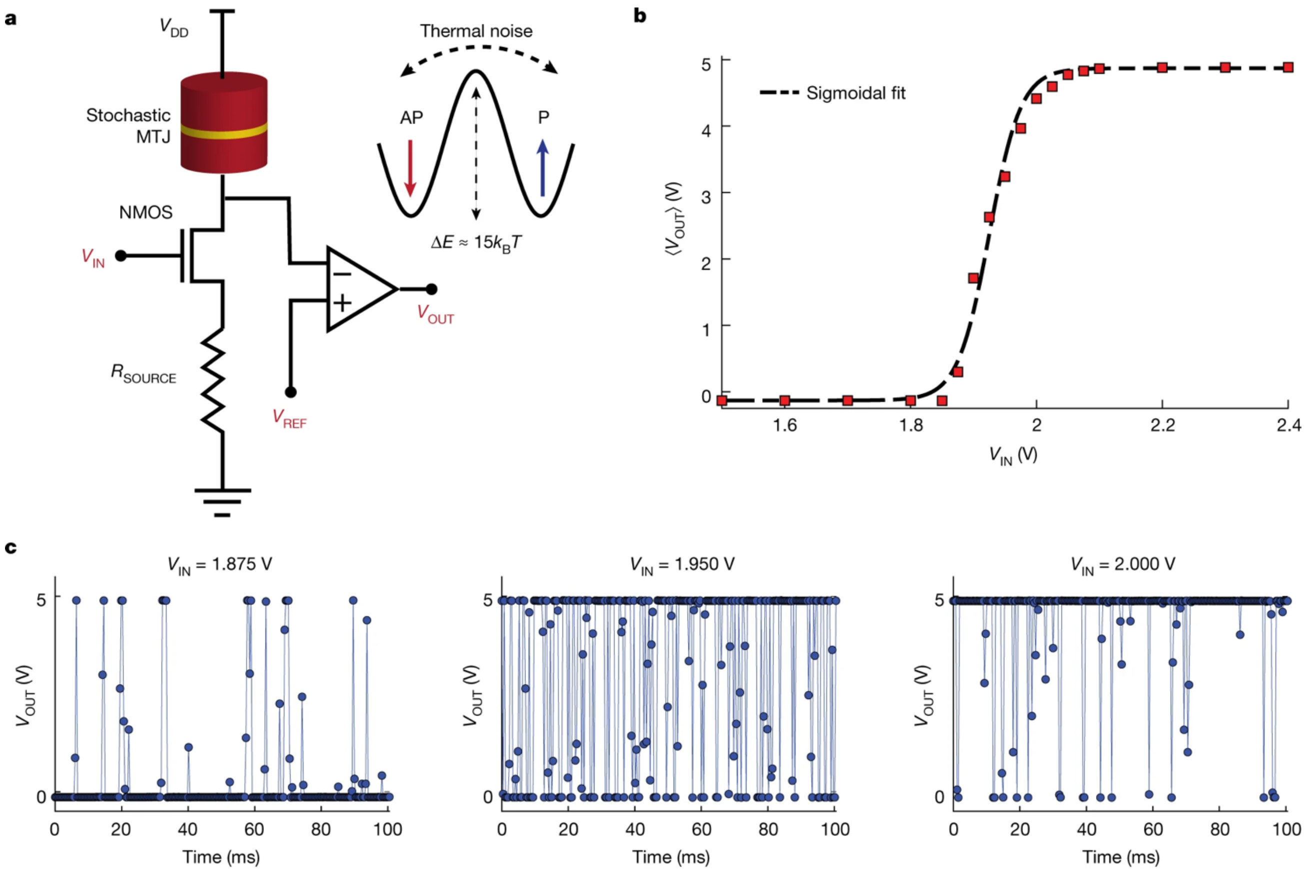
SNSPD Study Summary Report

This study focuses on the use of SNSPDs for probabilistic computing (P-Bits). This idea is inspired by stochastic magnetic tunnel junctions (MTJs) for probabilistic computing.

1. Stochastic MTJ for Probabilistic Computing

The MTJ is characterised by its tunnelling magnetoresistance, which can be switched between high and low values by varying the angle between the magnetisation direction of the two ferromagnetic layers. The stochastic MTJ (S-MTJ) is an unstable MTJ with a free layer (broken) that has a relatively low energy barrier, allowing thermal noise to make it fluctuate between its stable states, one being parallel (P, low resistance) to the fixed layer and the other being anti-parallel (AP, high resistance). The input voltage can affect the stability of the free layer, thereby changing its resistance to vary the output voltage. This property can be used to build the stochastic neural networks for probabilistic computing. To form this, the stochastic MTJs are connected with standard n-type metal–oxide–semiconductor (NMOS) transistors to obtain a three-terminal p-bit.



The output voltage for the p-bit, $V_{OUT,i}$ from this composite unit can be written in terms of the input voltage $V_{IN,i}$ in a form similar to the binary stochastic neuron described above:

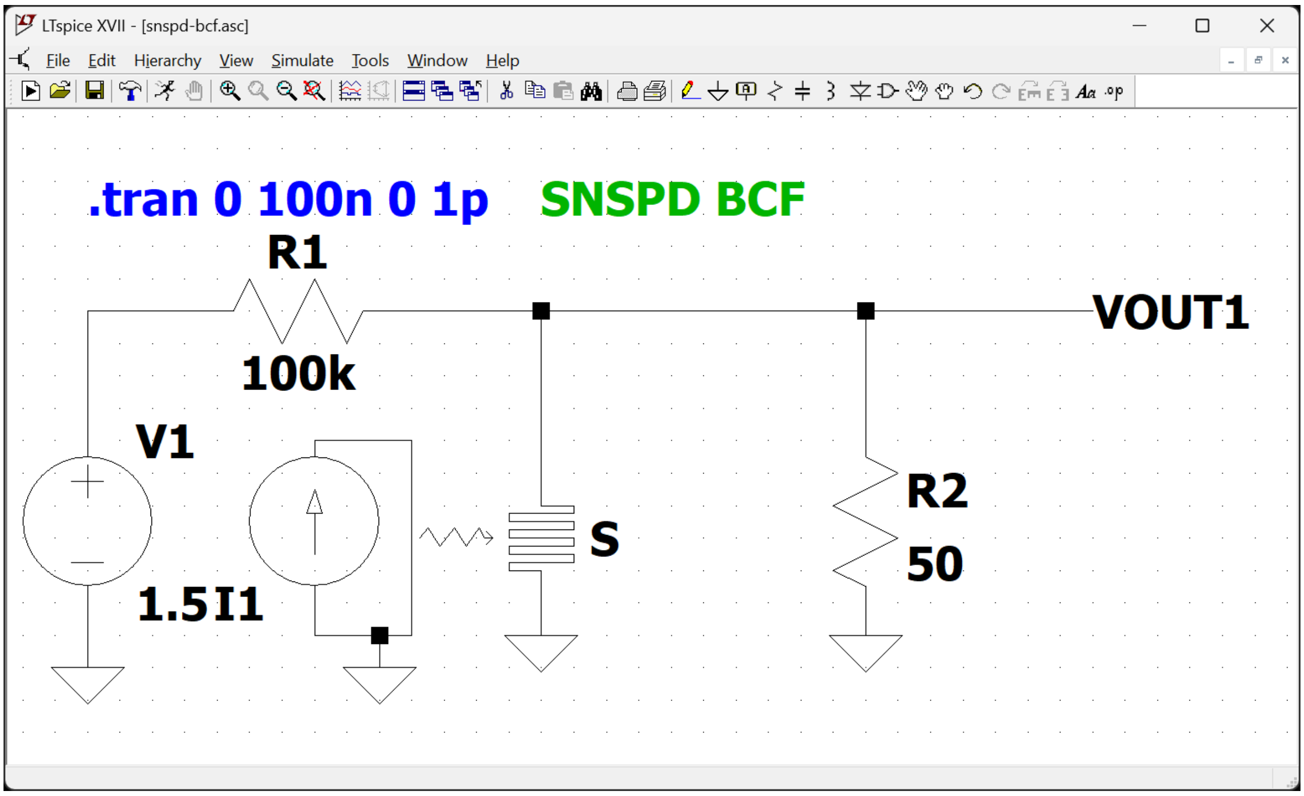
$$\frac{V_{OUT,i}}{V_{DD}} \approx \vartheta\left(\sigma\left\{\frac{V_{IN,i} - v_{0,i}}{V_{o,i}}\right\} - r\right)$$

where V_{DD} is the supply voltage, $V_{0,i}$ is the scaling voltage determined by the transistor, and $v_{0,i}$ is the offset voltage.

2. Superconducting Nanowire Single-Photon Detectors (SNSPD)

The nanowire is cooled well below its superconducting critical temperature and biased with a **DC current** that is close to but less than the superconducting critical current of the nanowire. When a photon is detected (Cooper pairs are broken), a localised non-superconducting region (hotspot) with finite electrical resistance is formed. This produces a measurable voltage pulse that is approximately equal to the bias current multiplied by the load resistance. Then most of the bias current flowing through the load resistance, the hotspot cools and returns to the superconducting state. The time for the current to return to the nanowire is typically set by the inductive time constant (kinetic inductance of the nanowire divided by the impedance of the readout circuit) of the nanowire.

To investigate the potential of SNSPDs for probabilistic computing, simulating the electrical properties is necessary. Based on this, LTspice is used to study the electrical properties of SNSPDs. Then, by adding extra circuitry in LTspice and using Python for data processing and LTspice interaction, it is possible to achieve a phenomenon similar to that of stochastic MTJs.


LTspice XVII - [snspd-bcf.asc]
File Edit Hierarchy View Simulate Tools Window Help
R1
100k
VOUT1
V1
1.5I1
S
R2
50

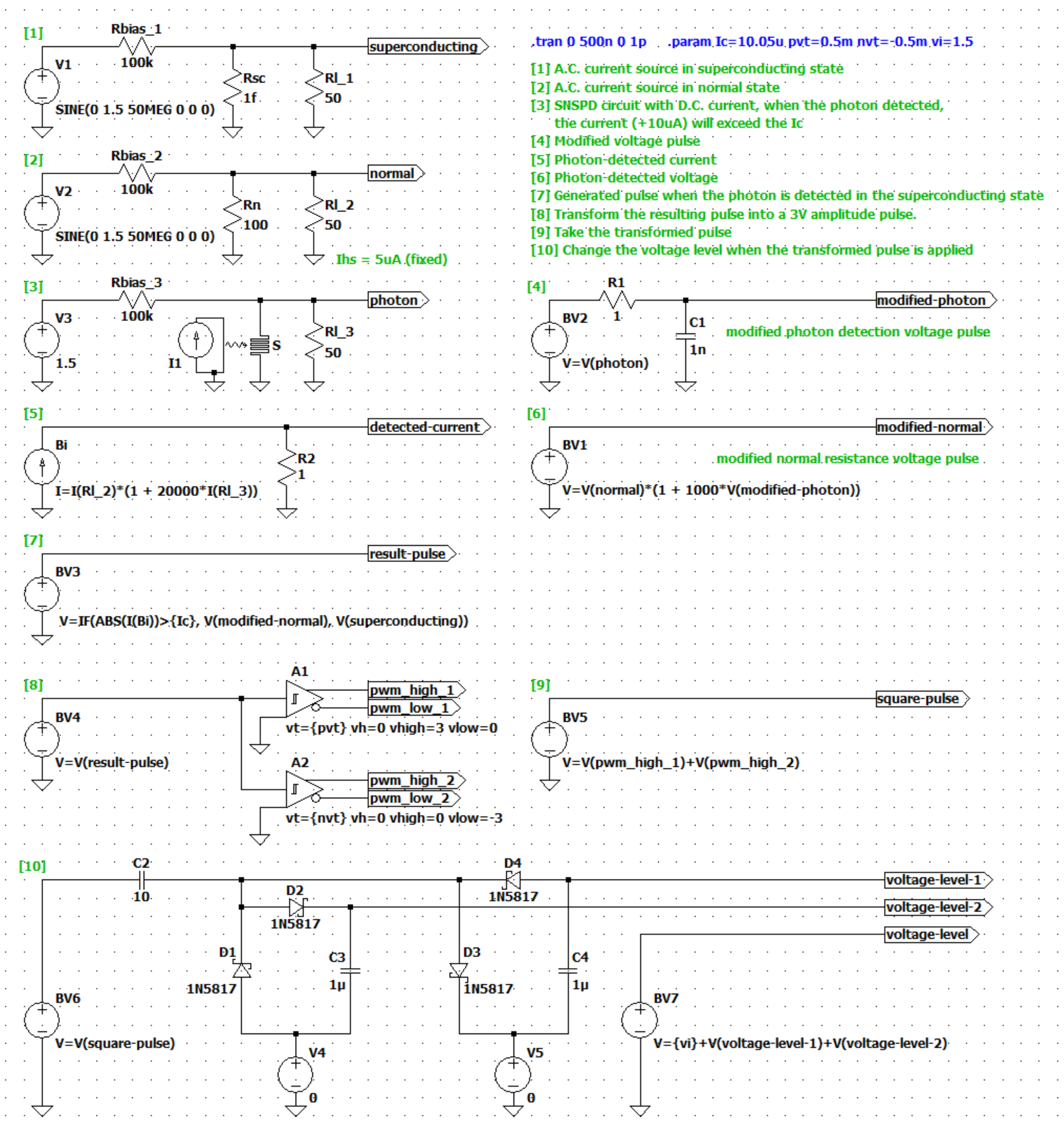
```
.tran 0 100n 0 1p
```

SNSPD BCF

The LTspice model of SNSPD, the circuit is supplied with a D.C. current source. When a photon is detected, the arbitrary behavioural current source will generate a current pulse of 1 μA (modifiable), which will be amplifier ($\times 10$) and will trigger the switch in the model S to generate a resistance which is higher than the load resistance. This will lead to a voltage pulse being generated in the circuit. As the current pulse decreases, the switch will turn off to achieve superconductivity again.

SNSPD LTspice Model for Probabilistic Computing

LTspice model

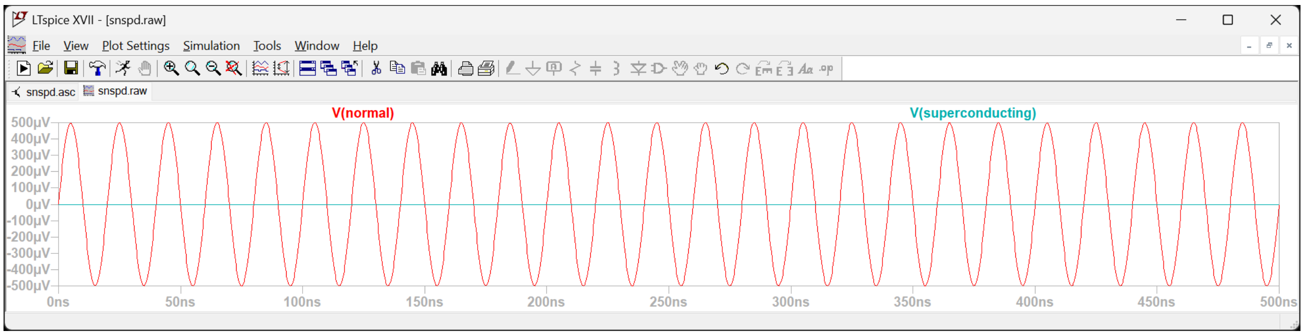


```
.tran 0 500n 0 1p    .param Ic=10.05u pvt=0.5m nvt=-0.5m vi=1.5
```

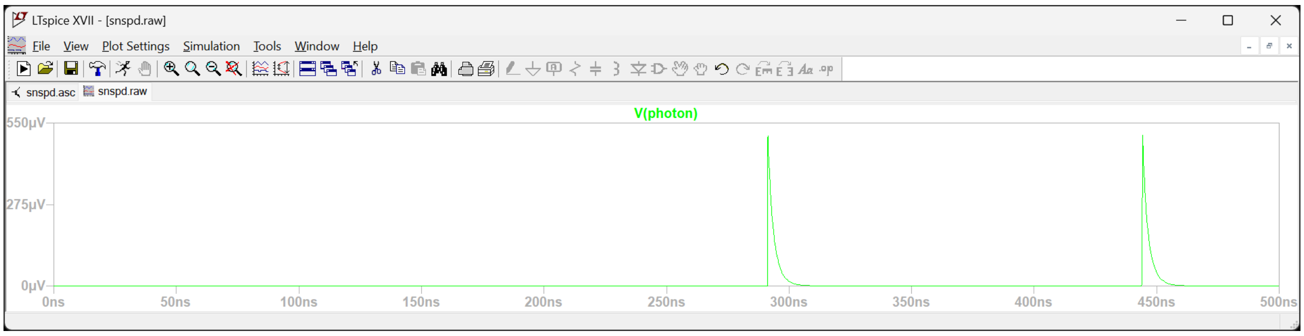
[1] A.C. current source in superconducting state
[2] A.C. current source in normal state
[3] SNSPD circuit with D.C. current, when the photon detected, the current (+10uA) will exceed the Ic
[4] Modified voltage pulse
[5] Photon-detected current
[6] Photon-detected voltage
[7] Generated pulse when the photon is detected in the superconducting state
[8] Transform the resulting pulse into a 3V amplitude pulse.
[9] Take the transformed pulse
[10] Change the voltage level when the transformed pulse is applied

Since the A.C. current source supplied to an SNSPD leads to more noise and makes the detection process more complex (e.g., Joule heating and hysteresis), the SNSPD model cannot be supplied with an A.C. current source directly. Therefore, to achieve an SNSPD detection process with an A.C. current source, the SNSPD is detected with a D.C. current source and is modified in an A.C. current source.

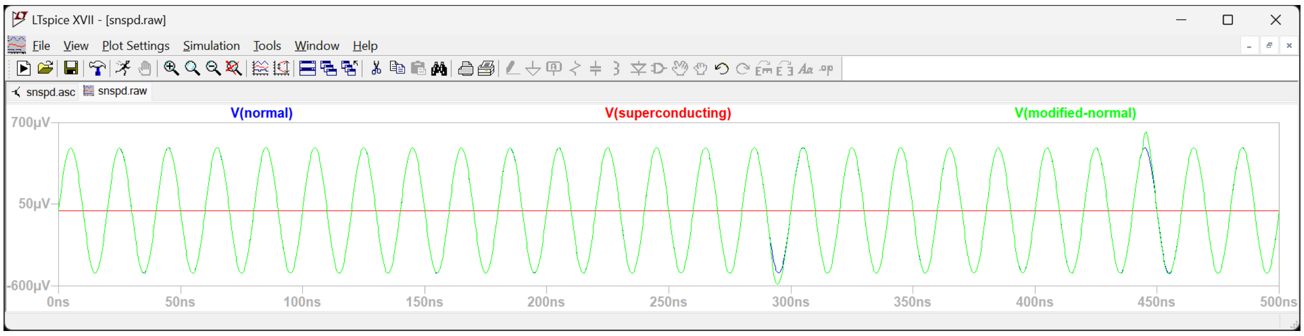
Circuits 1 and 2 show the voltage output in the superconducting state and the normal state, respectively (A.C. voltage 1.5 V).



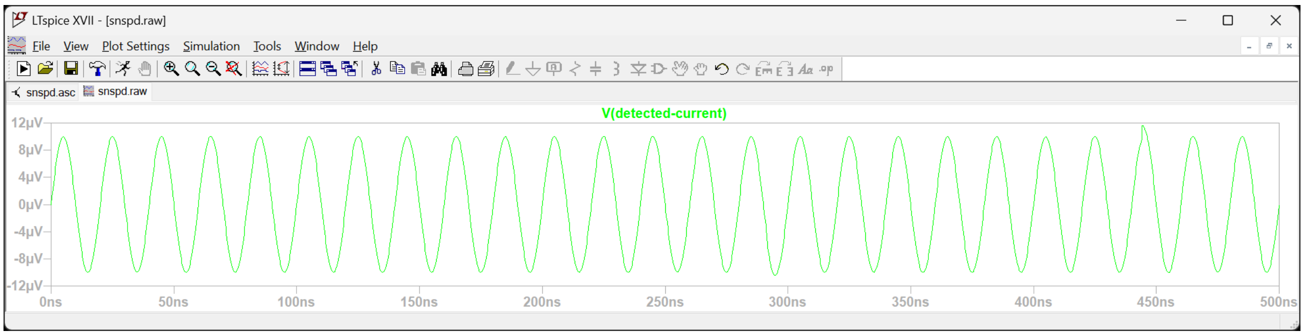
Circuit 3 is the SNSPD model circuit, which is used to generate pulse when a photon is detected.



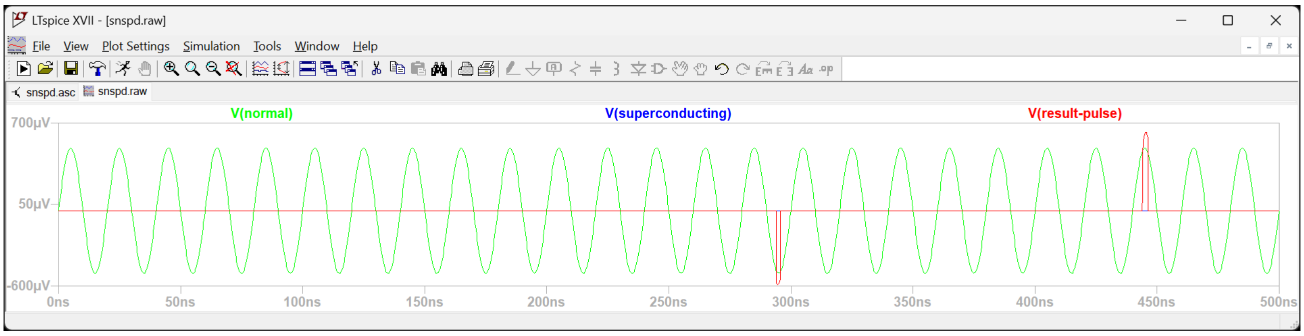
Circuit 5 is an arbitrary behavioural current source that shows the current change when a photon is detected, used for **checking the state change**.



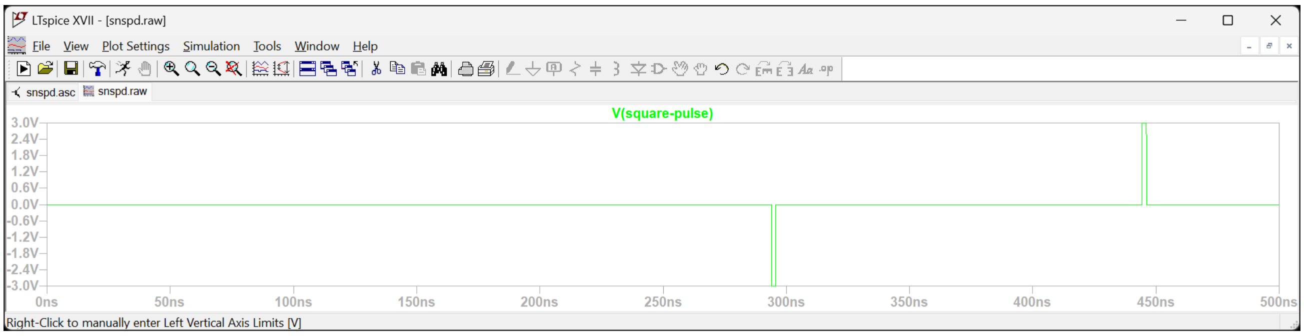
Circuit 6 is an arbitrary behavioural voltage source that shows the voltage change when a photon is detected.



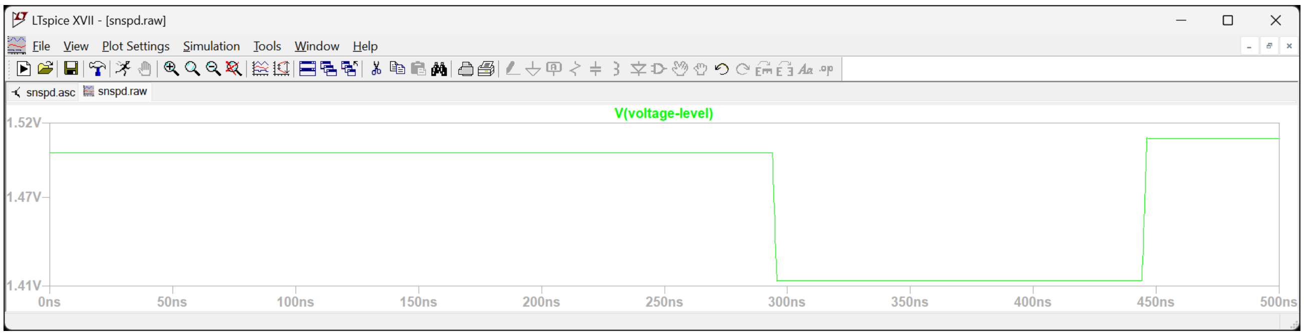
Circuit 7 illustrates the output voltage pulse (in red) after photon detection.



Circuit 8 is a circuit that includes two comparators to transform both positive and negative voltages into a 3V amplitude output voltage. Circuit 9 is used to combine these two types of voltage (positive and negative) into one voltage signal.


LTspice XVII - [snspd.raw]
File View Plot Settings Simulation Tools Window Help
snspd.asc snspd.raw
V(square-pulse)
3.0V
2.4V
1.8V
1.2V
0.6V
0.0V
-0.6V
-1.2V
-1.8V
-2.4V
-3.0V
0ns
50ns
100ns
150ns
200ns
250ns
300ns
350ns
400ns
450ns
500ns
Right-Click to manually enter Left Vertical Axis Limits [V]

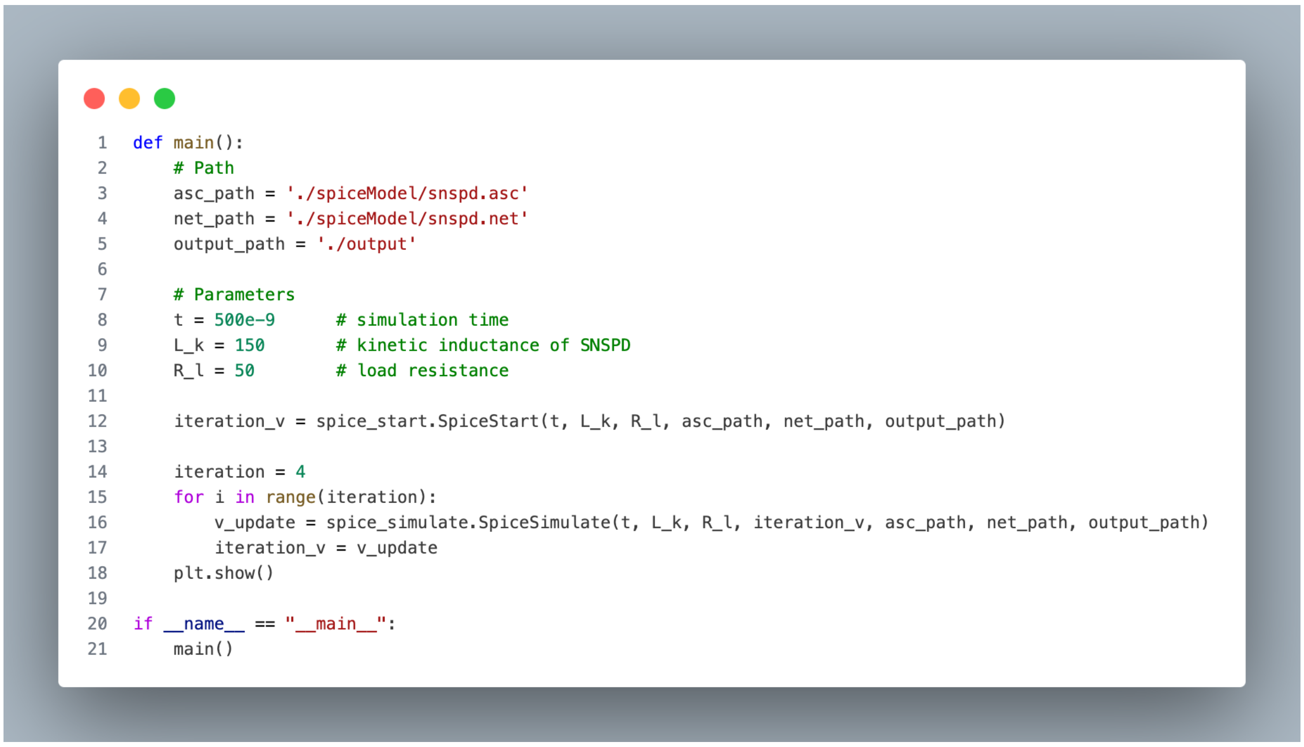
Circuit 10 takes the 3V square pulse as the input. The output pulse will change the voltage according to the direction and number of the 3V pulses. For example, if the initial voltage is 1.5V, photon detection will generate a pulse. This pulse will transform into a 3V square pulse and change the value of the initial voltage.



In summary, this model can simulate the SNSPD photon detection pulse and apply it in an A.C. source. It then transforms the pulse into a higher voltage square pulse (e.g., 3V). These square pulses can change the output voltage level, similar to the operation of a moiré transistor. This output voltage level is then used as the new input voltage source, which affects the SNSPD detection efficiency. This process is similar to the stochastic MTJ used in probabilistic computing.

Python code

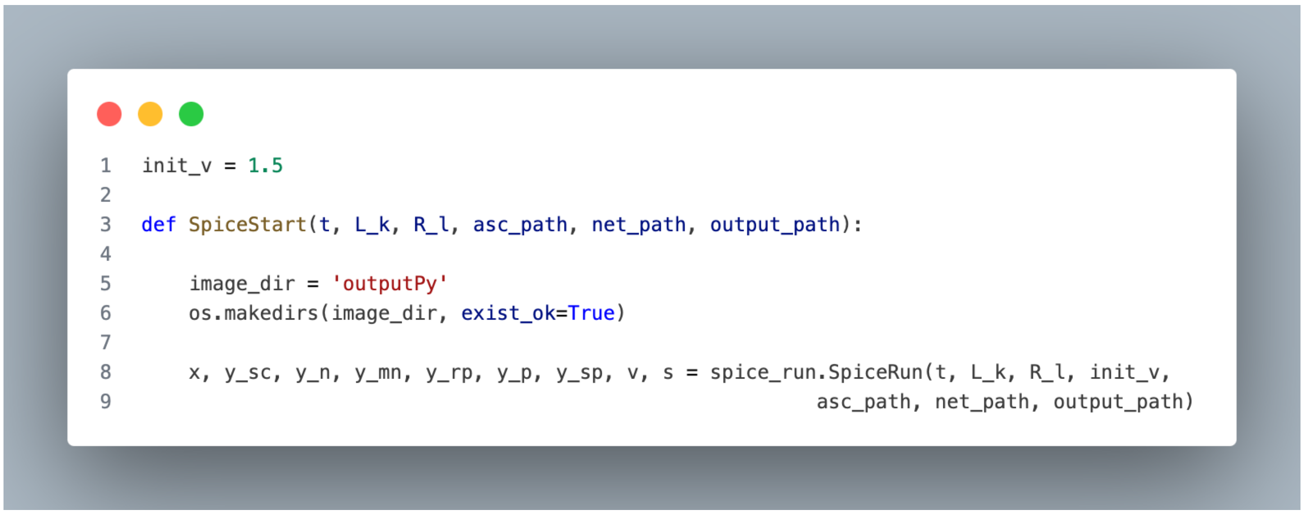
`main.py`



```python
def main():
    # Path
    asc_path = './spiceModel/snspd.asc'
    net_path = './spiceModel/snspd.net'
    output_path = './output'

    # Parameters
    t = 500e-9      # simulation time
    L_k = 150       # kinetic inductance of SNSPD
    R_l = 50        # load resistance

    iteration_v = spice_start.SpiceStart(t, L_k, R_l, asc_path, net_path, output_path)

    iteration = 4
    for i in range(iteration):
        v_update = spice_simulate.SpiceSimulate(t, L_k, R_l, iteration_v, asc_path, net_path, output_path)
        iteration_v = v_update
    plt.show()

if __name__ == "__main__":
    main()
```

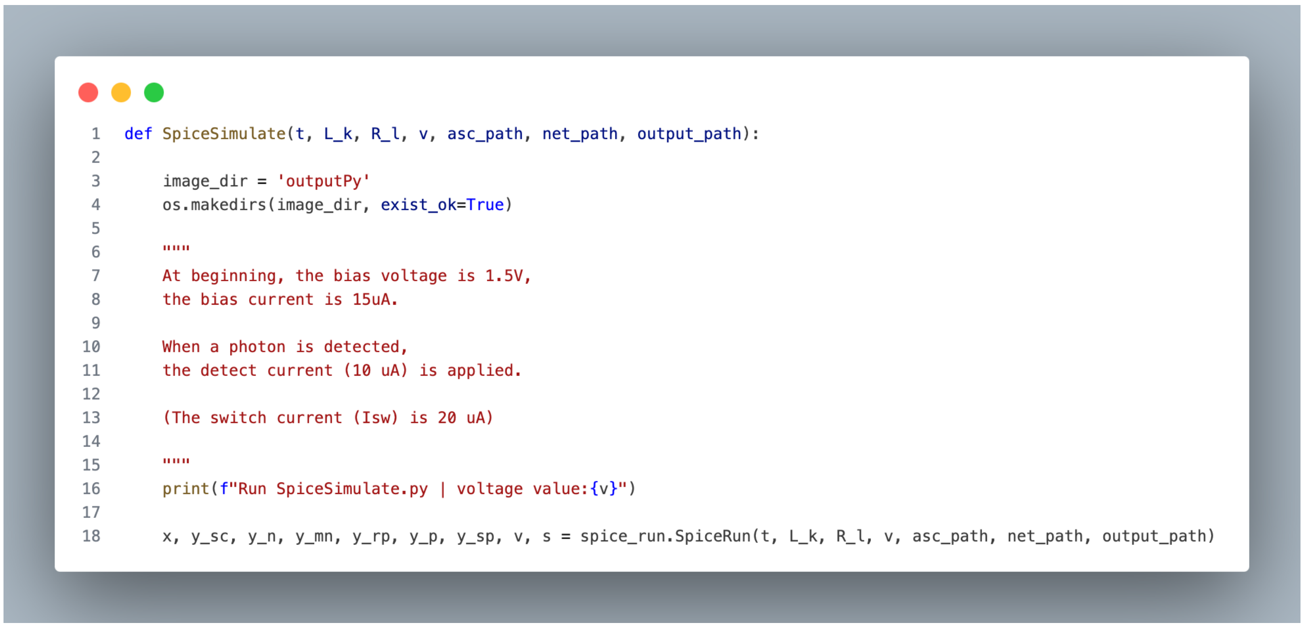
The variables (time simulation, kinetic inductance and load resistance) are set. The `SpiceStart` function initiates the first simulation and provides the output voltage level. The variable `iteration` indicates the number of further simulations to run. The `SpiceSimulate` function is then executed for each subsequent simulation, using the previous output voltage as the input for the next simulation.

`SpiceStart.py`



```python
init_v = 1.5

def SpiceStart(t, L_k, R_l, asc_path, net_path, output_path):

    image_dir = 'outputPy'
    os.makedirs(image_dir, exist_ok=True)

    x, y_sc, y_n, y_mn, y_rp, y_p, y_sp, v, s = spice_run.SpiceRun(t, L_k, R_l, init_v,
                                                                   asc_path, net_path, output_path)
```

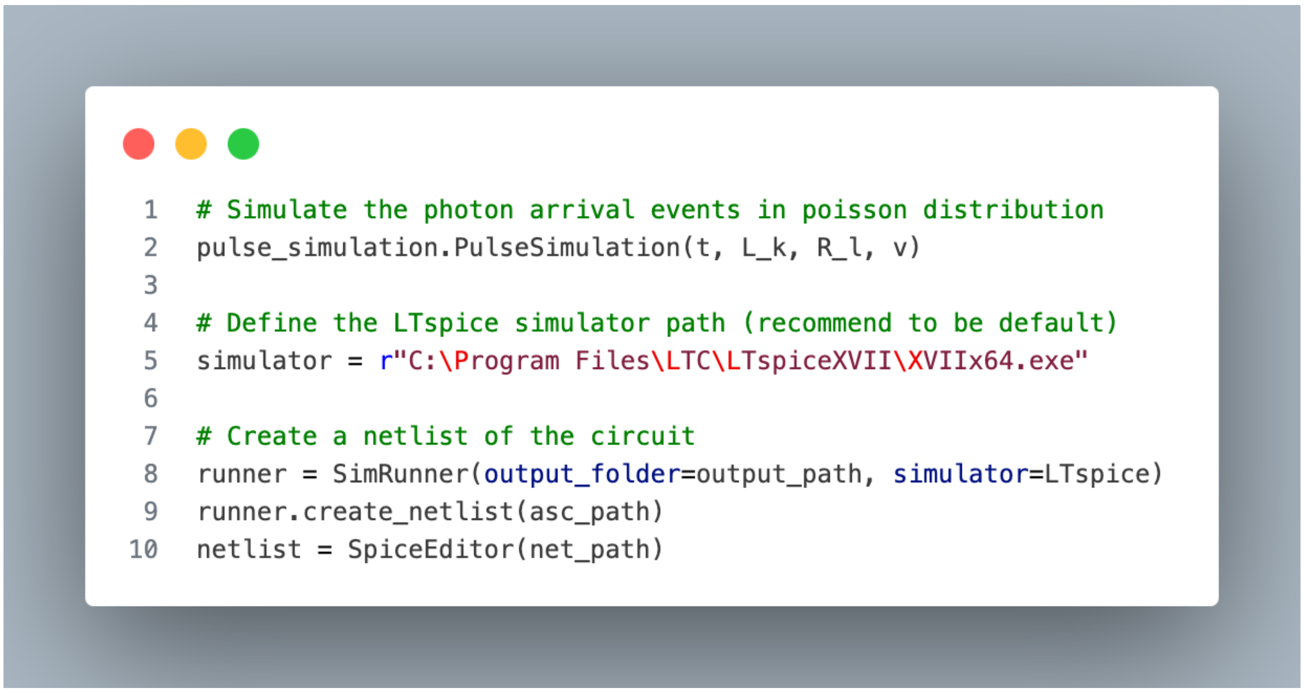
`SpiceSimulate.py`



```python
def SpiceSimulate(t, L_k, R_l, v, asc_path, net_path, output_path):

    image_dir = 'outputPy'
    os.makedirs(image_dir, exist_ok=True)

    """
    At beginning, the bias voltage is 1.5V,
    the bias current is 15uA.

    When a photon is detected,
    the detect current (10 uA) is applied.

    (The switch current (Isw) is 20 uA)

    """
    print(f"Run SpiceSimulate.py | voltage value:{v}")

    x, y_sc, y_n, y_mn, y_rp, y_p, y_sp, v, s = spice_run.SpiceRun(t, L_k, R_l, v, asc_path, net_path, output_path)
```

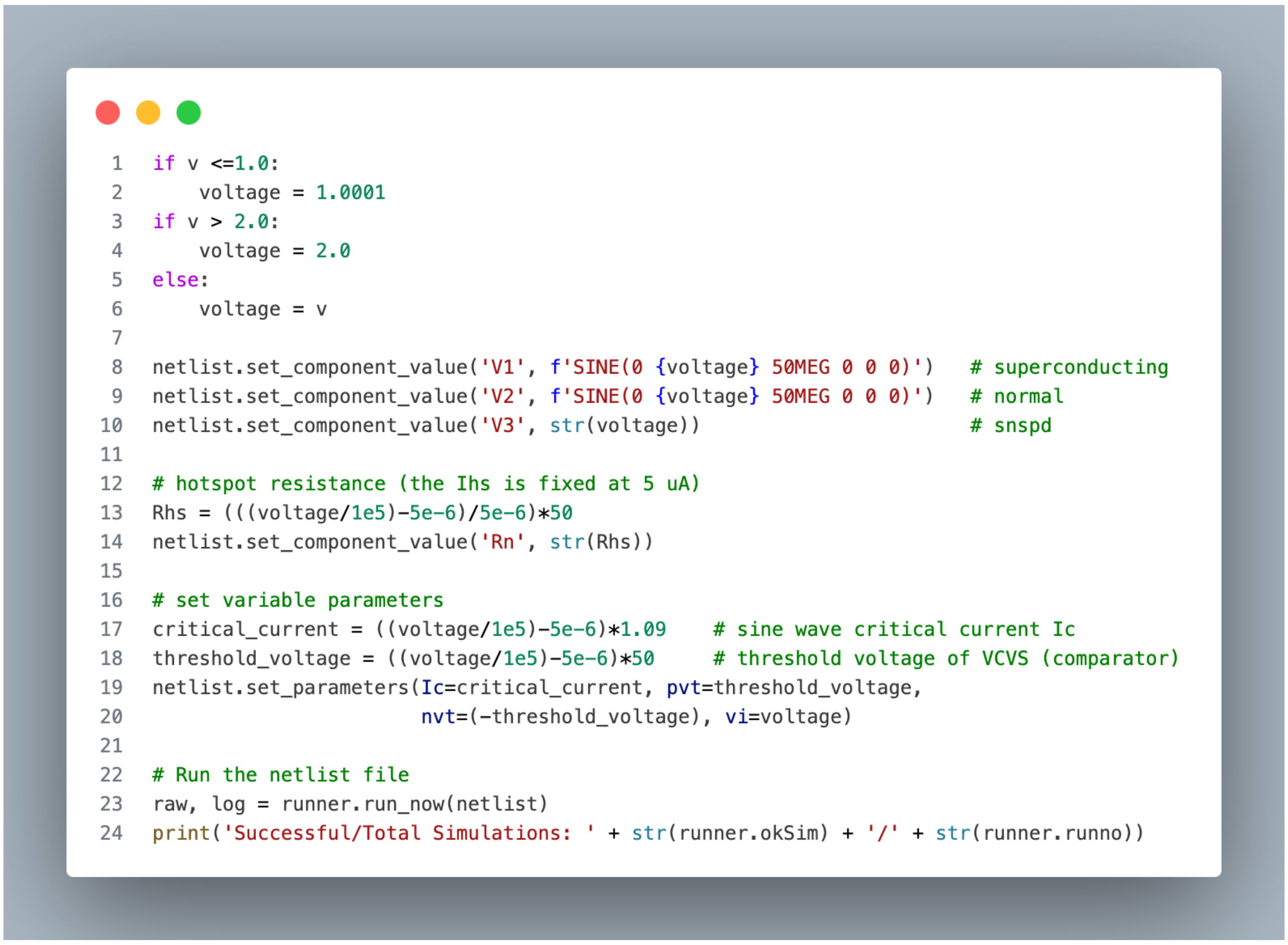
Both the `SpiceStart.py` file and `SpiceSimulate.py` file call the `SpiceRun` function to interact with LTspice and generate the necessary data, including time, superconducting state voltage, normal state voltage, result pulse, and output voltage.

`SpiceRun.py`



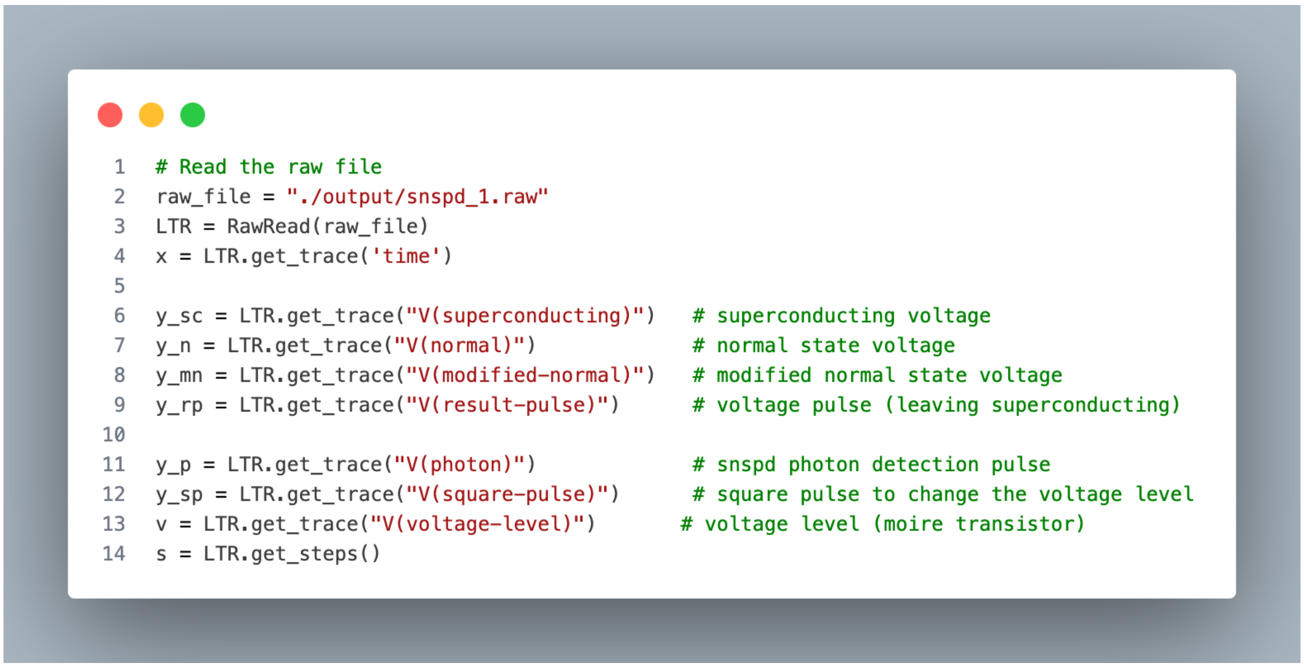
```python
# Simulate the photon arrival events in poisson distribution
pulse_simulation.PulseSimulation(t, L_k, R_l, v)

# Define the LTspice simulator path (recommend to be default)
simulator = r"C:\Program Files\LTC\LTspiceXVII\XVIIx64.exe"

# Create a netlist of the circuit
runner = SimRunner(output_folder=output_path, simulator=LTspice)
runner.create_netlist(asc_path)
netlist = SpiceEditor(net_path)
```

The code calls the `PulseSimulation` function to generate the pulse time points (details will be explained later). It also defines the LTspice simulator and converts the ASCII circuit (.asc file) into a netlist (.net file) for interaction.



```python
if v <=1.0:
    voltage = 1.0001
if v > 2.0:
    voltage = 2.0
else:
    voltage = v

netlist.set_component_value('V1', f'SINE(0 {voltage} 50MEG 0 0 0)')   # superconducting
netlist.set_component_value('V2', f'SINE(0 {voltage} 50MEG 0 0 0)')   # normal
netlist.set_component_value('V3', str(voltage))                        # snspd

# hotspot resistance (the Ihs is fixed at 5 uA)
Rhs = (((voltage/1e5)-5e-6)/5e-6)*50
netlist.set_component_value('Rn', str(Rhs))

# set variable parameters
critical_current = ((voltage/1e5)-5e-6)*1.09    # sine wave critical current Ic
threshold_voltage = ((voltage/1e5)-5e-6)*50     # threshold voltage of VCVS (comparator)
netlist.set_parameters(Ic=critical_current, pvt=threshold_voltage,
                       nvt=(-threshold_voltage), vi=voltage)

# Run the netlist file
raw, log = runner.run_now(netlist)
print('Successful/Total Simulations: ' + str(runner.okSim) + '/' + str(runner.runno))
```

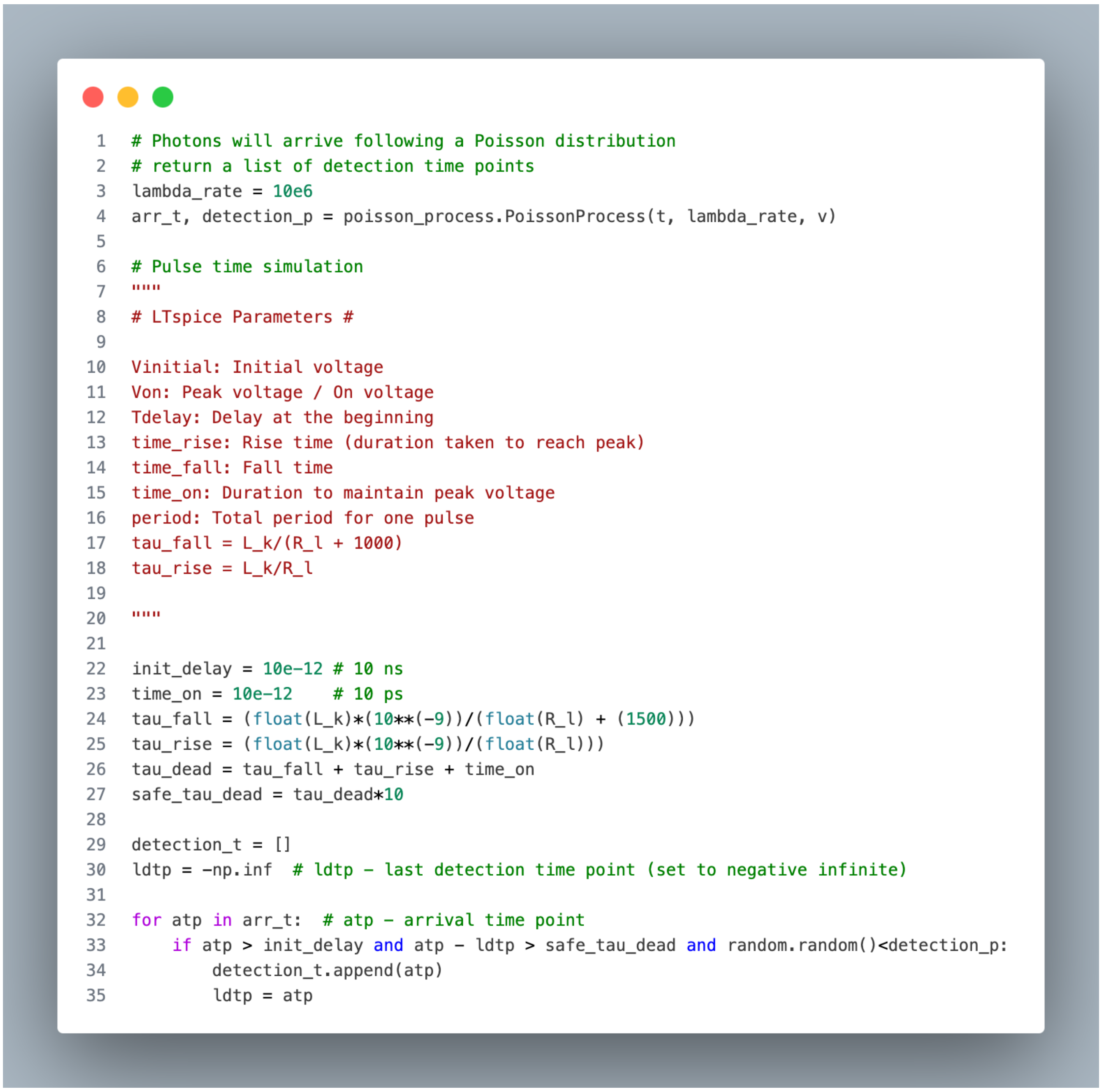
Then the voltage is first checked to ensure it is within the working range of 1.0 V to 2.0 V, to avoid errors. Then, functions from the PyLTspice library are used to change the voltage in the LTspice circuit and adjust all necessary elements before running the netlist file.



```python
# Read the raw file
raw_file = "./output/snspd_1.raw"
LTR = RawRead(raw_file)
x = LTR.get_trace('time')

y_sc = LTR.get_trace("V(superconducting)")   # superconducting voltage
y_n = LTR.get_trace("V(normal)")             # normal state voltage
y_mn = LTR.get_trace("V(modified-normal)")   # modified normal state voltage
y_rp = LTR.get_trace("V(result-pulse)")      # voltage pulse (leaving superconducting)

y_p = LTR.get_trace("V(photon)")             # snspd photon detection pulse
y_sp = LTR.get_trace("V(square-pulse)")      # square pulse to change the voltage level
v = LTR.get_trace("V(voltage-level)")       # voltage level (moire transistor)
s = LTR.get_steps()
```

After changing the elements and running the simulation, read the outputs for further simulation.

`PulseSimulation.py`



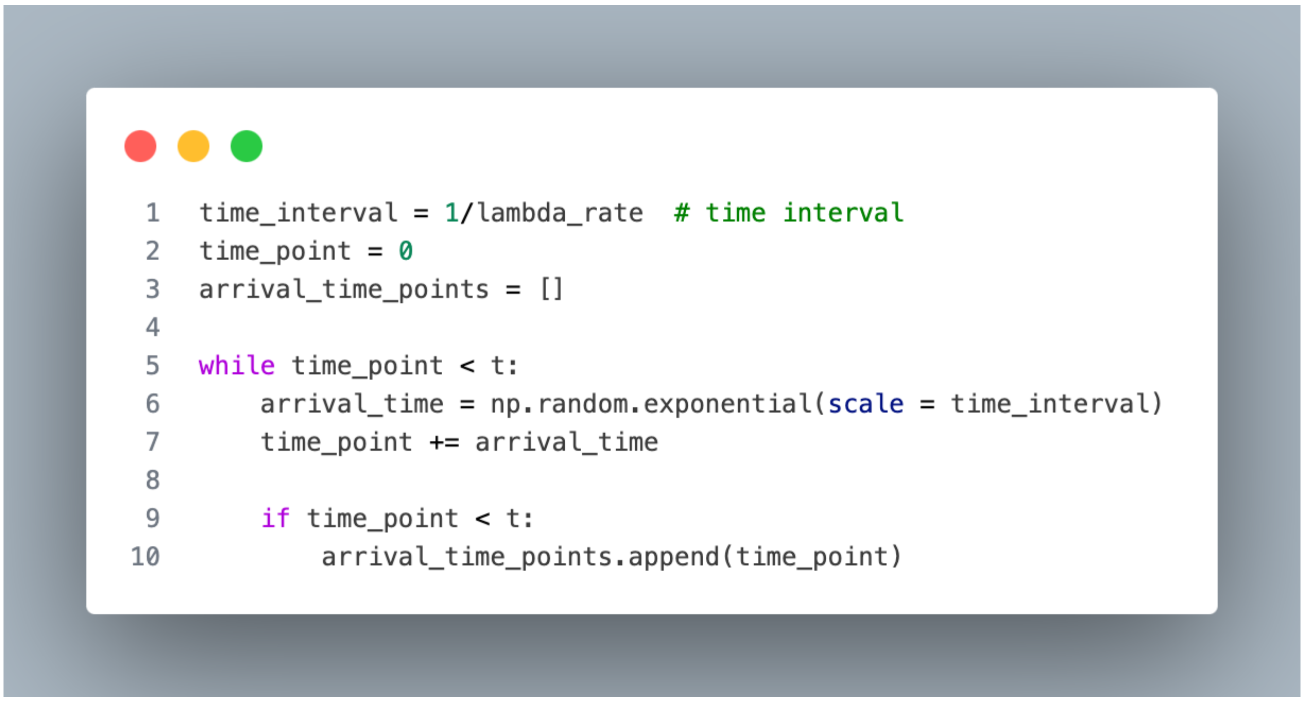
```python
# Photons will arrive following a Poisson distribution
# return a list of detection time points
lambda_rate = 10e6
arr_t, detection_p = poisson_process.PoissonProcess(t, lambda_rate, v)

# Pulse time simulation
"""
# LTspice Parameters #

Vinitial: Initial voltage
Von: Peak voltage / On voltage
Tdelay: Delay at the beginning
time_rise: Rise time (duration taken to reach peak)
time_fall: Fall time
time_on: Duration to maintain peak voltage
period: Total period for one pulse
tau_fall = L_k/(R_l + 1000)
tau_rise = L_k/R_l

"""

init_delay = 10e-12 # 10 ns
time_on = 10e-12    # 10 ps
tau_fall = (float(L_k)*(10**(-9))/(float(R_l) + (1500)))
tau_rise = (float(L_k)*(10**(-9))/(float(R_l)))
tau_dead = tau_fall + tau_rise + time_on
safe_tau_dead = tau_dead*10

detection_t = []
ldtp = -np.inf  # ldtp - last detection time point (set to negative infinite)

for atp in arr_t:  # atp - arrival time point
    if atp > init_delay and atp - ldtp > safe_tau_dead and random.random()<detection_p:
        detection_t.append(atp)
        ldtp = atp
```

This function is used to simulate the pulse time points, ensuring no pulse is shown during the SNSPD recovery time, and to define the pulse shape (affected by the kinetic inductance and load resistance).

In this function, it calls the `PoissonProcess` function to get the photon arrival time points and the detection probability of the SNSPD (affected by the voltage level). It then calculates the rise time and fall time to define the recovery time.

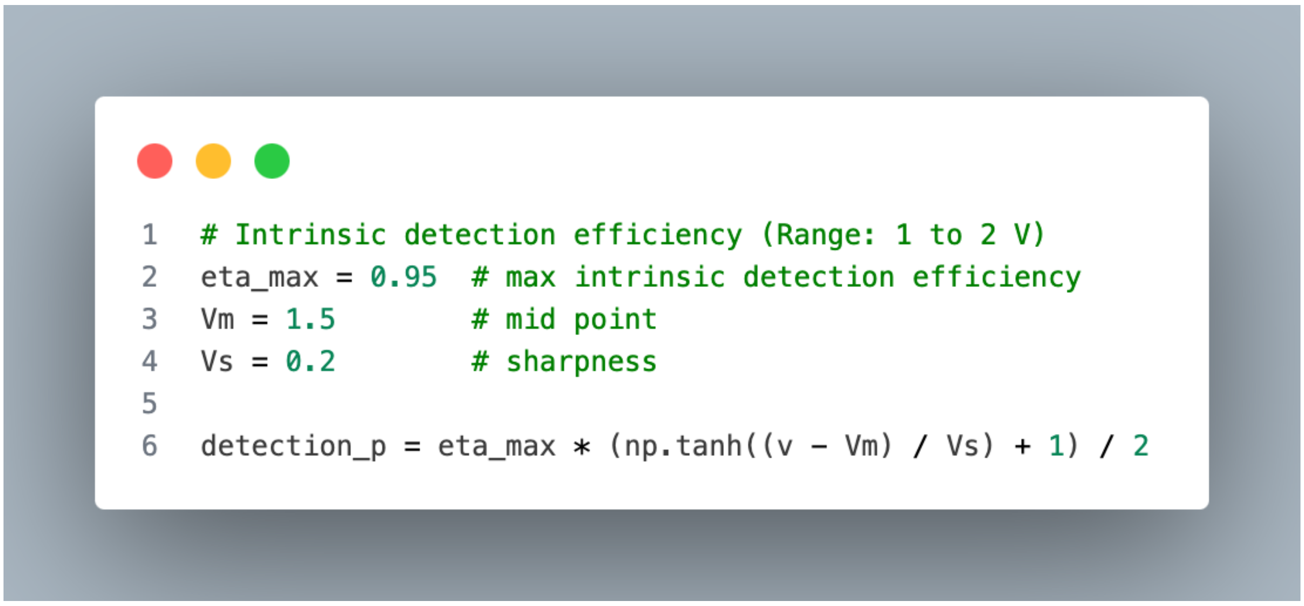
The **for loop** is used to filter out arrival time points that occur during the recovery time, and the random.random()function is used to achieve the probabilistic detection of photons for the SNSPD.

PossionProcess.py



```python
time_interval = 1/lambda_rate  # time interval
time_point = 0
arrival_time_points = []

while time_point < t:
    arrival_time = np.random.exponential(scale = time_interval)
    time_point += arrival_time

    if time_point < t:
        arrival_time_points.append(time_point)
```

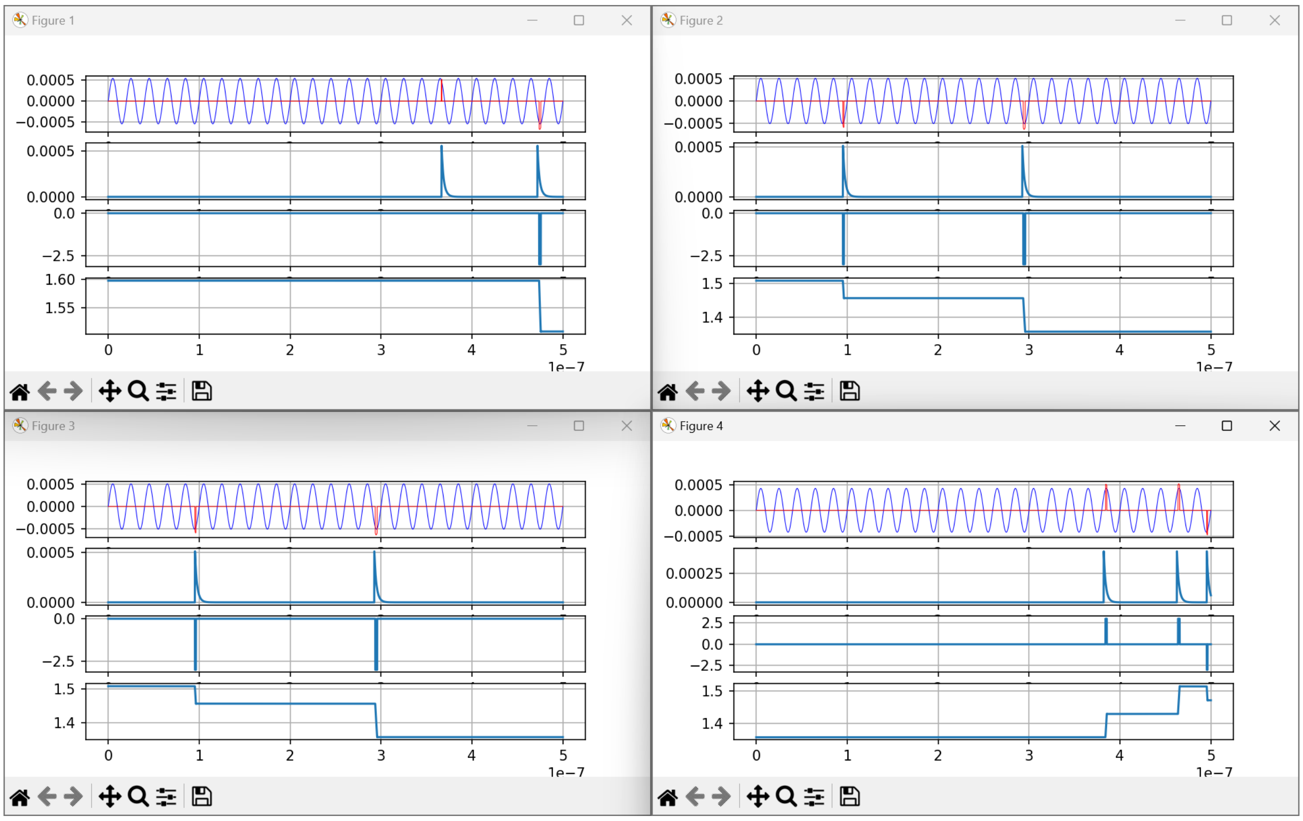
This code is used to generate photon arrival events with a Poisson distribution and to produce the arrival time points for further use.



```python
# Intrinsic detection efficiency (Range: 1 to 2 V)
eta_max = 0.95  # max intrinsic detection efficiency
Vm = 1.5        # mid point
Vs = 0.2        # sharpness

detection_p = eta_max * (np.tanh((v - Vm) / Vs) + 1) / 2
```

Since the voltage input of the SNSPD affects its intrinsic detection efficiency, this code is used to calculate the detection probability for a specific voltage.

Running example with the initial voltage set at 1.5V and iteration times set to 4 (5 simulations in total). After the first simulation, the voltage level is changed to around 1.6V.



Some place can be optimised:

Sometimes, the photon is detected by the SNSPD model, but no pulse is generated, leading to no change in the voltage level. This issue is due to the A.C. current source is modified by the D.C. bias current pulse of SNSPD detection. Achieving a perfect modification of the A.C. current source might fix this problem.

Additionally, the photon generation and intrinsic probability calculation can be more complex to achieve more realistic results. Currently, only a simple exponential calculation is used to define the Poisson distribution of photons in this project.

Reference

[1] Karl K Berggren et al, "A superconducting nanowire can be modeled by using SPICE", Supercond. Sci. Technol. 31 055010, 2018.

[2] Andrew J. Kerman et al, "Kinetic-inductance-limited reset time of superconducting nanowire photon counters", Appl. Phys. Lett. 88, 111116, 2006.

GitHub Repos Investigated

[1] qnngroup/snspd-spice

[2] DongHoonPark/ltspice_pytool

[3] nunobrum/PyLTSpice