ChatGPT

# PV Curve Analysis with pandapower

A **P–V (power–voltage) curve** plots the relationship between load (active power) and bus voltage under increasing load, and its "nose" (tip) indicates the maximum loadability (voltage collapse point) of the system [1] [2]. In practice, one chooses a standard IEEE test network (e.g. 9, 14, 39, 57, 118 bus systems) and gradually increases load at a bus or across all loads until the power flow fails. Pandapower includes many IEEE cases via `pandapower.networks` (e.g. `case9()`, `case14()`, `case39()`, `case57()`, `case118()`) [3] [4]. When setting up a PV sweep, key inputs include:

- **Test system selection:** Use a known IEEE case (9, 14, 39, 57, 118 bus). Pandapower provides these via functions like `pn.case14()` or `pn.case118()` [3] [5]. These are converted from MATPOWER/PYPOWER data and represent standard base-case loads and generation.
- **Load increment size:** Choose a small load step (typically 0.01–0.05 p.u. per step) to trace the curve accurately. For example, one study increased system load by 0.01 p.u. per step (≈41 MW for their system) at constant power factor [6]. Smaller steps (e.g. 0.005 p.u.) may be used near collapse to refine the nose.
- **Maximum load factor:** Continue the sweep until convergence fails. The "collapse point" is reached when no solution exists beyond a certain load multiplier. In practice one scales loads until run-time divergence (e.g. ~2.0–2.2 times the base load in one study [7]). That last converged point is taken as the nose (voltage stability limit).
- **All loads vs single bus:** You can scale *all* loads uniformly (to get a system-wide PV curve) or increase one load bus at a time (to examine a specific bus's margin). In either case, loads are usually modeled as PQ (constant P, Q) and their power factor kept constant during scaling [6]. The choice depends on the analysis goal: a single-bus PV curve reveals local weakness, while a total-scaling PV curve shows global margin.

Proper normalization (per-unit) and maintaining reactive power proportion (constant PF) are important so that the shape of the curve reflects true system physics. All chosen loads/generators should be "in service," and any slack or reference bus must be fixed (usually a swing generator) so that increasing load forces the system toward its stability limit.

## PV Sweep Setup in pandapower

To compute a PV curve in pandapower, one performs a parametric sweep of power-flow solutions. The typical procedure is:

1. **Load the network:** For example:

```
import pandapower as pp
import pandapower.networks as pn
net = pn.case14()   # IEEE 14-bus system
```

This uses an IEEE test case with default load and generation. Pandapower also offers case9, case39, case57, case118, etc. for these standard networks [3] [4] [5].

2. **Identify target bus(es):** Decide which load bus(es) to increase. You can loop over all `net.load` entries or select a particular bus index (e.g. `load_idx = net.load.index[0]`).

3. **Scale loads in steps:** Keep a copy of the base loads (e.g. `baseP = net.load['p_mw'].copy()`) and for each step multiply by a factor (1.00, 1.01, 1.02, …) up to the maximum factor. For example:

```
factors = np.arange(1.0, 2.05, 0.01)  # 0%-105% increase
for fac in factors:
    net.load['p_mw'] = baseP * fac
    # (Optionally also scale net.load['q_mvar'] if constant PF)
    pp.runpp(net)
    ...
```

   Each iteration calls `pp.runpp(net)` to solve the power flow at the new load. Pandapower's `runpp()` uses a Newton-Raphson solver by default [8]. One can specify algorithms if needed (e.g. `algorithm='iwamoto_nr'` for robustness) [8].

4. **Run power flow and check convergence:** After each load increase, call `pp.runpp(net)`. Pandapower will update `net.res_bus` with the solution if it converges [9]. To handle failures, you can catch exceptions or check `net["converged"]` (pandapower usually raises a `PowerflowNotConverged` exception or sets `net["converged"] = False`). For example:

```
try:
    pp.runpp(net)
except pp.LoadflowNotConverged:
    break   # stop at collapse
```

   or after `runpp`, check `if not net.converged: break`. In practice, one stops the sweep when the solver fails (indicating the nose has been passed) [10].

5. **Store results:** On each successful run, record the loading factor and the resulting bus voltages (and optionally powers). Pandapower stores results in `net.res_bus`, which includes the bus voltages (`vm_pu`) and angles [9]. For a single-bus PV curve, you would track `net.res_bus.vm_pu` at the specific bus. For a multi-bus curve, you may record each bus voltage per step.

Throughout the sweep, keep other system parameters (like generator outputs except the slack) fixed. It's important to disable any automatic controls (OLTC, AVR) or else they may adjust and mask the stability limit. In simple static PV analysis, treat transformers and generators as fixed except the swing bus which supplies extra load.

## Extracting Voltages and Plotting PV Curves

After each power flow solution, bus voltages are available in `net.res_bus['vm_pu']` [9]. To build the PV curve, one typically compiles a series of (P_load, V_bus) points:

- **Active power axis:** For a single-bus curve, plot the incremental *bus* active load P (in p.u. or MW) on the horizontal axis. This could be simply the cumulative increase (`base_P * factor`). For an "all-bus" curve, use total system load or the loading multiplier as the x-axis.

- **Voltage axis:** Plot the monitored bus's voltage magnitude `vm_pu` on the vertical axis. Often the voltage at the same bus whose load is increased is used. If scaling all loads, one could plot the minimum bus voltage in the system or a particular critical bus.

In code, you might collect values in lists or a Pandas DataFrame. For example:

```python
pv_points = []
baseP = net.load['p_mw'].copy()
for fac in np.arange(1.0, 2.0, 0.01):
    net.load['p_mw'] = baseP * fac
    try:
        pp.runpp(net)
    except pp.LoadflowNotConverged:
        break
    Vm = net.res_bus.vm_pu[bus_idx]   # voltage of bus of interest
    pv_points.append((net.load.p_mw.sum(), Vm))
```

After the loop, one can separate the list into arrays `P_load, V_bus` and use Matplotlib or Seaborn:

```python
import matplotlib.pyplot as plt
P_load = [pt[0] for pt in pv_points]
V_bus = [pt[1] for pt in pv_points]
plt.plot(P_load, V_bus, marker='o')
plt.xlabel("Load (MW)"); plt.ylabel("Voltage (p.u.)")
plt.title("PV Curve at bus %d" % bus_idx)
plt.axvline(P_nose, color='r', linestyle='--')  # mark nose if known
```

This will show the characteristic nose shape: as load increases, voltage decreases until the tip. Plotting multiple buses on one graph (or separate subplots) can reveal weaker buses that collapse sooner. Ensure axes are labeled and the curves are clearly distinguished.

## Determining the Nose (Collapse) Point

The **nose point** on the PV curve is the maximum load (horizontal axis) at which a valid power flow exists (voltage is lowest but still solvable) [1] [2]. Beyond this point the system cannot supply more load and voltages collapse. In practice, the nose is identified as:

- **Last converged step:** During the sweep, it is usually taken as the loading factor just before `runpp` diverges. For example, if the 2.05× load point failed, then 2.04× (the previous step) is the nose. In many case studies the collapse was found around 2.0–2.2 p.u. of base load [7].
- **Curve extremum:** More precisely, one could fit a smooth curve to (P,V) points and find the maximum of P(V) (where dV/dP = 0). This requires interpolation. However, for most cases the discrete sweep suffices, since smaller steps near the expected tip give good accuracy.

At the nose, the voltage is minimal for that loading. According to theory, operating beyond the nose (if somehow still converged by a different method) would lead to an unstable branch with much lower voltages [2]. Therefore, one should record the nose load and voltage, and maintain operating margins (keeping actual load a safe distance left of the nose) to avoid collapse.

## Example Code and Best Practices

A sketch of a Pandapower implementation combining the above steps is:

```
import pandapower as pp, pandapower.networks as pn
import numpy as np, matplotlib.pyplot as plt

# 1. Load test case
net = pn.case14()  # e.g. 14-bus system

# 2. Choose load bus(es) to vary (e.g. bus 6)
load_idx = 6  # index of PQ bus
# Option A: increase only one load at that bus
# find all loads at bus 6
target_loads = net.load[net.load.bus == load_idx].index

# 3. Prepare for sweep
baseP = net.load.p_mw.copy()
factors = np.arange(1.0, 2.05, 0.01)  # scaling factor from 1.0 to ~2.0

P_values = []; V_values = []
for fac in factors:
    net.load['p_mw'] = baseP * fac
    # keep Q proportional if constant PF:
    net.load['q_mvar'] = net.load['q_mvar'] * fac

    # Run power flow
    try:
        pp.runpp(net)
    except Exception:
        print(f"Loadflow failed at factor {fac:.2f}")
        break

    # Extract voltage at the bus of interest
    V = net.res_bus.vm_pu[load_idx]
    P = net.load.p_mw[net.load.bus == load_idx].sum()
    V_values.append(V)
    P_values.append(P)

# 4. Plot PV curve
plt.plot(P_values, V_values, '-o')
plt.xlabel("Load at bus %d (MW)"%load_idx)
plt.ylabel("Bus %d Voltage (p.u.)" % load_idx)
plt.title("PV Curve for bus %d" % load_idx)
plt.grid(True)
plt.show()
```

**Best practices:**
- **Validation:** Before the sweep, solve the base case and check all voltages are near 1.0 p.u. and no limits are hit. Ensure the network is correctly configured (one slack bus, correct transformer turns ratios, etc.).
- **Step size tuning:** Use coarse steps initially, then refine near collapse (e.g. halve step size after voltage falls sharply). This improves accuracy of the nose point.
- **Convergence handling:** Use try/except or check `net.converged`. If many fails occur, try a different solver or damping. Pandapower's Newton-Raphson (`algorithm='nr'`) is robust, but the `iwamoto_nr` option can help if convergence is difficult [8].
- **Output analysis:** Save intermediate results (voltage, line flows, bus loadings) for further analysis. Voltage magnitudes are in `net.res_bus.vm_pu` [9] and power flows in `net.res_line`/`net.res_load`.

By following these steps, one can generate P–V curves for the chosen buses on standard IEEE systems using pandapower. The plots will show how voltage collapses as load grows, and the nose point marks the static voltage stability limit [1] [10].

**Sources:** Pandapower documentation and studies on voltage stability [3] [8] [9] [1] [6] were used to outline the methodology above.

---

[1] [2] Power-voltage curve - Wikipedia
https://en.wikipedia.org/wiki/Power-voltage_curve

[3] [4] [5] Power System Test Cases — pandapower 3.1.1 documentation
https://pandapower.readthedocs.io/en/develop/networks/power_system_test_cases.html

[6] [7] [10] Analysis of Voltage Stability of the Slovak Republic's Power System
https://www.mdpi.com/2227-9717/10/12/2613?src=695918

8 Balanced AC Power Flow — pandapower 2.9.0 documentation
https://pandapower.readthedocs.io/en/v2.9.0/powerflow/ac.html

9 Bus — pandapower 2.0.0 documentation
https://pandapower.readthedocs.io/en/v2.0.0/elements/bus.html