Stabilizer Mechanics – Python Reference Implementation

```
import math
import random
from dataclasses import dataclass
from typing import Callable, Dict, List, Tuple

# ----------------------------
# 0) CONFIG
# ----------------------------
@dataclass
class Config:
    steps: int = 200
    target: float = 1.0

    # Noise / deviation knobs (simulate "regular deviated code")
    noise_std: float = 0.08
    drift: float = 0.002  # slow drift away from target

    # Baseline update strength
    k_p: float = 0.18

    # Stabilizer knobs
    a: float = 0.12          # energy-paced correction gain (0 < a < 1 typical)
    beta: float = 0.04       # nonlinear clamp strength (>=0)
    use_tanh_clamp: bool = True

    # Optional velocity damping (helps kill oscillation)
    use_velocity_damping: bool = True
    lam: float = 0.35        # damping on velocity-like term

    # Safety clamp on delta (prevents runaway steps)
    delta_clip: float = 0.35


# ----------------------------
# 1) YOUR "REGULAR DEVIATED" UPDATE (EDIT THIS)
# ----------------------------
def base_update(x: float, meas: float, cfg: Config) -> float:
    """
    Replace this with your real 'regular deviated code' update rule.
    Return Δ_base: how you change x each step.
    """
    # Example: proportional step toward measurement (which is noisy/drifty)
    return cfg.k_p * (meas - x)


# ----------------------------
# 2) STABILIZER MECHANICS (WRAPPER)
# ----------------------------
def stabilizer_delta(x: float, meas: float, cfg: Config) -> float:
    """
    Stabilizer adds a paced correction term based on current error e = (x - target),
    and optionally a nonlinear tanh clamp that prevents large error from producing
    uncontrolled correction.
    """
    e = x - cfg.target

    # paced correction
    d_paced = -cfg.a * e

    # optional nonlinear clamp
    d_nl = 0.0
    if cfg.use_tanh_clamp and cfg.beta > 0.0:
        d_nl = -cfg.beta * math.tanh(e)

    return d_paced + d_nl
```

```python
def clip(v: float, lim: float) -> float:
    if v > lim: return lim
    if v < -lim: return -lim
    return v


def run_sim(name: str,
            cfg: Config,
            use_stabilizer: bool):

    x = 0.0
    v = 0.0
    data = {"x": [], "e": [], "delta": [], "energy": [], "meas": []}

    drift_state = 0.0
    for k in range(cfg.steps):
        drift_state += cfg.drift
        meas = cfg.target + drift_state + random.gauss(0.0, cfg.noise_std)

        d = base_update(x, meas, cfg)

        if use_stabilizer:
            d += stabilizer_delta(x, meas, cfg)

        if cfg.use_velocity_damping:
            v = (1.0 - cfg.lam) * v + d
            d_eff = v
        else:
            d_eff = d

        d_eff = clip(d_eff, cfg.delta_clip)

        x = x + d_eff
        e = x - cfg.target

        data["x"].append(x)
        data["e"].append(e)
        data["delta"].append(d_eff)
        data["energy"].append(e * e)
        data["meas"].append(meas)

    return data


def summarize(label, data):
    e = data["e"]
    energy = data["energy"]
    abs_e_final = abs(e[-1])
    rms_e = math.sqrt(sum(v*v for v in e) / len(e))
    total_energy = sum(energy)

    print(f"=== {label} ===")
    print(f"Final |error|:     {abs_e_final}")
    print(f"RMS error:         {rms_e}")
    print(f"Total energy Σe²:  {total_energy}")


def main():
    random.seed(7)
    cfg = Config()

    baseline = run_sim("baseline", cfg, use_stabilizer=False)
    stabilized = run_sim("stabilized", cfg, use_stabilizer=True)

    summarize("Baseline", baseline)
    summarize("With stabilizer", stabilized)


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