total P.
           P = n_list[0]  # The P result is stored to the p_list.
           p1 = n_list[1] # 1st p result is stored to the p_list.
           pq1 = bin_list[4]
           p2 = n_list[2] # 2nd p result is stored to the p_list.
           pq2 = bin_list[5]
           p3 = n_list[3] # 3rd p result is stored to the p_list.
           pq3 = bin_list[6]

    ibmq_p_max = max(p1, p2, p3)  # Identify the max value from the p list.
    max_p_index = n_list.index(ibmq_p_max) # Store the index value for p_max from the ibmq_result file.

    ibmq_p_min = min(p1, p2, p3)  # Identify the min value from the p list.
    min_p_index = n_list.index(ibmq_p_min) # Store the index value for p_min from the ibmq_result file.

    # Set the default p color codes to 'white' as defined below in the p_color list until an if condition applies.
    p_color = [Style.BRIGHT + Fore.WHITE, Style.BRIGHT + Fore.WHITE, Style.BRIGHT + Fore.WHITE,
               Style.BRIGHT + Fore.WHITE, Style.BRIGHT + Fore.WHITE]

    if round(p1, 2) <= round(ibmq_p_min, 2): # Classify/color code min(p1).
         p_color[0] = Style.BRIGHT + Fore.RED
    if round(p2, 2) <= round(ibmq_p_min, 2): # Classify/color code min(p2).
         p_color[1] = Style.BRIGHT + Fore.RED
    if round(p3, 2) <= round(ibmq_p_min, 2): # Classify/color code min(p3).
         p_color[2] = Style.BRIGHT + Fore.RED
    if round(p1, 2) >= round(ibmq_p_max, 2): # Classify/color code max(p1).
```

```
            p_color[0] = Style.BRIGHT + Fore.CYAN
        if round(p2, 2) >= round(ibmq_p_max, 2): # Classify/color code max(p2).
            p_color[1] = Style.BRIGHT + Fore.CYAN
        if round(p3, 2) >= round(ibmq_p_max, 2): # Classify/color code max(p3).
            p_color[2] = Style.BRIGHT + Fore.CYAN

        print(f"{Fore.LIGHTMAGENTA_EX}{hline}\nPlot = [{Fore.LIGHTGREEN_EX}P samples of the IBMQ model circuit,\
        if meets {Fore.LIGHTYELLOW_EX} ∝ {Fore.LIGHTGREEN_EX} p of the selected dataset file\
        {{ {Fore.LIGHTYELLOW_EX + fileresult[filenum-1] + Fore.LIGHTGREEN_EX} }}\non pairwise \
        qubits{Fore.LIGHTMAGENTA_EX}]...\nThen a strong vs weak p match, and the distance between QDF circuit events\
        are determined.{Fore.YELLOW}* {Fore.LIGHTGREEN_EX}\n{hline}{Fore.LIGHTGREEN_EX}\
        \n{Fore.YELLOW}* Computation model:{Fore.GREEN} κ Scalar Ψ0 Field Switch and Correlation Model, Ref. [1] of the DIB article.\
        \n  {Fore.LIGHTCYAN_EX}P(Ψ ↔  κ²Ψ) = ⟨P(Ψ ↔ Φ)⟩ = P(b|ij)); |ij⟩ ≡ |q_i q_j⟩ ≡ |qᵢqⱼ⟩. {Fore.GREEN}\
        \n{hline}"), sleep(0.5)

        print(f'''{Fore.LIGHTMAGENTA_EX}\n\033[4m<-- P Sampling between IBM QDF Circuit and the Selected QDF Circuit Dataset -->\033[0m''')
        if dir_flag == 1:  # Read and display the printed circuit when dataset dir is in SAFE MODE.
              with open("ibm-qdf-circuit_output.bin", 'r', encoding='utf-8') as file_to_read:
                    print(f'{fg.black}{bg.lightgrey}\n'+file_to_read.read(),"\n<-- IBM QDF Circuit Sample Printed in SAFE MODE --> "
                          + Back.BLUE + str(datetime.datetime.now()) + Back.RESET)
                    file_to_read.close() # End reading and displaying the printed circuit.
        print(f'''{Fore.LIGHTMAGENTA_EX}\n\033[4m<-- P Results Sampled from {{ {ibmq_result} }} file are: -->\033[0m''')
        with alive_bar(3, bar = 'blocks' , manual=True) as bar:
         model_fig = tpl.figure() # Now plot histogram with horizontal bars for the computed probabilities.
         model_fig.barh([P, p1, p2, p3], [Style.BRIGHT + Fore.LIGHTGREEN_EX+"P(Total)", p_color[0]+f"ibmq qdf p({bin_list[4]})",
                                          p_color[1]+f"ibmq qdf p({bin_list[5]})", p_color[2]+f"ibmq qdf p({bin_list[6]})"],
                force_ascii=False), sleep(1)
         model_fig.show(), sleep(1)
         print('')
         bar(1)
         dir_flag = 0 # Reset flag until recalled.
         qdf_bit_pairs = ['',''] # To register which qdf_bit_pair has the max or min p.
         #-------------------------------------------------------------------------------------
         # Now classify and print which sampled qdf_bit_pair set from the
         # ibmq_result file is max_p and which min_p.
         if min_p_index == 1:
          qdf_bit_pairs[0] = pq1
         if min_p_index == 2:
          qdf_bit_pairs[0] = pq2
         if min_p_index == 3:
          qdf_bit_pairs[0] = pq3
         if max_p_index == 1:
          qdf_bit_pairs[1] = pq1
         if max_p_index == 2:
          qdf_bit_pairs[1] = pq2
         if max_p_index == 3:
          qdf_bit_pairs[1] = pq3

        print(Fore.YELLOW+f'{hline}')
        print(fg.purple+f'*- min(P) from {{ {ibmq_result} }} is performed by the QDF bit pairs\
        {{ {fg.red+ qdf_bit_pairs[0] +fg.purple} }} as {{ ibm qdf }} sample set {fg.yellow}#{min_p_index} = {ibmq_p_min}')
        print(fg.purple+f'*- max(P) from {{ {ibmq_result} }} is performed by QDF bit pairs\
        {{ {fg.lightgreen+ qdf_bit_pairs[1] +fg.purple} }} as {{ ibm qdf }} sample set {fg.yellow}#{max_p_index} = {ibmq_p_max}')
        print(Fore.YELLOW+f'{hline}')
        #-------------------------------------------------------------------------------------------------------
        # Recall and store which binary string had the minimum P from the table of the selected dataset (dataframe).
        global df_min, df_min_bin
        if  (sample_data.min(axis=0)[1] < 1):
                 df_min=df.loc[(df['probability'] <= sample_data.min(axis=0)[1]), :].binary_string[:]
                 df_min_bin = df_min.to_string(index=False, header=False)
        #-------------------------------------------------------------------------------------------------------
        deltaMatch_max = (1 - abs(ibmq_p_max - float(df3Mem)) ) # Calculate Δp of max(P) Match.
        deltaMatch_min = (1 - abs(ibmq_p_min - sample_data.min(axis=0)[1]) ) # Calculate Δp of min(P) Match.
        delta_p_min = abs(ibmq_p_min - sample_data.min(axis=0)[1]) # Calculate Δp of min(P).
        delta_p_max = abs(ibmq_p_max - float(df3Mem)) # Calculate Δp of max(P).

        Valid_Verdict = ['Weak',f'{{∞ , "/₀}}', 'Avg.', 'Strong', '№']  # Validation Verdict of a strong
                                                                        # vs. weak correlation match between
                                                                        # the selected file dataset p's and the IBM QDF
                                                                        # circuit event p's...
        # Note: A verdict of № denotes missing information or unknown about the max or min of the p's
        # (calculable if config is corrected or complemented).

        if round(deltaMatch_max, 2) >= 0 and round(deltaMatch_max, 2) < 0.5: # Classify/color code max Δp match.
            p_color[3] = Style.BRIGHT + Fore.RED
            Valid_Verdict = Fore.LIGHTRED_EX + Valid_Verdict[0]

        if round(deltaMatch_max, 2) >= 0.5 and round(deltaMatch_max, 2) <= 0.55: # Classify/color code max Δp match.
            p_color[3] = Style.NORMAL + fg.silver
            Valid_Verdict = Fore.LIGHTWHITE_EX + Valid_Verdict[1]

        if round(deltaMatch_min, 2) >= 0 and round(deltaMatch_min, 2) < 1/3: # Classify/color code min Δp match.
            p_color[4] = Style.BRIGHT + Fore.RED
            Valid_Verdict = Fore.LIGHTRED_EX + Valid_Verdict[0]

        if (round(deltaMatch_min, 2) <= 1/3 and
            round(deltaMatch_max, 2) <= 1/3) or (round(deltaMatch_min, 2) <= 1/2 and
                                                 round(deltaMatch_max, 2) <= 1/2): # Classify/color code min-max Δp match.
            p_color[3] = Style.NORMAL + fg.silver
            p_color[4] = p_color[3]
            Valid_Verdict = Fore.LIGHTWHITE_EX + Valid_Verdict[1]

        if round(deltaMatch_max, 2) > 0.55 and round(deltaMatch_max, 2) < 0.66: # Classify/color code max Δp match.
            p_color[3] = Style.BRIGHT + Fore.LIGHTYELLOW_EX
            Valid_Verdict = Fore.LIGHTYELLOW_EX + Valid_Verdict[2]

        if (round(deltaMatch_max, 2) >= 0.66 and
            round(deltaMatch_max, 2) < 0.9) and (round(deltaMatch_min, 2) >= 0.66 and
                                                 round(deltaMatch_max, 2) < 0.9): # Classify/color code min-max Δp match.
            p_color[3] = Style.BRIGHT + Fore.LIGHTMAGENTA_EX
            p_color[4] = p_color[3]
            Valid_Verdict = Fore.LIGHTCYAN_EX + 'Above ' + Valid_Verdict[2]

        if (round(deltaMatch_max, 2) >= 0.9 and
            round(deltaMatch_max, 2) <= 1) and (round(deltaMatch_min, 2) >= 0.9 and
                                                round(deltaMatch_max, 2) <= 1): # Classify/color code min-max Δp match.
            p_color[3] = Style.BRIGHT + Fore.LIGHTGREEN_EX
            p_color[4] = p_color[3]
            Valid_Verdict = Fore.LIGHTGREEN_EX + Valid_Verdict[3]

        if (round(deltaMatch_min, 2) <= 1/3 and
            round(deltaMatch_max, 2) <= 1/2) or (round(deltaMatch_min, 2) <= 1/2 and
                                                 round(deltaMatch_max, 2) <= 1/2): # Classify/color code min-max Δp match.
            p_color[3] = Style.NORMAL + fg.silver
            p_color[4] = p_color[3]
```

```
            Valid_Verdict = fg.silver + Valid_Verdict[4] # See note for a ₙᵤₗₗ verdict!

    PAnalysis_fig = tpl.figure() # Now plot histogram with horizontal bars for the computed probabilities of the two datasets.
    PAnalysis_fig.barh([P, float(df3Mem), float(dfcompMem), sample_data.min(axis=0)[1], ibmq_p_min, ibmq_p_max,
                        delta_p_min, delta_p_max, deltaMatch_min, deltaMatch_max],
        [Style.BRIGHT + Fore.LIGHTGREEN_EX + f"P(\x1B[4mTotal\x1B[0m{Style.BRIGHT + Fore.LIGHTGREEN_EX})",
         f'{Style.BRIGHT+Fore.WHITE}{{ {fileresult[filenum-1]} }} max(P(\x1B[4m{df2bin}\x1B[0m{Style.BRIGHT+Fore.WHITE}))',
         f'{Style.RESET_ALL + Fore.WHITE}{{ {fileresult[filenum-1]} }} comp(P(\x1B[4m{bitcomp}\x1B[0m{Style.DIM+Fore.WHITE}))',
         Style.BRIGHT + Fore.WHITE + f"{{ {fileresult[filenum-1]} }} min(P(\x1B[4m{df_min_bin}\x1B[0m{Style.BRIGHT + Fore.WHITE}))",
         f"{Style.BRIGHT +  Fore.YELLOW}{{ ibm qdf }} min((P(\x1B[4m{qdf_bit_pairs[0]}\x1B[0m{Style.BRIGHT + Fore.YELLOW})) ",
         f"{Style.BRIGHT +  Fore.YELLOW}{{ ibm qdf }} max((P(\x1B[4m{qdf_bit_pairs[1]}\x1B[0m{Style.BRIGHT + Fore.YELLOW})) ",
         p_color[4] + "Δp of \x1B[4mmin(P)\x1B[0m" + p_color[4],
         p_color[3] + f"Δp of \x1B[4mmax(P)\x1B[0m" + p_color[3],
         p_color[4]+ "Δp of\x1B[4m min(P) match\x1B[0m" + p_color[4],
         p_color[3]+ "Δp of\x1B[4m max(P) match\x1B[0m" + p_color[3]], force_ascii=False), sleep(1)
    PAnalysis_fig.show()
    bar(1.)

    print(f"{Fore.LIGHTYELLOW_EX+hline+ Fore.LIGHTMAGENTA_EX}\n*- Δp Data: ☣  \033[4mValidation Verdict\033[0m ☣{fg.yellow}  \
    between the two dataset samples from {fg.cyan}{{ {fileresult[filenum-1]} , {ibmq_result} }}{fg.yellow} \
    \n   is a/an: \
    \033[4m{Valid_Verdict + ' Match {{ '+ str(round(deltaMatch_min, 2)) +' , '+ str(round(deltaMatch_max, 2)) +' }}'}\033[0m\
    {Fore.LIGHTYELLOW_EX}")

    DeltaP_verdict = Valid_Verdict
    Valid_Verdict = ['Weak',f'{{∞ , ⁿ/ₐ}}', 'Avg.', 'Strong', 'ₙᵤₗₗ']  # Reset Valid_Verdict list elements upon
                                                                      # the previously rendered verdict.
    #-------------------------------------------------------------------------------------
    # Validate circuit config. based on recorded P's associated to their qubit pair binaries (strings)
    s1 = df_min_bin
    s2 = qdf_bit_pairs[0]
    s3 = df2bin
    s4 = qdf_bit_pairs[1]
    vv_circuits =[False]
    vv_bins = [False]

    # Validation Verdict on the two circuits qubit string sets over min and max P's as compared for a match.
    if re.match(s2, s1):
        vv_bins = fg.lightgreen+ Valid_Verdict[3]
        qdf_s_match = f"{{ {s1} , {s2} }}"
    elif re.match(s4, s3):
        vv_bins = fg.lightgreen + Valid_Verdict[3]
        qdf_s_match = f"{{ {s3} , {s4} }}"
    else:
        vv_bins = fg.red + Valid_Verdict[4] # See note for a ₙᵤₗₗ verdict!

    # Validation Verdict on the two circuits as compared for a config. match.
    if (vv_bins == fg.lightgreen + Valid_Verdict[3]) and (DeltaP_verdict == Fore.LIGHTGREEN_EX + Valid_Verdict[3]):
        vv_circuits = fg.green + Valid_Verdict[3]
    if (vv_bins == fg.lightgreen + Valid_Verdict[3]) and (DeltaP_verdict == Fore.LIGHTGREEN_EX + Valid_Verdict[1]):
        vv_circuits = fg.yellow + Valid_Verdict[2]
    if (vv_bins == fg.lightgreen + Valid_Verdict[3]) and (DeltaP_verdict == Fore.LIGHTCYAN_EX + 'Above ' + Valid_Verdict[2]):
        vv_circuits = Fore.LIGHTCYAN_EX + 'Above ' + Valid_Verdict[2]
    if (vv_bins == fg.red + Valid_Verdict[4]) and (DeltaP_verdict == Fore.LIGHTGREEN_EX + Valid_Verdict[3]):
        vv_circuits = fg.yellow + Valid_Verdict[2]
    if ((vv_bins == fg.red +
         Valid_Verdict[4]) or (vv_bins == fg.lightgreen + Valid_Verdict[3])) and (DeltaP_verdict == fg.silver + Valid_Verdict[4]):
         vv_circuits = f'{{ {fg.lightred + Valid_Verdict[4]} , {False} }}' # A ₙᵤₗₗ or false match verdict! See also next line/condition.
    if (float(df3Mem) <= 1/3 and sample_data.min(axis=0)[1]<= 1/3) and (ibmq_p_max >= 1/2):
        vv_circuits = f'{fg.lightred}{{ {Valid_Verdict[4]} , {fg.yellow}{Valid_Verdict[1]}{fg.lightred} }}' # A ₙᵤₗₗ match verdict! In short,
         # a state transition matrix with all elements between certain circuit components return a 0 state or null match and
         # state 2 for one of the two circuits! See P's preliminary analysis for the measured QDF circuit.
         # This circuit event outcome can also be interpreted as a rejected match between the two circuits, based on their configuration.
    if (vv_bins == fg.red + Valid_Verdict[4]) and (Valid_Verdict == Fore.LIGHTYELLOW_EX + Valid_Verdict[2]):
        vv_circuits = fg.red + Valid_Verdict[2]
    #-------------------------------------------------------------------------------------
    print(f"\033[1m{Fore.LIGHTYELLOW_EX+hline+ Fore.LIGHTMAGENTA_EX}\n*- QDF Qubit Sets: ☣  \033[4mValidation Verdict\033[0m ☣{fg.yellow}  \
    between the two dataset samples from {fg.cyan}{{ {fileresult[filenum-1]} , {ibmq_result} }}{fg.yellow} \
    \n   is a/an: \033[4m{vv_bins + ' Match ' + qdf_s_match}\033[0m {Fore.LIGHTYELLOW_EX}")

    print(f"\033[1m{Fore.LIGHTYELLOW_EX+hline+ Fore.LIGHTMAGENTA_EX}\n*- QDF Circuits: ☣  \033[4mValidation Verdict\033[0m ☣{fg.yellow}  \
    between the two QDF circuit configurations from {fg.cyan}{{ {fileresult[filenum-1]} , {ibmq_result} }}{fg.yellow} \
    \n   is a/an: \033[4m{vv_circuits} Match \033[0m {Fore.LIGHTYELLOW_EX}")

    # Log validation verdict results on simulation and dataset analysis.
    entry_stage = 2
    sim_state[0] = f"Δp Data: {Valid_Verdict}, QDF Qubit Sets: {vv_bins + ' Match ' + qdf_s_match},  QDF Circuits: {vv_circuits} Match"
    sim_log()

    print(f"{hline + Fore.LIGHTMAGENTA_EX}\n *- The IBM QDF circuit can be reconfigured for worst and best case Hamiltonian scenarios,\
    \n    given the expected QDF measurement outcomes from the collected QFLCA datasets for a QFLCC.\
    \n\x1b[38;5;226m Simulation completed on: \033[1;32m{fg.yellow+bg.blue+str(datetime.datetime.now())+Back.RESET} \
    \n{hline + Fore.RESET}"), sleep(4)

    dir_flag = 0
    print(Back.LIGHTGREEN_EX + Fore.YELLOW + "\033[1m<--- QDF CIRCUITS SIMULATION_DATASET_ANALYSIS CONCLUDED --->\033[0m"
          + Fore.LIGHTGREEN_EX + Back.RESET)

    # Log the end of simulation and dataset analysis.
    entry_stage = 3
    sim_state[0] = " QDF CIRCUITS SIMULATION_DATASET_ANALYSIS CONCLUDED "
    sim_log()
```

▼ 🔽 sim_log( ) function code:

ℹ️ To log simulation events of the IBM QDF circuit and the selected dataset by the user for analysis.

- ▼ 🔽 Source code in `QAI-LCode_QFLCC.py`

```
def sim_log():
 ########################################
 # To log simulation checkpoints/steps
 ########################################
 global entry_stage

 entry_stage += 1
 idxplus = len(res) + entry_stage
 subprocess.run("echo {}- Checkpoint logged on {} for file # {}\
 as {} parallel to simulation run file {{ QDF-LCode_IBMQ-2024-codable }}, \
 Simulation State ///--{}--///".format(idxplus, now.strftime("%Y-%m-%d %H:%M:%S"),
                                       idx, fileresult[idx-1], sim_state), shell=True, stdout=file_)
 """End of simulation log."""
```

▼ 🔽 ℹ️ Expand/Collapse All...

* Click ▼Expand All to view all code blocks from QAI-LCode_QFLCC.py on this page.
* Click ▶Collapse All to selectively view a code block from QAI-LCode_QFLCC.py on this page.

▼Expand All ▶Collapse All

Documentation built with MkDocs.