```python
#!/usr/bin/env python3
import math
import os
import re
import subprocess
import tempfile
import matplotlib
matplotlib.use("Agg")  # Headless-Backend fÃ¼r Server ohne X11/GUI
import matplotlib.pyplot as plt

Z0 = 50.0                    # Bezugsimpedanz in Ohm
Z_CENTER = 2.0               # Tiefster Punkt in der Mitte (m)
END_HEIGHTS = range(2, 11)   # HÃ¶he der EndaufhÃ¤ngungen: 2 m bis 10 m
TOTAL_LEN = 39.2             # GesamtdrahtlÃ¤nge (m)
N_SEGMENTS = 31              # Ungerade Segmentanzahl fÃ¼r mittige Einspeisung
WIRE_RADIUS = 0.001026       # Drahtradius (m)

NEC_HEADER = """CM 80 m catenary dipole (center {z_c} m, ends {z_e} m) above average ground (Sommerfeld, GN 2)
CE
"""

NEC_FOOTER = """GE 1
GN 2 0 0 0 13 0.005
EX 0 {center_tag} 1 0 1 0
FR 0 31 0 0 3.5 0.01
RP 0 46 73 1000 0 0 2 5
EN
"""

def build_catenary_deck(z_center: float, z_end: float) -> tuple[str, float, float]:
    half_len = TOTAL_LEN / 2.0
    dz = z_end - z_center
    if dz < 0:
        raise ValueError("EndhÃ¶he muss >= ScheitelhÃ¶he (z_center) sein.")

    ds = TOTAL_LEN / N_SEGMENTS
    nodes = []

    if dz == 0:
        a = float("inf")
        x_end = half_len
        for k in range(N_SEGMENTS + 1):
            s_k = (k - N_SEGMENTS / 2.0) * ds
            nodes.append((s_k, z_center))
    else:
```

```
        a = (half_len ** 2 - dz ** 2) / (2.0 * dz)
        x_end = a * math.asinh(half_len / a)
        for k in range(N_SEGMENTS + 1):
            s_k = (k - N_SEGMENTS / 2.0) * ds
            x_k = a * math.asinh(s_k / a)
            z_k = z_center + (s_k ** 2) / (math.sqrt(a ** 2 + s_k ** 2) + a)
            nodes.append((x_k, z_k))

    gw_lines = []
    for i in range(N_SEGMENTS):
        x1, z1 = nodes[i]
        x2, z2 = nodes[i + 1]
        gw_lines.append(
            f"GW {i + 1} 1 {x1:.6f} 0 {z1:.6f} {x2:.6f} 0 {z2:.6f} {WIRE_RADIUS:.6f}"
        )

    center_tag = (N_SEGMENTS + 1) // 2
    deck = (
        NEC_HEADER.format(z_c=z_center, z_e=z_end)
        + "\n".join(gw_lines)
        + "\n"
        + NEC_FOOTER.format(center_tag=center_tag)
    )
    return deck, x_end, a

def calc_swr(r: float, x: float, z0: float = 50.0) -> float:
    num = (r - z0) ** 2 + x ** 2
    den = (r + z0) ** 2 + x ** 2
    gamma = math.sqrt(num / den)
    return (1.0 + gamma) / (1.0 - gamma) if gamma < 1.0 else float("inf")

def parse_nec_output(out_file: str, z0: float = 50.0):
    freqs, swrs, impedances = [], [], []
    freq_mhz = None
    in_input_section = False

    with open(out_file, "r", encoding="utf-8", errors="replace") as f:
        for line in f:
            freq_match = re.search(r"FREQUENCY\s*[=:]\s*([0-9.eE+-]+)", line)
            if freq_match:
                freq_mhz = float(freq_match.group(1))

            if "ANTENNA INPUT PARAMETERS" in line:
                in_input_section = True
                continue

```

```
            if in_input_section:
                floats = re.findall(r"[+-]?\d+\.\d+[eE][+-]\d+", line)
                if len(floats) >= 6 and freq_mhz is not None:
                    r = float(floats[4])
                    x = float(floats[5])
                    freqs.append(freq_mhz)
                    swrs.append(calc_swr(r, x, z0))
                    impedances.append((r, x))
                    in_input_section = False

    return freqs, swrs, impedances

def create_plot(h_end: int, x_end: float, freqs, swrs, impedances, output_png: str):
    min_idx = min(range(len(swrs)), key=lambda i: swrs[i])
    min_f, min_swr = freqs[min_idx], swrs[min_idx]
    min_r, min_x = impedances[min_idx]

    if min_idx < len(freqs) * 0.2:
        ha, dx = "left", 5
    elif min_idx > len(freqs) * 0.8:
        ha, dx = "right", -5
    else:
        ha, dx = "center", 0

    fig, ax = plt.subplots(figsize=(8, 4.8), dpi=150)
    ax.plot(freqs, swrs, color="#1f77b4", linewidth=2, marker="o", markersize=4, label=f"SWR ({Z0:.0f} Î©)")

    ax.axhline(1.5, color="#2ca02c", linestyle="--", linewidth=1, label="SWR 1.5")
    ax.axhline(2.0, color="#d62728", linestyle=":", linewidth=1.2, label="SWR 2.0")

    ax.scatter([min_f], [min_swr], color="#ff7f0e", zorder=5, s=50)
    ax.annotate(
        f"Min: SWR {min_swr:.2f} @ {min_f:.3f} MHz\n(Z = {min_r:.1f} {min_x:+.1f}j Î©)",
        xy=(min_f, min_swr),
        xytext=(dx, 18),
        textcoords="offset points",
        ha=ha,
        fontsize=8.5,
        bbox=dict(boxstyle="round,pad=0.3", fc="white", ec="#ff7f0e", alpha=0.9),
        arrowprops=dict(arrowstyle="->", color="#ff7f0e"),
    )

    y_max = min(max(swrs) * 1.15, max(10.0, min_swr * 1.3))
    ax.set_title(
        f"SWR-Verlauf â€“ 80 m Kettenlinien-Dipol (Mitte {Z_CENTER:.0f} m, Enden {h_end} m, Spannw. {2*x_end:.1f} m)"
    )
```

```
    ax.set_xlabel("Frequenz (MHz)")
    ax.set_ylabel(f"SWR (Zâ‚€ = {Z0:.0f} Î©)")
    ax.set_xlim(min(freqs), max(freqs))
    ax.set_ylim(1.0, y_max)
    ax.grid(True, which="both", linestyle="--", alpha=0.4)
    ax.legend(loc="upper right")

    plt.tight_layout()
    plt.savefig(output_png)
    plt.close(fig)
    return min_f, min_swr, min_r, min_x

def main():
    with tempfile.TemporaryDirectory() as tmpdir:
        for h_end in END_HEIGHTS:
            deck, x_end, a = build_catenary_deck(Z_CENTER, float(h_end))
            nec_path = os.path.join(tmpdir, f"sim_cat_{h_end}.nec")
            out_path = os.path.join(tmpdir, f"sim_cat_{h_end}.out")

            with open(nec_path, "w", encoding="utf-8") as f:
                f.write(deck)

            subprocess.run(["nec2c", "-i", nec_path, "-o", out_path], check=True)
            freqs, swrs, impedances = parse_nec_output(out_path, Z0)

            if not freqs:
                raise RuntimeError(f"Keine Frequenzdaten fÃ¼r EndhÃ¶he {h_end} m gefunden.")

            output_png = f"swr_nvis80_cat_{h_end}m.png"
            min_f, min_swr, min_r, min_x = create_plot(h_end, x_end, freqs, swrs, impedances, output_png)
            a_str = f"{a:6.1f} m" if math.isfinite(a) else "   inf  "
            print(
                f"{output_png:22s} | Spannw.: {2*x_end:4.1f} m | a: {a_str} | "
                f"Min-SWR: {min_swr:5.2f} @ {min_f:.3f} MHz | Z = {min_r:5.1f} {min_x:+5.1f}j Î©"
            )

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