CLT-speed-race-exercise

February 8, 2026

1 CLT Speed Race

This notebook presents an interactive, simulation-based visualization of the **speed of convergence in the Central Limit Theorem (CLT)**. While the classical CLT guarantees that the standardized sample mean converges in distribution to a standard normal random variable, the **rate of convergence** depends materially on the underlying population distribution — particularly its skewness, kurtosis, and tail behavior. The aim is to develop geometric and empirical intuition for *how fast* the Gaussian approximation becomes accurate across different distributions.

1.1 Mathematical Setup

Let $X_1, \dots, X_n$ be i.i.d. random variables with mean μ and variance σ^2, assumed finite unless explicitly noted otherwise.

The sample mean is

$$\bar{X}_n = \frac{1}{n} \sum_{i=1}^{n} X_i.$$

The standardized mean is

$$Z_n = \frac{\sqrt{n}(\bar{X}_n - \mu)}{\sigma}.$$

Under the classical Lindeberg–Lévy Central Limit Theorem, if

$$0 < \sigma^2 < \infty,$$

then

$$Z_n \xrightarrow{d} \mathcal{N}(0, 1).$$

1.2 Simulation Design

For an increasing sequence of sample sizes n, we run a Monte Carlo experiment with R independent repetitions and compute

$$Z_n^{(1)}, \dots, Z_n^{(R)}.$$

At each step, the notebook updates:

- A histogram of the simulated Z_n
- The standard normal density $\phi(x)$

- The empirical–normal CDF difference

$$\widehat{F}_n(x) - \Phi(x)$$

- An empirical Kolmogorov–Smirnov distance (computed numerically on a fixed grid)

$$\max_{x \in \mathcal{G}} |\widehat{F}_n(x) - \Phi(x)|$$

To keep axes stable for visualization, the histogram display is clipped to a fixed window (e.g. $[-12, 12]$), while discrepancy metrics (ECDF difference and KS) are computed using the unclipped simulated draws. The fraction of draws outside the display window is reported as a tail diagnostic.

This produces a dynamic **"speed race"** illustrating how quickly the Gaussian approximation stabilizes as n grows.

1.3 Distributional Effects on Convergence

Although the CLT holds whenever variance is finite, convergence speed varies substantially:

Distribution	Behavior
Uniform	Very fast convergence
Normal	Immediate baseline
Exponential	Slower due to skewness
Laplace	Moderate heavy tails
Student-t (small df)	Very slow convergence
Pareto ($\alpha \leq 2$)	Infinite variance $\rightarrow$ CLT fails
Cauchy	No mean/variance $\rightarrow$ CLT fails

Heavy-tailed distributions can slow convergence dramatically, and in infinite-variance cases the classical Gaussian CLT does not apply.

1.4 Interpretation

If the CLT approximation is accurate:

- The histogram of Z_n approaches the standard normal density
- The ECDF difference approaches zero
- The KS distance decreases
- The curves stabilize as n grows

If variance is infinite, stabilization does **not** occur — and this is theoretically correct.

1.5 Purpose

This notebook aims to:

- Compare convergence speeds across distributions
- Build intuition for finite-sample normal approximation
- Demonstrate when Gaussian approximation is reliable
- Illustrate when and why it fails

1.6 Reproducibility

All simulations are vectorized NumPy operations with a fixed random seed for deterministic replication.

Author: Muhammed İkbal Yılmaz
Email: myucanlar@gmail.com

[13]:
```
import numpy as np
import matplotlib.pyplot as plt
import ipywidgets as W
import math, os, shutil, subprocess
from math import sqrt, pi
from datetime import datetime
from pathlib import Path
from IPython.display import display, clear_output

# ---- Backend ----
INTERACTIVE = False
try:
    import ipympl  # noqa
    get_ipython().run_line_magic("matplotlib", "widget")
    INTERACTIVE = True
except Exception:
    try:
        get_ipython().run_line_magic("matplotlib", "inline")
    except Exception:
        pass

# ---- Styling ----
plt.rcParams.update({
    "font.family": "serif",
    "font.serif": ["Times New Roman", "DejaVu Serif"],
    "font.size": 12,
    "axes.titlesize": 13,
    "axes.labelsize": 12,
    "figure.titlesize": 14,
})
```

```
# ---- Normal CDF/PDF (no SciPy) ----
_erf = np.vectorize(math.erf)
Phi = lambda x: 0.5 * (1 + _erf(np.asarray(x) / sqrt(2)))
phi = lambda x: (1 / sqrt(2 * pi)) * np.exp(-0.5 * np.asarray(x) ** 2)

DISTS =␣
 ↪["Normal","Uniform","Exponential","Laplace","Lognormal","Chi-square","F","Student-t","Paret

def moments(d, p):
    if d == "Normal":      return p["mu"], p["sg"]
    if d == "Uniform":
        a, b = p["a"], max(p["b"], p["a"] + 1e-12)
        return 0.5*(a+b), sqrt((b-a)**2/12)
    if d == "Exponential": s = p["sc"]; return s, s
    if d == "Laplace":     return p["mu"], sqrt(2)*p["b"]
    if d == "Lognormal":
        m, s = p["ml"], p["sl"]
        mu = np.exp(m + 0.5*s*s)
        var = (np.exp(s*s)-1) * np.exp(2*m + s*s)
        return mu, sqrt(var)
    if d == "Chi-square":  df = p["df"]; return df, sqrt(2*df)
    if d == "F":
        d1, d2 = p["d1"], p["d2"]
        mu = (d2/(d2-2)) if d2 > 2 else None
        sg = (sqrt((2*(d2**2)*(d1+d2-2))/(d1*((d2-2)**2)*(d2-4)))) if d2 > 4␣
 ↪else None
        return mu, sg
    if d == "Student-t":
        df = p["df"]
        return (0.0 if df > 1 else None), (sqrt(df/(df-2)) if df > 2 else None)
    if d == "Pareto":
        xm, a = p["xm"], p["al"]
        mu = (a*xm/(a-1)) if a > 1 else None
        sg = (sqrt((a*xm*xm)/(((a-1)**2)*(a-2)))) if a > 2 else None
        return mu, sg
    return None, None

def draw(d, rng, shape, p):
    if d == "Normal":      return rng.normal(p["mu"], p["sg"], size=shape)
    if d == "Uniform":     return rng.uniform(p["a"], max(p["b"], p["a"] +␣
 ↪1e-12), size=shape)
    if d == "Exponential": return rng.exponential(p["sc"], size=shape)
    if d == "Laplace":     return rng.laplace(p["mu"], p["b"], size=shape)
    if d == "Lognormal":   return rng.lognormal(p["ml"], p["sl"], size=shape)
    if d == "Chi-square":  return rng.chisquare(p["df"], size=shape)
    if d == "F":           return rng.f(p["d1"], p["d2"], size=shape)
    if d == "Student-t":   return rng.standard_t(p["df"], size=shape)
```

```
    if d == "Pareto":      return p["xm"] * (1 + rng.pareto(p["al"],␣
 ↪size=shape))
    return rng.standard_t(1.0, size=shape)  # Cauchy

def ns_path(nmax, steps, n0=10):
    return np.unique(np.round(np.exp(np.linspace(np.log(n0), np.log(nmax),␣
 ↪steps))).astype(int))

def safe_slug(s):
    return "".join(c if c.isalnum() or c in "-_." else "_" for c in str(s))

def make_gif_from_frames(frames_dir: Path, gif_path: Path, fps: int):
    # Pure-Python (Pillow) GIF writer
    from PIL import Image
    frames = sorted(frames_dir.glob("frame_*.png"))
    if not frames:
        raise RuntimeError("No frames found to build GIF.")
    imgs = [Image.open(p).convert("P", palette=Image.Palette.ADAPTIVE) for p in␣
 ↪frames]
    duration_ms = int(1000 / max(1, fps))
    imgs[0].save(
        gif_path,
        save_all=True,
        append_images=imgs[1:],
        duration=duration_ms,
        loop=0,
        optimize=False,
    )

def make_mp4_with_ffmpeg(frames_dir: Path, mp4_path: Path, fps: int):
    ff = shutil.which("ffmpeg")
    if not ff:
        raise RuntimeError("ffmpeg not found.")
    # Use numbered PNG sequence
    cmd = [
        ff, "-y",
        "-framerate", str(max(1, fps)),
        "-i", str(frames_dir / "frame_%05d.png"),
        "-pix_fmt", "yuv420p",
        "-vf", "pad=ceil(iw/2)*2:ceil(ih/2)*2",
        str(mp4_path),
    ]
    subprocess.run(cmd, check=True, stdout=subprocess.PIPE, stderr=subprocess.
 ↪PIPE)

# ---- UI ----
```

```
dist  = W.Dropdown(options=DISTS, value="Student-t", description="Dist:",␣
 ↪layout=W.Layout(width="220px"))
seed  = W.IntText(value=42, description="Seed:", layout=W.Layout(width="160px"))
R     = W.IntSlider(value=2000, min=200, max=6000, step=100, description="R:",␣
 ↪continuous_update=False, layout=W.Layout(width="420px"))
nmax  = W.IntSlider(value=10000, min=200, max=40000, step=200,␣
 ↪description="nmax:", continuous_update=False, layout=W.Layout(width="420px"))
steps = W.IntSlider(value=30, min=6, max=60, step=1, description="steps:",␣
 ↪continuous_update=False, layout=W.Layout(width="420px"))
ms    = W.IntSlider(value=30, min=0, max=250, step=5, description="ms:",␣
 ↪continuous_update=False, layout=W.Layout(width="420px"))

# Recording controls
record = W.Checkbox(value=False, description="Record", indent=False)
fmt = W.Dropdown(options=[("GIF (no ffmpeg)", "gif"), ("MP4 (needs ffmpeg)",␣
 ↪"mp4"), ("Frames only", "frames")],
                 value="gif", description="Format:", layout=W.
 ↪Layout(width="220px"))
fps = W.IntSlider(value=20, min=5, max=60, step=1, description="FPS:",␣
 ↪continuous_update=False, layout=W.Layout(width="420px"))
outdir = W.Text(value="clt_renders", description="Save dir:", layout=W.
 ↪Layout(width="420px"))
dpi = W.IntSlider(value=140, min=80, max=220, step=10, description="DPI:",␣
 ↪continuous_update=False, layout=W.Layout(width="420px"))

runb  = W.Button(description="Run", button_style="success", icon="play")
stopb = W.Button(description="Stop", button_style="danger", icon="stop")
status = W.HTML()
out = W.Output()

STATE = {"stop": False}
stopb.on_click(lambda _: STATE.__setitem__("stop", True))

# Params
mu = W.FloatText(value=0.0, description="mu");  sg = W.FloatText(value=1.0,␣
 ↪description="sg")
a  = W.FloatText(value=0.0, description="a");   b  = W.FloatText(value=1.0,␣
 ↪description="b")
sc = W.FloatText(value=1.0, description="sc")
lbm= W.FloatText(value=0.0, description="mu");  lbb= W.FloatText(value=1.0,␣
 ↪description="b")
ml = W.FloatText(value=0.0, description="ml");  sl = W.FloatText(value=0.6,␣
 ↪description="sl")
df = W.FloatText(value=5.0, description="df");  d1 = W.FloatText(value=5.0,␣
 ↪description="d1"); d2 = W.FloatText(value=10.0, description="d2")
```

```
xm = W.FloatText(value=1.0, description="xm");  al = W.FloatText(value=2.5,␣
 ↪description="al")

pbox = W.HBox([])

def P():
    d = dist.value
    return {
        "Normal":      {"mu": mu.value, "sg": sg.value},
        "Uniform":     {"a": a.value, "b": b.value},
        "Exponential": {"sc": sc.value},
        "Laplace":     {"mu": lbm.value, "b": lbb.value},
        "Lognormal":   {"ml": ml.value, "sl": sl.value},
        "Chi-square":  {"df": df.value},
        "Student-t":   {"df": df.value},
        "F":           {"d1": d1.value, "d2": d2.value},
        "Pareto":      {"xm": xm.value, "al": al.value},
    }.get(d, {})

def upd(*_):
    d = dist.value
    pbox.children = (
        (mu, sg) if d == "Normal" else
        (a, b) if d == "Uniform" else
        (sc,) if d == "Exponential" else
        (lbm, lbb) if d == "Laplace" else
        (ml, sl) if d == "Lognormal" else
        (df,) if d in ["Chi-square", "Student-t"] else
        (d1, d2) if d == "F" else
        (xm, al) if d == "Pareto" else
        tuple()
    )

dist.observe(upd, names="value")
upd()

def run(_=None):
    STATE["stop"] = False
    d = dist.value
    p = P()
    rng = np.random.default_rng(int(seed.value))
    Rv = int(R.value)
    ns = ns_path(int(nmax.value), int(steps.value))

    mu0, sg0 = moments(d, p)
    clt_ok = (mu0 is not None) and (sg0 is not None) and np.isfinite(mu0) and␣
 ↪np.isfinite(sg0) and (sg0 > 0)
```

```
    s = np.zeros(Rv)
    nprev = 0

    LIM = 12.0
    B = 140
    bins = np.linspace(-LIM, LIM, B+1)
    ctr  = 0.5*(bins[:-1] + bins[1:])
    w    = bins[1] - bins[0]
    grid = np.linspace(-LIM, LIM, 401)
    Phig = Phi(grid)

    # ---- Recording setup ----
    rec = bool(record.value)
    timestamp = datetime.now().strftime("%Y%m%d_%H%M%S")
    run_name = f"CLT_{safe_slug(d)}_R{Rv}_nmax{int(nmax.value)}_steps{int(steps.
↪value)}_{timestamp}"
    base_dir = Path(outdir.value).expanduser().resolve()
    frames_dir = base_dir / run_name / "frames"
    frames_dir.mkdir(parents=True, exist_ok=True)
    out_gif = base_dir / run_name / f"{run_name}.gif"
    out_mp4 = base_dir / run_name / f"{run_name}.mp4"

    with out:
        out.clear_output(wait=True)
        plt.close("all")

        fig, (axH, axD) = plt.subplots(1, 2, figsize=(13.2, 5.2),␣
↪constrained_layout=True)

        bars = axH.bar(ctr, np.zeros_like(ctr), width=w, alpha=0.28,␣
↪edgecolor="black", linewidth=0.5)
        gx = np.linspace(-LIM, LIM, 600)
        axH.plot(gx, phi(gx), color="black", lw=2)

        (lineD,) = axD.plot(grid, np.zeros_like(grid), lw=2, color="black")
        axD.axhline(0, color="black", lw=1)

        axH.set(xlim=(-LIM, LIM), xlabel=r"$Z_n$", ylabel="density",␣
↪title=r"$Z_n$ histogram (fixed bins)")
        axD.set(xlim=(-LIM, LIM), xlabel="x", ylabel=r"$\hat F(x)-\Phi(x)$",␣
↪title="ECDF difference (speed)")

        display(fig)
        fig.canvas.draw()

        if not clt_ok:
```

```
            status.value = ("<b style='color:#b00'>CLT is not applicable here: "
                            "finite mean/standard deviation is not available.</
↪b>")
            return

        frame_idx = 0

        for n in ns:
            if STATE["stop"]:
                status.value = "<b style='color:#b00'>Stopped.</b>"
                break

            k = n - nprev
            X = draw(d, rng, (Rv, k), p)
            s += X.sum(1)
            nprev = n

            # ---- Raw stats (correct) ----
            z_raw = sqrt(n) * (s/n - mu0) / sg0
            z_raw = z_raw[np.isfinite(z_raw)]
            if z_raw.size < 200:
                continue

            zsort = np.sort(z_raw)
            M = zsort.size
            Fhat = np.searchsorted(zsort, grid, side="right") / M
            Dv = Fhat - Phig
            lineD.set_ydata(Dv)
            ks = float(np.max(np.abs(Dv)))

            # ---- Display histogram (stable) ----
            tail = float(np.mean(np.abs(z_raw) > LIM))
            z_plot = np.clip(z_raw, -LIM, LIM)
            cnt, _ = np.histogram(z_plot, bins=bins, density=True)
            for bb, hh in zip(bars, cnt):
                bb.set_height(float(hh))
            axH.set_ylim(0, max(0.35, float(cnt.max()) * 1.12))

            fig.suptitle(
                f"CLT Speed Race - {d} | n={n} R={Rv} | KS={ks:.4f} |␣
↪tail(|Z|>{LIM:g})={tail:.4f}"
            )

            # Render / refresh
            if not INTERACTIVE:
                clear_output(wait=True)
                display(fig)
```

```
            fig.canvas.draw_idle()

            # ---- Save frame ----
            if rec:
                frame_path = frames_dir / f"frame_{frame_idx:05d}.png"
                fig.savefig(frame_path, dpi=int(dpi.value), bbox_inches="tight")
                frame_idx += 1

            plt.pause(ms.value / 1000)

        # ---- Finalize recording ----
        if rec and frame_idx > 0:
            try:
                if fmt.value == "frames":
                    status.value = f"<b style='color:green'>Saved frames:</b>␣
 ↪{frames_dir}"
                elif fmt.value == "gif":
                    make_gif_from_frames(frames_dir, out_gif, int(fps.value))
                    status.value = f"<b style='color:green'>Saved GIF:</b>␣
 ↪{out_gif}"
                else:  # mp4
                    try:
                        make_mp4_with_ffmpeg(frames_dir, out_mp4, int(fps.
 ↪value))
                        status.value = f"<b style='color:green'>Saved MP4:</b>␣
 ↪{out_mp4}"
                    except Exception:
                        # Fallback to GIF (no ffmpeg / error)
                        make_gif_from_frames(frames_dir, out_gif, int(fps.
 ↪value))
                        status.value = (f"<b style='color:#b50'>MP4 failed␣
 ↪(ffmpeg missing or error). "
                                        f"Saved GIF instead:</b> {out_gif}")
            except Exception as e:
                status.value = f"<b style='color:#b00'>Recording failed:</b>␣
 ↪{e}"
        elif not STATE["stop"]:
            status.value = "<b style='color:green'>Done.</b>"

runb.on_click(run)

display(W.VBox([
    W.HBox([dist, seed, runb, stopb]),
    W.HBox([R, nmax]),
    W.HBox([steps, ms]),
    pbox,
```

```
    W.HBox([record, fmt]),
    W.HBox([fps, dpi]),
    outdir,
    status,
    out
]))
```

```
VBox(children=(HBox(children=(Dropdown(description='Dist:', index=7,␣
↪layout=Layout(width='220px'), options=('N…
```