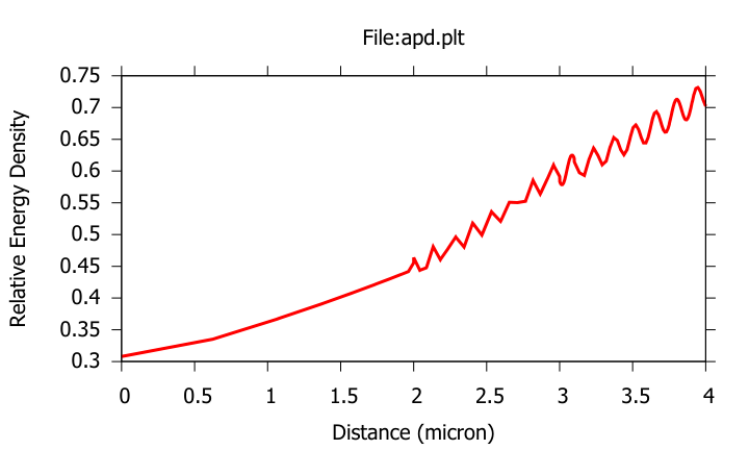
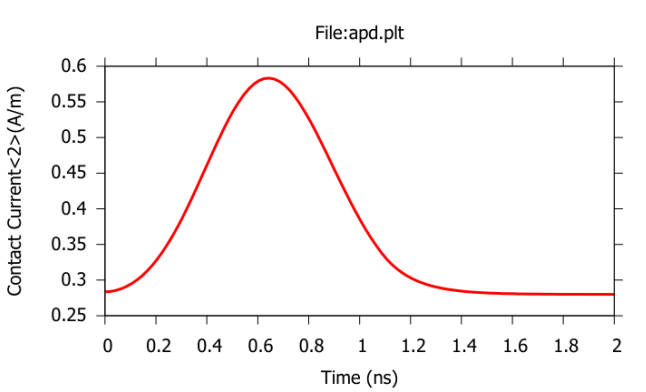
Crosslight APSYS Timing Jitter Extraction

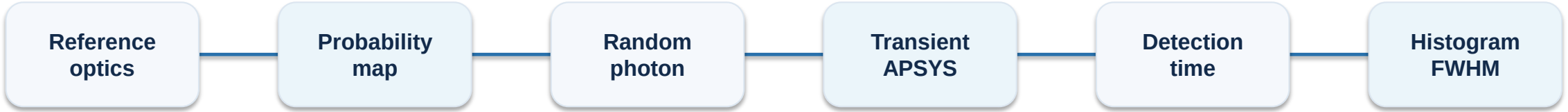
Tutorial: optical-weighted random single-photon excitation with trap-history-aware transient scans

APSYS + Python / AI workflow





Tutorial goal and simulation idea

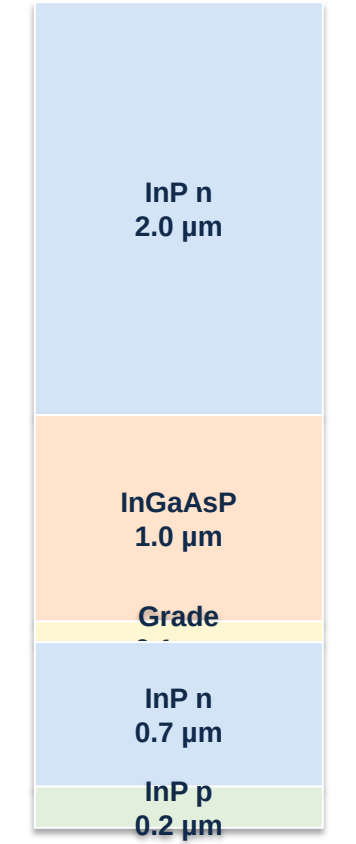


- Use the reference optical-energy distribution as the physical weight for single-photon generation in the 1D mesh.
- For each Monte Carlo event, choose a mesh location and randomized Gaussian start time gsn_t1.
- Run transient scan sequences so trap capture/release can influence later photon pulses.
- Extract detection time from the terminal current waveform and convert the distribution to timing jitter.

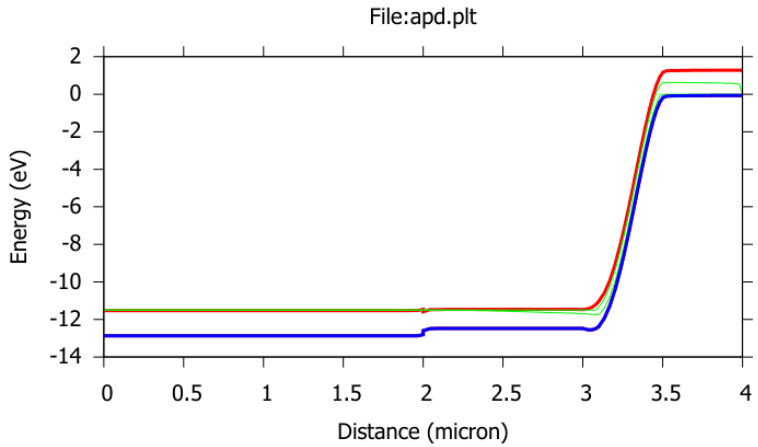
Customer deliverable

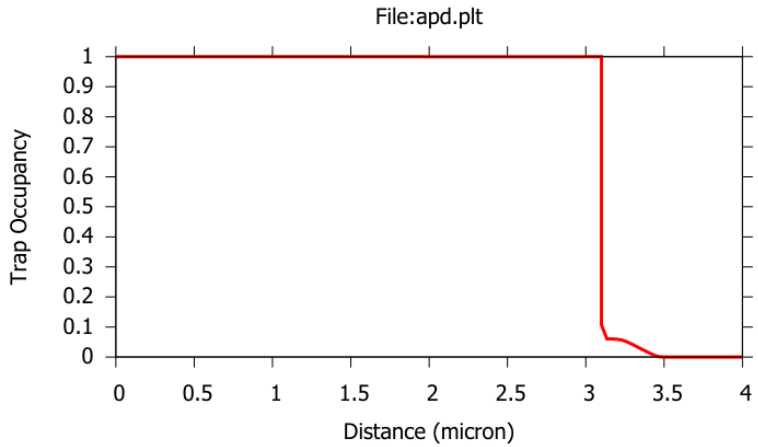
- Input: APD structure + reference optical run
- Automation: Python-generated APSYS cases
- Output: jitter histogram, σ, FWHM

Starting point: 1D APD stack and transient decks



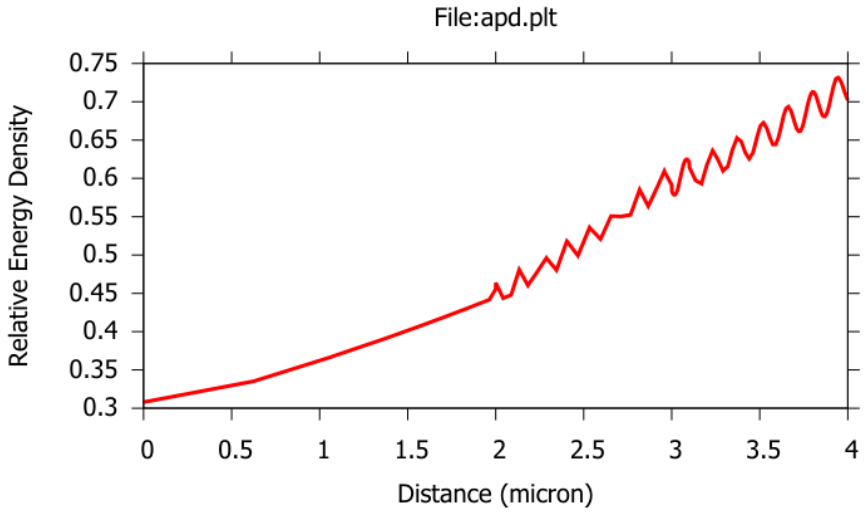
1D APD mesh

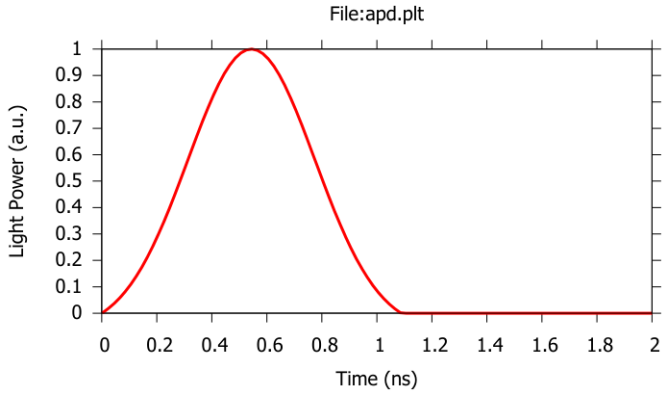




- The uploaded layer file defines a 4 µm vertical APD stack with InP / InGaAsP / graded / InP layers.
- The transient decks include donor traps in InP and InGaAsP, then bias to 11.5 V before the optical pulse scan.
- V3 keeps this deck structure but replaces a fixed photon coordinate with weighted random mesh coordinates.

Reference simulation: get optical-energy weight





Reference deck uses light_power, not single_photon_generation

```
light_power wavelength=0.9
incident_power=1.e4

plot_1d variable=optical_energy
from=(0.5 0.) to=(0.5, 5.)
```

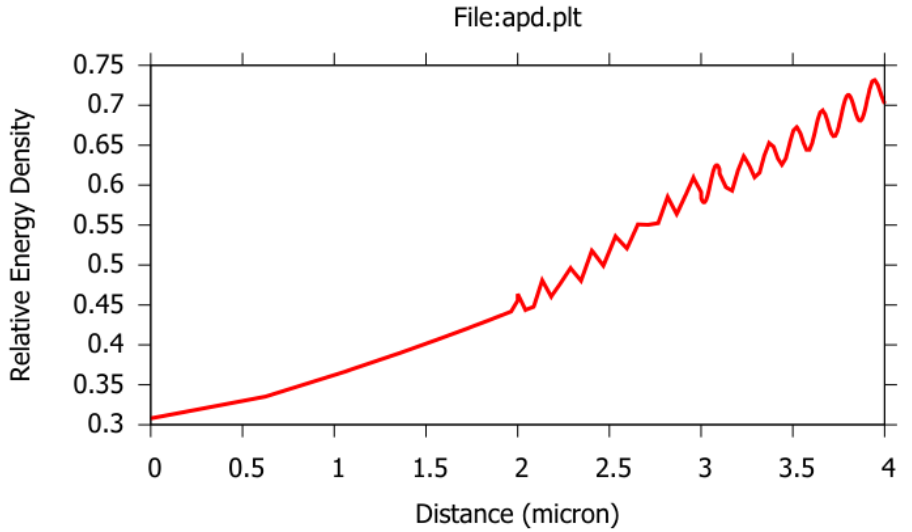
- The optical-energy curve is the depth-dependent probability weight for where the absorbed photon is generated.
- Because this is a 1D tutorial, the random variable is the vertical mesh coordinate y.
- The same Gaussian light pulse gives the time distribution; gsn_t1 can then be randomized for each trial.

Convert optical energy to a mesh-point probability map

For mesh point i

$$w_i = \text{optical\_energy}(y_i) \times \Delta y_i$$
$$P_i = w_i / \Sigma_j w_j$$
$$CDF_i = \Sigma_{j \le i} P_j$$

- Use cell length Δy_i when the mesh is nonuniform.
- Use the CDF for inverse transform sampling: find the first index with $CDF_i \ge$ random(0,1).
- The selected y_i is inserted into single_photon_generation for that Monte Carlo event.



Use as spatial weight, not as a deterministic source location

AI / Python generator: randomized photon list

```
# Inputs from APSYS post-processing
mesh_y, optical_energy =
read_raw_output('optical_energy.txt')
cell_dy = mesh_cell_width(mesh_y)
weight = optical_energy * cell_dy
prob = weight / weight.sum()
cdf = np.cumsum(prob)

# Monte Carlo event table
for k in range(N_events):
u = rng.random()
i = np.searchsorted(cdf, u)
y_hit = mesh_y[i]
t_hit = rng.uniform(t_min, t_max)  # randomized gsn_t1
write_apd_event(k, y_hit, t_hit)
```

Event table produced by Python / AI assistant

- event_id
- y_hit
- x_hit = 0.2 μm or centerline
- z_hit = 0
- gsn_t1
- random_seed
- case filename

Recommended: keep the event table as CSV for reproducibility.

APSYS template: replace fixed source with generated event

Current fixed-source example

```
single_photon_generation  &&
xcord=0.2 ycord=3.2 zcord=0 &&
pulse=0.5e-9 wavelength=0.9 &&
      photon_number=1000
```

Generated event-source example

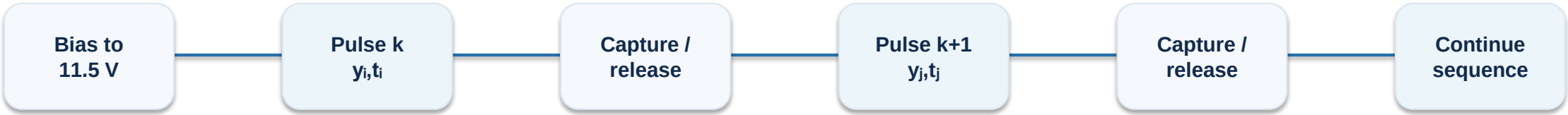
```
single_photon_generation  &&
xcord={x_hit} ycord={y_hit} zcord=0 &&
pulse=0.5e-9 wavelength=0.9 &&
      photon_number={Nph}
```

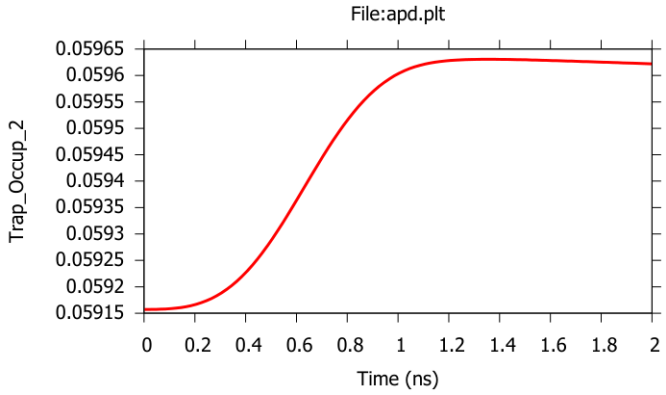
Randomized timing

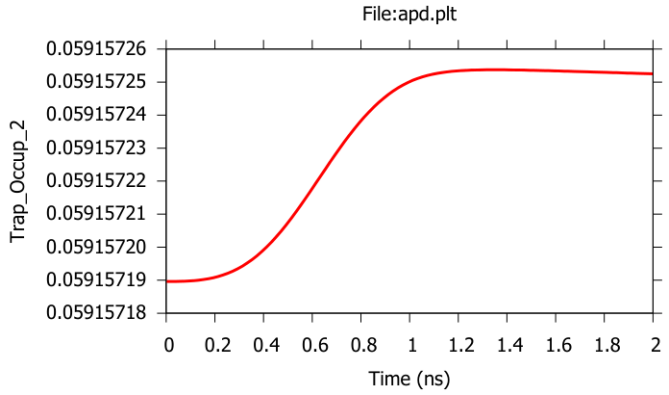
```
scan_function label=gs_func
      type=gaussian
    gsn_dt=0.544e-9
    gsn_t1={t_hit}
```

- For a strict single-photon simulation, set the event photon number according to the APSYS single-photon model and customer calibration.
- For early tutorial runs, a larger photon_number can be used as a detectable current-response proxy, then reduced after the extraction flow is validated.
- Keep wavelength, pulse width, bias point, and trap settings identical across Monte Carlo events unless testing sensitivity.

Trap-history-aware scan strategy

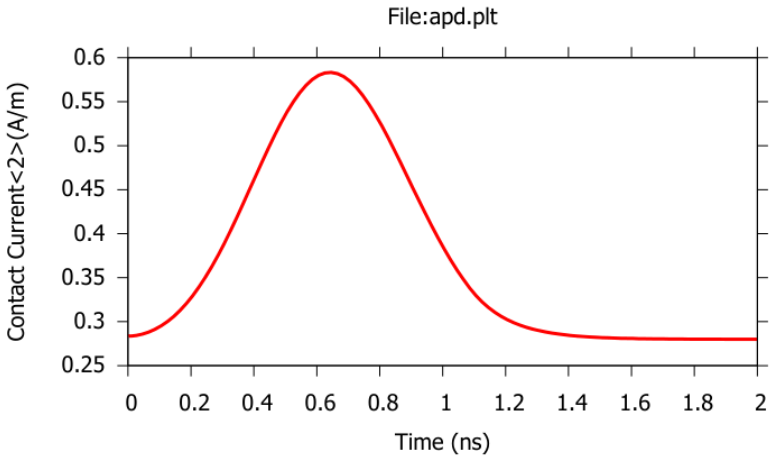


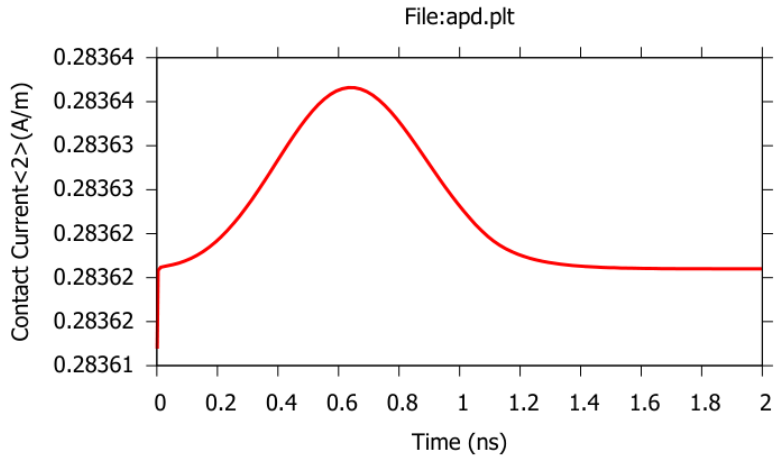




- Do not reset the trap state between closely spaced pulses when studying afterpulsing or pulse-pulse interaction.
- Randomized gsn_t1 values should be scheduled inside one transient history, or reloaded from saved states if separate jobs are required.
- The trap occupancy plot is the customer-facing evidence that optical pulses can change the internal state.

Transient extraction: define detection time





Detection-time rule

- Baseline subtract: I_sig(t)=I(t)-I_dark(t)
- Choose threshold: e.g., 50% of event peak or fixed current threshold
- Use interpolation around the threshold crossing for sub-step timing
- Save t_det for each event into a CSV table

Use one rule consistently across all Monte Carlo trials.

From event table to timing jitter

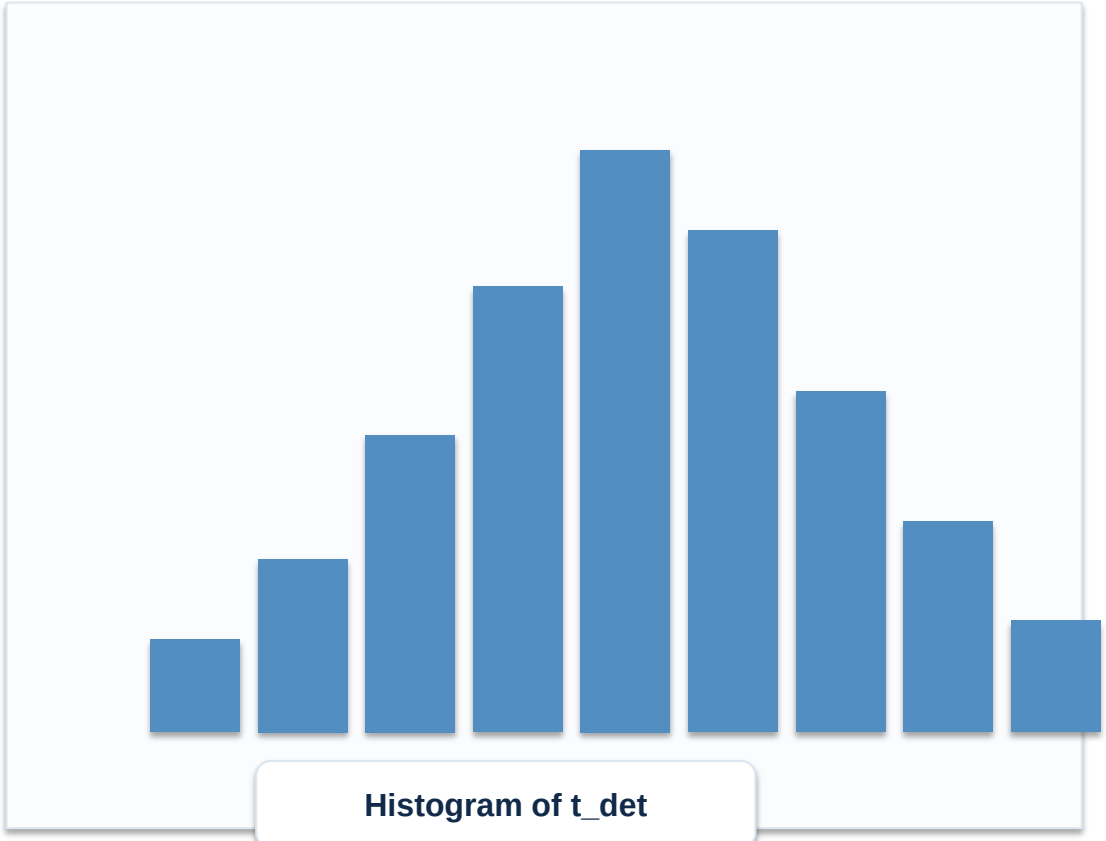
Event CSV columns

```
event_id, y_hit, gsn_t1, t_det, status
  0001, 3.18e-6, 0.11e-9, 0.67e-9, OK
  0002, 2.93e-6, 0.38e-9, 0.91e-9, OK
                ...
```

Statistics

$$\sigma = \mathrm{std}(t_{det})$$

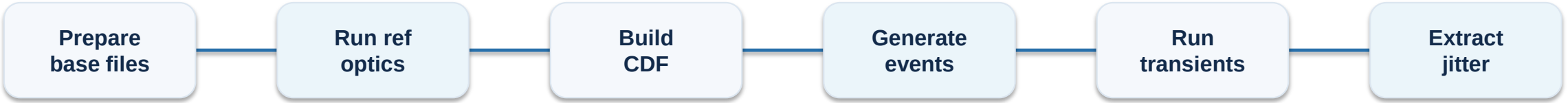
$$\mathrm{FWHM} \approx 2.355 \times \sigma$$



Histogram of t_det

- Report both raw histogram and Gaussian fit if applicable.
- Separate intrinsic timing spread from numerical timestep and threshold sensitivity.

Recommended customer automation workflow



Input files

- apd.layer
- ref_apd.sol / plt
- apd.sol / plt
- optical_energy raw output

Generated files

- events.csv
- apd_event_####.sol
- APD_Curr_####.txt
- jitter_summary.csv

Customer report

- Optical-energy map
- Random hit distribution
- Representative transients
- Histogram + σ + FWHM

Tip: keep random seed and APSYS version in the output summary for auditability.

Customer QA checklist

Physics checks

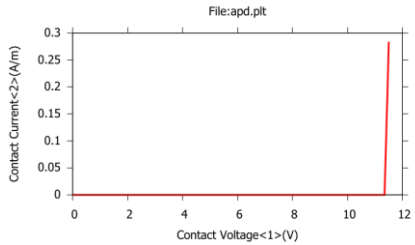
- Breakdown / avalanche bias is reasonable
- Optical-energy profile matches the intended illumination geometry
- Trap occupancy evolves during pulses and relaxes in the expected direction
- Timing extraction threshold is stable

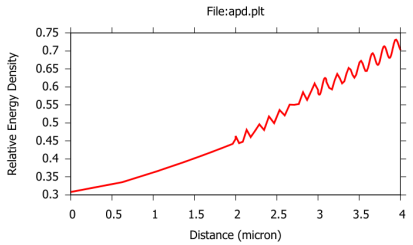
Numerical checks

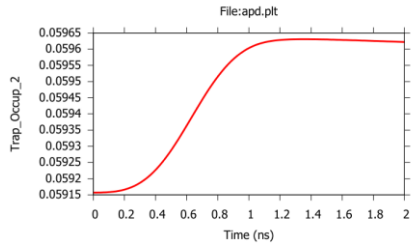
- Transient timestep is much smaller than target jitter
- Newton tolerances converge for all event cases
- Failed events are marked, not silently removed
- Mesh weighting includes cell width for nonuniform mesh

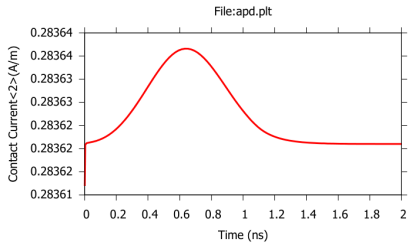
Statistics checks

- Use enough Monte Carlo events
- Report σ, FWHM, mean delay, and sample count
- Compare single-pulse and trap-history cases
- Repeat with a second random seed









Appendix: APSYS / Python integration pattern

```
1. Run reference deck
apsys ref_apd.sol
pstprc ref_apd.plt  # export optical_energy(y)

2. Build weighted random event table
python build_events.py --optical optical_energy.txt --n
1000

3. Create event decks
python make_apd_cases.py --template apd_template.sol --
events events.csv

4. Run transient cases or pulse sequence
apsys apd_event_0001.sol ...

5. Extract detection times
python extract_tdet.py --threshold 0.5_peak --curr
APD_Curr_*.txt

6. Report jitter
python jitter_report.py --input tdet.csv
```

Decision for V3 / production flow

- Option A: one long transient sequence with repeated pulses and continuous trap history.
- Option B: many separate events initialized from saved trap states.
- Option C: independent single-pulse events for intrinsic timing jitter only.

Recommendation

- Use Option A for afterpulsing interaction studies.
- Use Option C as the baseline before adding trap-history effects.

Two-level Monte Carlo structure

Spatial randomization = project folders

- Example: create 100 folders
- Each folder chooses one photon-hit location
- Location is sampled from optical-energy probability map
- The selected mesh point is written into that folder's apd.sol

Temporal randomization = scans inside each deck

- Example: 9 transient scan statements per apd.sol
- Each scan has randomized gs_t1
- Trap occupancy carries from scan 1 → scan 9
- Pulse-to-pulse interaction occurs through trap capture/release

Statistics example: 100 spatial folders × 9 randomized pulse scans = 900 timing samples.

APSYS deck pattern: 9 randomized transient scans

For folder_i, keep one photon-hit location fixed:

```
single_photon_generation xcord=... ycord=y_i pulse=... wavelength=... photon_number=1
```

Then insert multiple transient scans in the same apd.sol:

```
scan var=time value_to=Tstop var2=light function_label2=gs_func_1 ...
scan_function label=gs_func_1 type=gaussian gsn_dt=... gsn_t1=random_t1
scan var=time value_to=Tstop var2=light function_label2=gs_func_2 ...
scan_function label=gs_func_2 type=gaussian gsn_dt=... gsn_t1=random_t2
…
scan var=time value_to=Tstop var2=light function_label2=gs_func_9 ...
scan_function label=gs_func_9 type=gaussian gsn_dt=... gsn_t1=random_t9
```

Important: do not restart between the 9 scans, otherwise trap-mediated pulse interaction is lost.