lorenz_ae370

April 11, 2025

1 Lorenz-63 Numerical Study

AE 370 Project 1 – Code Notebook

This notebook implements the Lorenz-63 chaotic system, verifies an adaptive RKF45 solver, and prepares for future bifurcation and Lyapunov analyses.

```
[4]: %matplotlib inline
     import numpy as np
     import matplotlib.pyplot as plt
     from scipy.integrate import solve_ivp
     from mpl_toolkits.mplot3d import Axes3D  # Ensures 3-D plotting works properly
```

1.1 1. Lorenz-63 Right-Hand Side

The Lorenz-63 system is given by:

$$\dot{x} = \sigma\,(y - x),$$
$$\dot{y} = x\,(r - z) - y,$$
$$\dot{z} = x\,y - \beta\,z,$$

with parameters: - $\sigma = 10$, - $\beta = \frac{8}{3}$, - $r \in [0, 50]$.

This system serves as our testbed for numerical experiments and chaos theory in engineering.

```
[5]: def lorenz_rhs(t, u, sigma=10.0, beta=8/3, r=28.0):
         """
         Computes the right-hand side of the Lorenz-63 equations.

         Parameters:
             t (float): Time variable (not used explicitly but required by␣
     ↪solve_ivp).
             u (array-like): State vector [x, y, z].
             sigma (float): Parameter, default 10.
             beta (float): Parameter, default 8/3.
             r (float): Parameter, default 28.

         Returns:
```

```
        ndarray: Time derivatives [dx/dt, dy/dt, dz/dt].
    """
    x, y, z = u
    return np.array([sigma * (y - x),
                     x * (r - z) - y,
                     x * y - beta * z])
```

1.2 2. Adaptive RKF45 Integrator and Method Discussion

We solve the Lorenz equations using an adaptive Runge–Kutta–Fehlberg 4(5) method (RKF45) available via SciPy's `solve_ivp`.

1.2.1 Butcher Tableau for RKF45

The coefficients for the RKF45 method are given by:

$$\begin{array}{c|cccccc}
0 & & & & & & \\
\frac{1}{4} & \frac{1}{4} & & & & & \\
\frac{3}{8} & \frac{3}{32} & \frac{9}{32} & & & & \\
\frac{12}{13} & \frac{1932}{2197} & -\frac{7200}{2197} & \frac{7296}{2197} & & & \\
1 & \frac{439}{216} & -8 & \frac{3680}{513} & -\frac{845}{4104} & & \\
\frac{1}{2} & -\frac{8}{27} & 2 & -\frac{3544}{2565} & \frac{1859}{4104} & -\frac{11}{40} & \\
\hline
 & \frac{16}{135} & 0 & \frac{6656}{12825} & \frac{28561}{56430} & -\frac{9}{50} & \frac{2}{55}
\end{array}$$

1.2.2 Local Truncation Error Discussion

The RKF45 method computes two approximations at every step: a 5th-order solution $y^{[5]}$ and a 4th-order solution $y^{[4]}$. Their difference,

$$\delta = y^{[5]} - y^{[4]},$$

provides an estimate of the local truncation error, which is $\mathcal{O}(\Delta t^5)$. This error estimate is used to adjust the time step dynamically, ensuring the integration remains both accurate and efficient.

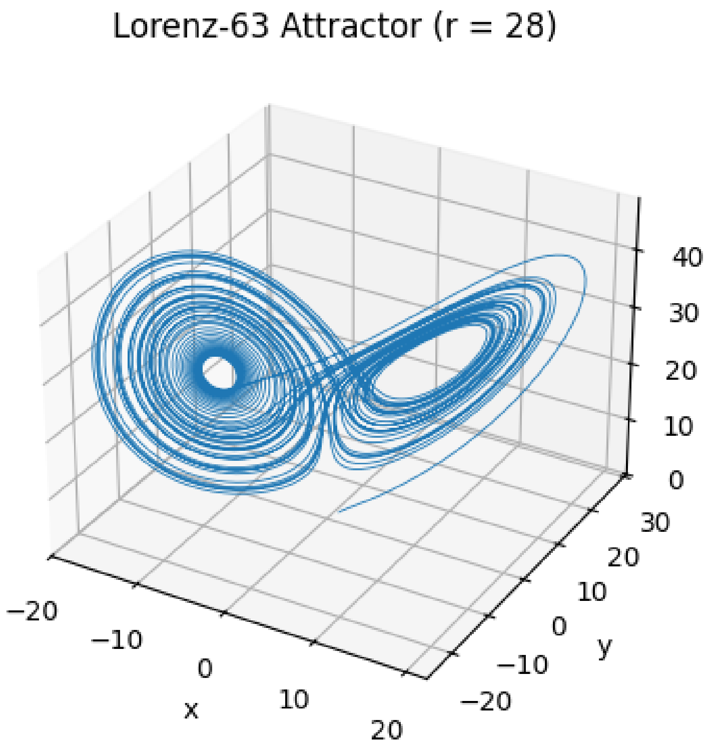
[6]:
```
def solve_lorenz(u0=(1.0, 1.0, 1.0), r=28.0, t_end=50.0, rtol=1e-8, atol=1e-10):
    """
    Integrate the Lorenz-63 system using an adaptive RKF45 method.

    Parameters:
        u0 (tuple): Initial condition (default: (1,1,1)).
        r (float): Rayleigh parameter (default: 28).
        t_end (float): Final time of integration.
        rtol (float): Relative tolerance.
        atol (float): Absolute tolerance.

    Returns:
        Bunch: A scipy.integrate.OdeResult object with time stamps and state␣
 ↪trajectory.
```

```
    """
    sol = solve_ivp(lorenz_rhs, (0.0, t_end), u0, args=(10.0, 8/3, r),
                    rtol=rtol, atol=atol, dense_output=True)
    return sol
```

1.3 3. Visualising the Strange Attractor

We now integrate the Lorenz-63 system with default parameters and plot its 3-D trajectory. This plot should reveal the characteristic "butterfly" structure of the attractor.

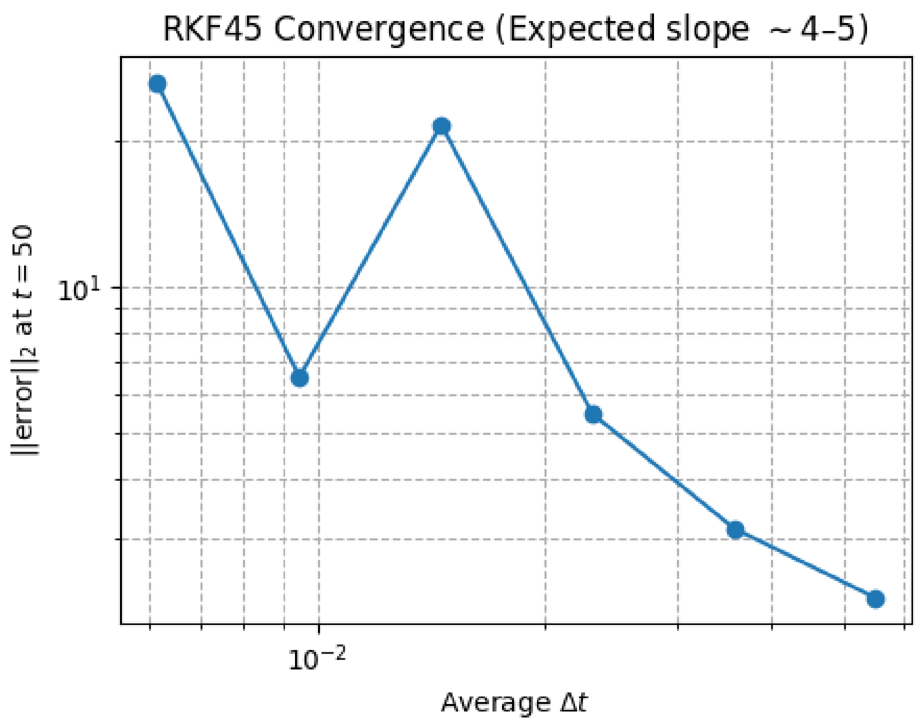
[7]:
```
# Run the solver and plot the Lorenz attractor in 3-D
sol = solve_lorenz()
fig = plt.figure(figsize=(6, 4))
ax = fig.add_subplot(111, projection='3d')
ax.plot(sol.y[0], sol.y[1], sol.y[2], lw=0.6)
ax.set_xlabel('x')
ax.set_ylabel('y')
ax.set_zlabel('z')
ax.set_title('Lorenz-63 Attractor (r = 28)')
plt.tight_layout()
plt.show()
```



1.4 4. Convergence Study

To verify our numerical implementation, we compare solutions computed with varying error tolerances.
We compute a "reference" solution using very tight tolerances and then measure the global error at $t = 50$ for increasing values of Δt.
In a log–log plot, the error should decrease with a slope near 4–5, which confirms the theoretical order of our RKF45 method.

[8]:
```
tol_list = 10.0 ** -np.arange(4, 10)  # Tolerances: 1e-4 to 1e-9
errors, dts = [], []

# Compute a high-accuracy reference solution
ref = solve_lorenz(rtol=1e-12, atol=1e-14).y[:, -1]

for tol in tol_list:
    sol = solve_lorenz(rtol=tol, atol=tol*1e-2)
    errors.append(np.linalg.norm(sol.y[:, -1] - ref))
    dts.append(np.mean(np.diff(sol.t)))

plt.figure(figsize=(5,4))
plt.loglog(dts, errors, 'o-')
plt.xlabel('Average $\\Delta t$')
# Use || for the error norm instead of \lVert and \rVert
plt.ylabel('$||\\text{error}||_2$ at $t = 50$')
plt.title('RKF45 Convergence (Expected slope $\\sim 4$-$5$)')
plt.grid(True, which='both', ls='--')
plt.tight_layout()
plt.show()
```



1.5 5. RKF45 Stability Region

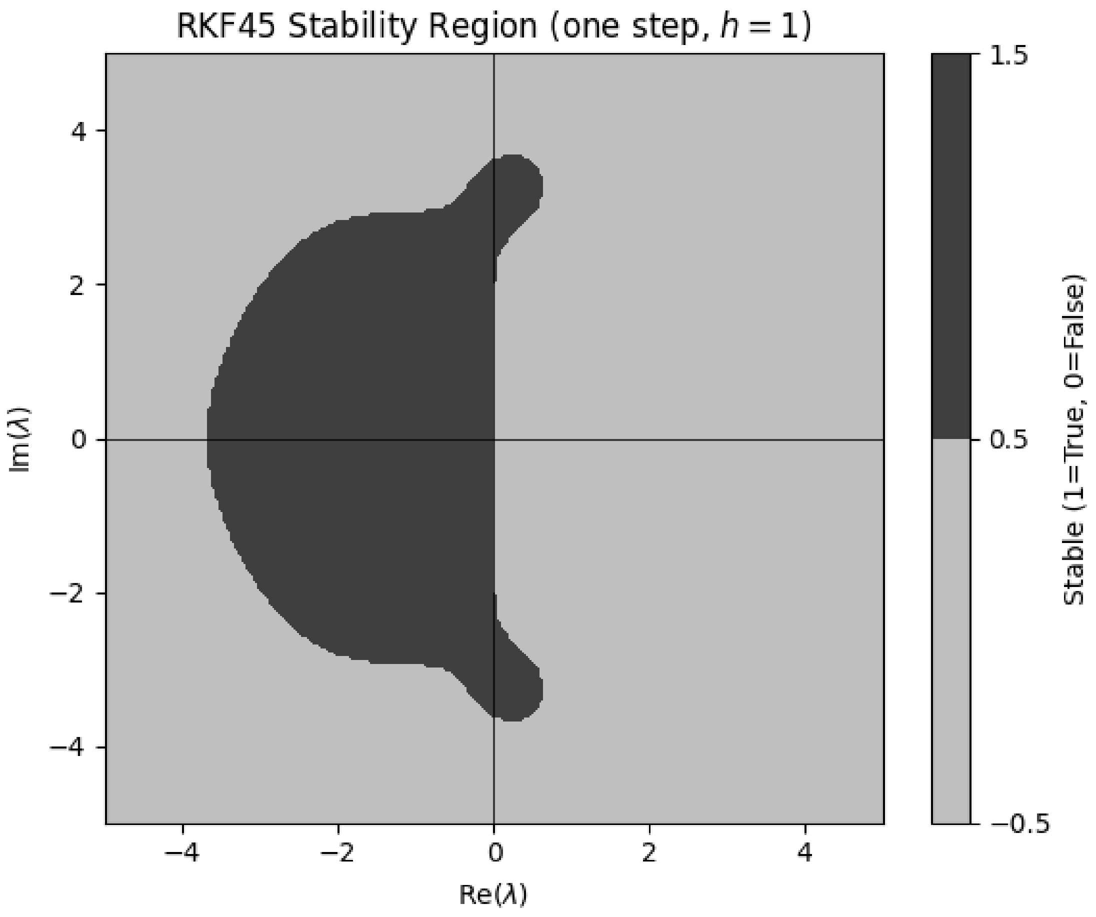
For further insight into the RKF45 method, we analyze the stability region for the scalar test equation

$$y' = \lambda y, \quad y(0) = 1,$$

using a single RKF45 step with $h = 1$. We consider a grid of λ values in the complex plane and check whether $|y_1| < 1$ (indicating stability).

Points where $|y_1| < 1$ belong to the stability region; otherwise, the method would be unstable for that λ.

[9]:
```
def rkf45_one_step(z, h=1.0):
    """
    Perform one RKF45 step for the test equation y' = z*y, starting from y(0)=1.

    Parameters:
        z (complex): The value of lambda in the test equation.
        h (float): Step size.

    Returns:
        complex: The 5th-order approximation y(h).
    """
    # Butcher tableau coefficients for RKF45 (only used in this one-step test)
    a = [
        [],                                 # i=0: no stages yet.
        [1/4],                              # i=1
        [3/32,       9/32],                 # i=2
        [1932/2197, -7200/2197, 7296/2197],    # i=3
        [439/216,   -8,       3680/513,   -845/4104],   # i=4
        [-8/27,      2,      -3544/2565, 1859/4104, -11/40]   # i=5
       ]
    c  = [0, 1/4, 3/8, 12/13, 1, 1/2]
    b5 = [16/135, 0, 6656/12825, 28561/56430, -9/50, 2/55]  # 5th order weights

    y = 1.0  # initial condition y(0)=1
    K = np.zeros(6, dtype=complex)

    for i in range(6):
        y_i = y + h * sum(a[i][j] * K[j] for j in range(i))
        K[i] = z * y_i

    y_next = y + h * sum(b5[i] * K[i] for i in range(6))
    return y_next

# Create a grid in the complex plane for lambda values.
N = 201
x_vals = np.linspace(-5, 5, N)
y_vals = np.linspace(-5, 5, N)
X, Y = np.meshgrid(x_vals, y_vals)
Z = X + 1j * Y

# Determine stability: stable if |y_1| < 1
stable_mask = np.zeros_like(X, dtype=bool)
for i in range(N):
    for j in range(N):
        y1 = rkf45_one_step(Z[i, j], h=1.0)
        stable_mask[i, j] = (abs(y1) < 1.0)

plt.figure(figsize=(6,5))
plt.contourf(X, Y, stable_mask, levels=[-0.5, 0.5, 1.5], cmap='binary')
plt.colorbar(label='Stable (1=True, 0=False)')
plt.axhline(0, color='k', lw=0.5)
plt.axvline(0, color='k', lw=0.5)
plt.title('RKF45 Stability Region (one step, $h=1$)')
plt.xlabel('Re($\\lambda$)')
plt.ylabel('Im($\\lambda$)')
plt.tight_layout()
plt.show()
```



```
[10]: # Sanity check: The origin should be an equilibrium for the Lorenz system.
      assert np.allclose(lorenz_rhs(0, (0, 0, 0)), (0, 0, 0)), "The origin is not an␣
       ↪equilibrium!"
      print("Sanity check passed: $f(0,0,0) = 0$.")
```

```
Sanity check passed: $f(0,0,0) = 0$.
```

1.6 6. Presentation of Results: Bifurcation Diagram

In this section, we investigate how the dynamics of the Lorenz-63 system change with the parameter (r). One common way to visualize these changes is to construct a *bifurcation diagram* for the variable (z).

Procedure: - We vary (r) over a given range (for example, from 20 to 50). - For each (r)-value, we integrate the system over a long enough time span to let transients decay. - We extract the local maxima (peaks) of (z(t)) in the post-transient part of the solution. - Finally, we plot these peak values versus (r).

The resulting diagram reveals transitions from steady behavior to periodic oscillations and to chaotic dynamics.

[11]:
```
import scipy.signal as signal

# Define the range of r values for the bifurcation study
r_values = np.linspace(20, 50, 300)
r_plot = []      # will hold r values corresponding to detected peaks
z_peak_plot = [] # will hold detected peak values for z

# Loop over each r value, integrate the Lorenz system, and extract peaks
for r in r_values:
    # Increase t_end to allow transients to decay; adjust as needed.
    sol = solve_lorenz(u0=(1, 1, 1), r=r, t_end=150.0, rtol=1e-8, atol=1e-10)
    t = sol.t
    z = sol.y[2]  # extract the z-component

    # Debug: Print minimum and maximum time in the integration
    # print(f"r = {r:.2f}, t_min = {t.min()}, t_max = {t.max()}")

    # Discard the transient portion (e.g. t < 120)
    mask = t > 120
    t_post = t[mask]
    z_post = z[mask]

    # Debug: Print number of points in post-transient data
    print(f"r = {r:.2f}, post-transient points = {len(z_post)}", end="; ")

    # Use SciPy's find_peaks to extract local maxima in the z data
    peaks, _ = signal.find_peaks(z_post, height=np.mean(z_post))
    print(f"peaks found = {len(peaks)}")

    if len(peaks) == 0:
        # If no peaks are detected, include the last value
        peak_values = [z_post[-1]]
    else:
        peak_values = z_post[peaks]

    # Append the r value and each corresponding peak value
    for pv in peak_values:
        r_plot.append(r)
        z_peak_plot.append(pv)

# Plot the bifurcation diagram
plt.figure(figsize=(6, 4))
plt.plot(r_plot, z_peak_plot, 'k.', markersize=1)
plt.xlabel('$r$')
plt.ylabel('Peak values of $z$')
plt.title('Bifurcation Diagram of the Lorenz-63 System')
plt.grid(True)
plt.tight_layout()
plt.show()
```

```
r = 20.00, post-transient points = 130; peaks found = 42
r = 20.10, post-transient points = 131; peaks found = 41
r = 20.20, post-transient points = 132; peaks found = 42
r = 20.30, post-transient points = 132; peaks found = 42
r = 20.40, post-transient points = 133; peaks found = 42
r = 20.50, post-transient points = 134; peaks found = 42
r = 20.60, post-transient points = 135; peaks found = 42
r = 20.70, post-transient points = 136; peaks found = 42
r = 20.80, post-transient points = 138; peaks found = 42
r = 20.90, post-transient points = 140; peaks found = 43
r = 21.00, post-transient points = 142; peaks found = 42
r = 21.10, post-transient points = 147; peaks found = 43
r = 21.20, post-transient points = 151; peaks found = 42
r = 21.30, post-transient points = 157; peaks found = 43
r = 21.40, post-transient points = 164; peaks found = 43
r = 21.51, post-transient points = 174; peaks found = 43
r = 21.61, post-transient points = 185; peaks found = 43
r = 21.71, post-transient points = 199; peaks found = 43
r = 21.81, post-transient points = 214; peaks found = 43
r = 21.91, post-transient points = 232; peaks found = 43
r = 22.01, post-transient points = 251; peaks found = 44
r = 22.11, post-transient points = 274; peaks found = 43
r = 22.21, post-transient points = 297; peaks found = 44
r = 22.31, post-transient points = 324; peaks found = 44
r = 22.41, post-transient points = 354; peaks found = 43
r = 22.51, post-transient points = 387; peaks found = 44
r = 22.61, post-transient points = 422; peaks found = 44
r = 22.71, post-transient points = 462; peaks found = 44
r = 22.81, post-transient points = 505; peaks found = 44
r = 22.91, post-transient points = 553; peaks found = 45
r = 23.01, post-transient points = 605; peaks found = 44
r = 23.11, post-transient points = 664; peaks found = 44
r = 23.21, post-transient points = 729; peaks found = 45
r = 23.31, post-transient points = 802; peaks found = 45
r = 23.41, post-transient points = 886; peaks found = 44
r = 23.51, post-transient points = 985; peaks found = 45
r = 23.61, post-transient points = 1109; peaks found = 45
r = 23.71, post-transient points = 1293; peaks found = 45
r = 23.81, post-transient points = 3158; peaks found = 36
r = 23.91, post-transient points = 3236; peaks found = 36
r = 24.01, post-transient points = 3265; peaks found = 36
r = 24.11, post-transient points = 3081; peaks found = 36
r = 24.21, post-transient points = 3248; peaks found = 36
r = 24.31, post-transient points = 3225; peaks found = 36
r = 24.41, post-transient points = 3173; peaks found = 36
```

```
r = 24.52, post-transient points = 3092; peaks found = 37
r = 24.62, post-transient points = 3220; peaks found = 36
r = 24.72, post-transient points = 3307; peaks found = 36
r = 24.82, post-transient points = 3377; peaks found = 36
r = 24.92, post-transient points = 3283; peaks found = 36
r = 25.02, post-transient points = 3205; peaks found = 37
r = 25.12, post-transient points = 2825; peaks found = 40
r = 25.22, post-transient points = 3261; peaks found = 38
r = 25.32, post-transient points = 3310; peaks found = 37
r = 25.42, post-transient points = 3351; peaks found = 37
r = 25.52, post-transient points = 3273; peaks found = 38
r = 25.62, post-transient points = 3289; peaks found = 37
r = 25.72, post-transient points = 3094; peaks found = 39
r = 25.82, post-transient points = 3233; peaks found = 38
r = 25.92, post-transient points = 3159; peaks found = 38
r = 26.02, post-transient points = 3352; peaks found = 38
r = 26.12, post-transient points = 3281; peaks found = 38
r = 26.22, post-transient points = 3285; peaks found = 39
r = 26.32, post-transient points = 3376; peaks found = 37
r = 26.42, post-transient points = 3104; peaks found = 39
r = 26.52, post-transient points = 2923; peaks found = 41
r = 26.62, post-transient points = 3183; peaks found = 39
r = 26.72, post-transient points = 3181; peaks found = 39
r = 26.82, post-transient points = 3186; peaks found = 40
r = 26.92, post-transient points = 3343; peaks found = 39
r = 27.02, post-transient points = 3309; peaks found = 39
r = 27.12, post-transient points = 3277; peaks found = 39
r = 27.22, post-transient points = 3305; peaks found = 40
r = 27.32, post-transient points = 3337; peaks found = 39
r = 27.42, post-transient points = 3448; peaks found = 39
r = 27.53, post-transient points = 3339; peaks found = 40
r = 27.63, post-transient points = 3267; peaks found = 40
r = 27.73, post-transient points = 3430; peaks found = 39
r = 27.83, post-transient points = 3309; peaks found = 40
r = 27.93, post-transient points = 3372; peaks found = 40
r = 28.03, post-transient points = 3333; peaks found = 40
r = 28.13, post-transient points = 3467; peaks found = 40
r = 28.23, post-transient points = 3179; peaks found = 41
r = 28.33, post-transient points = 3389; peaks found = 40
r = 28.43, post-transient points = 3438; peaks found = 40
r = 28.53, post-transient points = 3361; peaks found = 40
r = 28.63, post-transient points = 3372; peaks found = 40
r = 28.73, post-transient points = 3462; peaks found = 41
r = 28.83, post-transient points = 3247; peaks found = 41
r = 28.93, post-transient points = 3353; peaks found = 41
r = 29.03, post-transient points = 3462; peaks found = 40
r = 29.13, post-transient points = 3451; peaks found = 41
r = 29.23, post-transient points = 3440; peaks found = 41
```

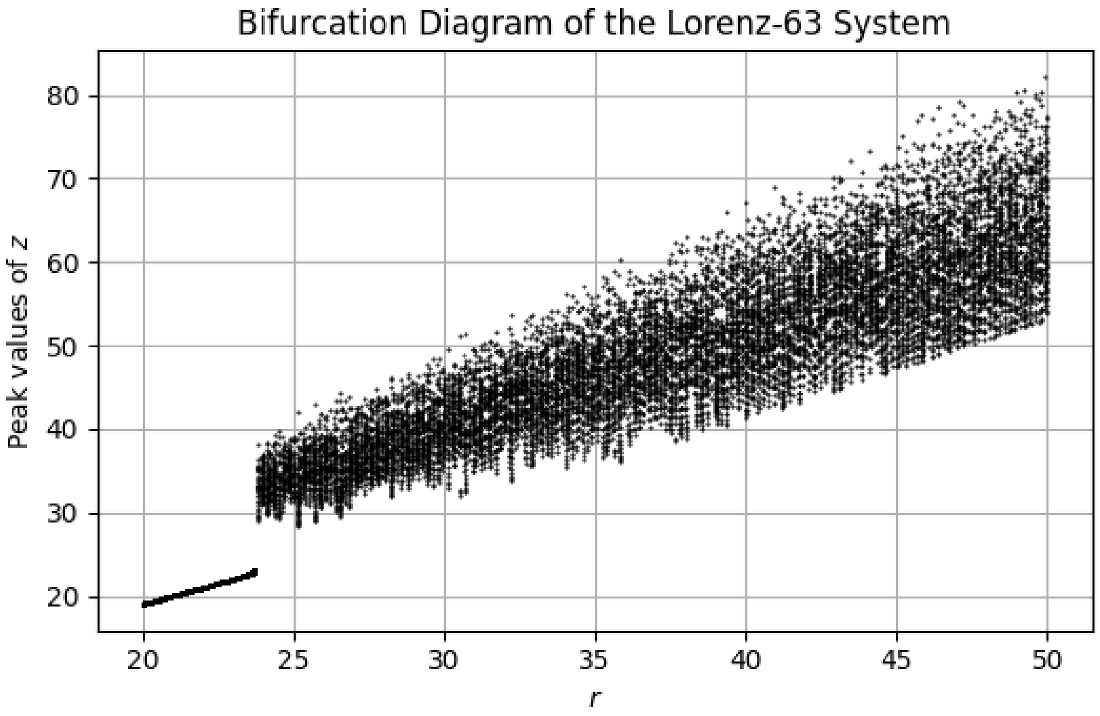
```
r = 29.33, post-transient points = 3431; peaks found = 41
r = 29.43, post-transient points = 3239; peaks found = 43
r = 29.53, post-transient points = 3385; peaks found = 42
r = 29.63, post-transient points = 3442; peaks found = 42
r = 29.73, post-transient points = 3462; peaks found = 42
r = 29.83, post-transient points = 3375; peaks found = 42
r = 29.93, post-transient points = 3400; peaks found = 41
r = 30.03, post-transient points = 3538; peaks found = 41
r = 30.13, post-transient points = 3409; peaks found = 42
r = 30.23, post-transient points = 3551; peaks found = 41
r = 30.33, post-transient points = 3440; peaks found = 42
r = 30.43, post-transient points = 3591; peaks found = 42
r = 30.54, post-transient points = 3504; peaks found = 42
r = 30.64, post-transient points = 3421; peaks found = 42
r = 30.74, post-transient points = 3221; peaks found = 43
r = 30.84, post-transient points = 3489; peaks found = 42
r = 30.94, post-transient points = 3509; peaks found = 43
r = 31.04, post-transient points = 3325; peaks found = 43
r = 31.14, post-transient points = 3534; peaks found = 43
r = 31.24, post-transient points = 3457; peaks found = 43
r = 31.34, post-transient points = 3466; peaks found = 43
r = 31.44, post-transient points = 3535; peaks found = 43
r = 31.54, post-transient points = 3462; peaks found = 43
r = 31.64, post-transient points = 3631; peaks found = 43
r = 31.74, post-transient points = 3404; peaks found = 43
r = 31.84, post-transient points = 3513; peaks found = 43
r = 31.94, post-transient points = 3646; peaks found = 43
r = 32.04, post-transient points = 3708; peaks found = 43
r = 32.14, post-transient points = 3472; peaks found = 44
r = 32.24, post-transient points = 3400; peaks found = 44
r = 32.34, post-transient points = 3624; peaks found = 43
r = 32.44, post-transient points = 3607; peaks found = 44
r = 32.54, post-transient points = 3711; peaks found = 43
r = 32.64, post-transient points = 3638; peaks found = 44
r = 32.74, post-transient points = 3548; peaks found = 44
r = 32.84, post-transient points = 3558; peaks found = 44
r = 32.94, post-transient points = 3399; peaks found = 45
r = 33.04, post-transient points = 3517; peaks found = 45
r = 33.14, post-transient points = 3622; peaks found = 44
r = 33.24, post-transient points = 3530; peaks found = 45
r = 33.34, post-transient points = 3654; peaks found = 44
r = 33.44, post-transient points = 3506; peaks found = 45
r = 33.55, post-transient points = 3447; peaks found = 45
r = 33.65, post-transient points = 3601; peaks found = 45
r = 33.75, post-transient points = 3679; peaks found = 45
r = 33.85, post-transient points = 3707; peaks found = 45
r = 33.95, post-transient points = 3616; peaks found = 45
r = 34.05, post-transient points = 3484; peaks found = 46
```

```
r = 34.15, post-transient points = 3577; peaks found = 46
r = 34.25, post-transient points = 3586; peaks found = 46
r = 34.35, post-transient points = 3680; peaks found = 46
r = 34.45, post-transient points = 3662; peaks found = 45
r = 34.55, post-transient points = 3668; peaks found = 45
r = 34.65, post-transient points = 3593; peaks found = 45
r = 34.75, post-transient points = 3528; peaks found = 46
r = 34.85, post-transient points = 3567; peaks found = 46
r = 34.95, post-transient points = 3680; peaks found = 45
r = 35.05, post-transient points = 3659; peaks found = 46
r = 35.15, post-transient points = 3738; peaks found = 45
r = 35.25, post-transient points = 3453; peaks found = 47
r = 35.35, post-transient points = 3469; peaks found = 47
r = 35.45, post-transient points = 3525; peaks found = 46
r = 35.55, post-transient points = 3702; peaks found = 47
r = 35.65, post-transient points = 3629; peaks found = 46
r = 35.75, post-transient points = 3575; peaks found = 47
r = 35.85, post-transient points = 3461; peaks found = 48
r = 35.95, post-transient points = 3669; peaks found = 46
r = 36.05, post-transient points = 3787; peaks found = 46
r = 36.15, post-transient points = 3671; peaks found = 47
r = 36.25, post-transient points = 3761; peaks found = 47
r = 36.35, post-transient points = 3564; peaks found = 48
r = 36.45, post-transient points = 3614; peaks found = 48
r = 36.56, post-transient points = 3692; peaks found = 47
r = 36.66, post-transient points = 3853; peaks found = 46
r = 36.76, post-transient points = 3754; peaks found = 47
r = 36.86, post-transient points = 3748; peaks found = 47
r = 36.96, post-transient points = 3733; peaks found = 47
r = 37.06, post-transient points = 3803; peaks found = 48
r = 37.16, post-transient points = 3661; peaks found = 47
r = 37.26, post-transient points = 3736; peaks found = 48
r = 37.36, post-transient points = 3720; peaks found = 48
r = 37.46, post-transient points = 3714; peaks found = 48
r = 37.56, post-transient points = 3575; peaks found = 49
r = 37.66, post-transient points = 3639; peaks found = 49
r = 37.76, post-transient points = 3764; peaks found = 48
r = 37.86, post-transient points = 3592; peaks found = 49
r = 37.96, post-transient points = 3745; peaks found = 48
r = 38.06, post-transient points = 3639; peaks found = 49
r = 38.16, post-transient points = 3900; peaks found = 47
r = 38.26, post-transient points = 3914; peaks found = 48
r = 38.36, post-transient points = 3641; peaks found = 49
r = 38.46, post-transient points = 3745; peaks found = 49
r = 38.56, post-transient points = 3615; peaks found = 49
r = 38.66, post-transient points = 3778; peaks found = 49
r = 38.76, post-transient points = 3724; peaks found = 49
r = 38.86, post-transient points = 3905; peaks found = 48
```

```
r = 38.96, post-transient points = 3604; peaks found = 50
r = 39.06, post-transient points = 3500; peaks found = 50
r = 39.16, post-transient points = 3699; peaks found = 50
r = 39.26, post-transient points = 3712; peaks found = 49
r = 39.36, post-transient points = 3646; peaks found = 50
r = 39.46, post-transient points = 3814; peaks found = 49
r = 39.57, post-transient points = 3780; peaks found = 50
r = 39.67, post-transient points = 3851; peaks found = 50
r = 39.77, post-transient points = 3736; peaks found = 50
r = 39.87, post-transient points = 3833; peaks found = 50
r = 39.97, post-transient points = 3757; peaks found = 50
r = 40.07, post-transient points = 3756; peaks found = 50
r = 40.17, post-transient points = 3775; peaks found = 50
r = 40.27, post-transient points = 3724; peaks found = 50
r = 40.37, post-transient points = 3857; peaks found = 50
r = 40.47, post-transient points = 3801; peaks found = 50
r = 40.57, post-transient points = 3688; peaks found = 51
r = 40.67, post-transient points = 3940; peaks found = 50
r = 40.77, post-transient points = 3720; peaks found = 51
r = 40.87, post-transient points = 3979; peaks found = 50
r = 40.97, post-transient points = 3669; peaks found = 51
r = 41.07, post-transient points = 3876; peaks found = 51
r = 41.17, post-transient points = 3665; peaks found = 52
r = 41.27, post-transient points = 3801; peaks found = 51
r = 41.37, post-transient points = 3655; peaks found = 51
r = 41.47, post-transient points = 3716; peaks found = 51
r = 41.57, post-transient points = 3931; peaks found = 51
r = 41.67, post-transient points = 3949; peaks found = 50
r = 41.77, post-transient points = 3779; peaks found = 52
r = 41.87, post-transient points = 3914; peaks found = 51
r = 41.97, post-transient points = 3909; peaks found = 51
r = 42.07, post-transient points = 3956; peaks found = 51
r = 42.17, post-transient points = 4024; peaks found = 51
r = 42.27, post-transient points = 3580; peaks found = 52
r = 42.37, post-transient points = 4031; peaks found = 51
r = 42.47, post-transient points = 3907; peaks found = 52
r = 42.58, post-transient points = 4002; peaks found = 51
r = 42.68, post-transient points = 3931; peaks found = 52
r = 42.78, post-transient points = 4094; peaks found = 51
r = 42.88, post-transient points = 3775; peaks found = 52
r = 42.98, post-transient points = 3978; peaks found = 52
r = 43.08, post-transient points = 3891; peaks found = 52
r = 43.18, post-transient points = 3900; peaks found = 53
r = 43.28, post-transient points = 3967; peaks found = 52
r = 43.38, post-transient points = 4028; peaks found = 51
r = 43.48, post-transient points = 3963; peaks found = 52
r = 43.58, post-transient points = 4118; peaks found = 52
r = 43.68, post-transient points = 3928; peaks found = 53
```

```
r = 43.78, post-transient points = 4012; peaks found = 52
r = 43.88, post-transient points = 3862; peaks found = 53
r = 43.98, post-transient points = 4096; peaks found = 52
r = 44.08, post-transient points = 4040; peaks found = 52
r = 44.18, post-transient points = 4071; peaks found = 53
r = 44.28, post-transient points = 4037; peaks found = 52
r = 44.38, post-transient points = 4017; peaks found = 53
r = 44.48, post-transient points = 3821; peaks found = 53
r = 44.58, post-transient points = 3691; peaks found = 54
r = 44.68, post-transient points = 4067; peaks found = 53
r = 44.78, post-transient points = 4037; peaks found = 53
r = 44.88, post-transient points = 4081; peaks found = 53
r = 44.98, post-transient points = 3976; peaks found = 53
r = 45.08, post-transient points = 3968; peaks found = 53
r = 45.18, post-transient points = 3945; peaks found = 53
r = 45.28, post-transient points = 3992; peaks found = 54
r = 45.38, post-transient points = 3927; peaks found = 54
r = 45.48, post-transient points = 4091; peaks found = 53
r = 45.59, post-transient points = 3984; peaks found = 54
r = 45.69, post-transient points = 3949; peaks found = 54
r = 45.79, post-transient points = 4027; peaks found = 54
r = 45.89, post-transient points = 3877; peaks found = 54
r = 45.99, post-transient points = 4078; peaks found = 54
r = 46.09, post-transient points = 4075; peaks found = 54
r = 46.19, post-transient points = 4051; peaks found = 54
r = 46.29, post-transient points = 4086; peaks found = 55
r = 46.39, post-transient points = 4034; peaks found = 54
r = 46.49, post-transient points = 4077; peaks found = 55
r = 46.59, post-transient points = 4040; peaks found = 55
r = 46.69, post-transient points = 3958; peaks found = 55
r = 46.79, post-transient points = 4061; peaks found = 55
r = 46.89, post-transient points = 4068; peaks found = 54
r = 46.99, post-transient points = 3947; peaks found = 55
r = 47.09, post-transient points = 3964; peaks found = 55
r = 47.19, post-transient points = 4050; peaks found = 55
r = 47.29, post-transient points = 4058; peaks found = 56
r = 47.39, post-transient points = 4174; peaks found = 55
r = 47.49, post-transient points = 4020; peaks found = 55
r = 47.59, post-transient points = 4073; peaks found = 56
r = 47.69, post-transient points = 4052; peaks found = 56
r = 47.79, post-transient points = 4109; peaks found = 55
r = 47.89, post-transient points = 4032; peaks found = 56
r = 47.99, post-transient points = 4233; peaks found = 55
r = 48.09, post-transient points = 4063; peaks found = 55
r = 48.19, post-transient points = 4165; peaks found = 55
r = 48.29, post-transient points = 4000; peaks found = 57
r = 48.39, post-transient points = 4137; peaks found = 55
r = 48.49, post-transient points = 4210; peaks found = 56
```

```
r = 48.60, post-transient points = 4043; peaks found = 56
r = 48.70, post-transient points = 4225; peaks found = 55
r = 48.80, post-transient points = 4134; peaks found = 56
r = 48.90, post-transient points = 4154; peaks found = 56
r = 49.00, post-transient points = 4173; peaks found = 57
r = 49.10, post-transient points = 4124; peaks found = 56
r = 49.20, post-transient points = 4075; peaks found = 56
r = 49.30, post-transient points = 4119; peaks found = 56
r = 49.40, post-transient points = 4111; peaks found = 56
r = 49.50, post-transient points = 4237; peaks found = 56
r = 49.60, post-transient points = 4221; peaks found = 56
r = 49.70, post-transient points = 4271; peaks found = 56
r = 49.80, post-transient points = 4237; peaks found = 56
r = 49.90, post-transient points = 4169; peaks found = 57
r = 50.00, post-transient points = 4201; peaks found = 56
```



1.6.1 Discussion of Results

The bifurcation diagram above shows the evolution of the peak values of (z) as (r) increases: - At lower values of (r), the system settles to a single peak (steady or periodic behavior). - As (r) crosses a threshold, you can observe a splitting in the peak values, which indicates the onset of bifurcation. - Further increases in (r) yield multiple peak values spread over a range, revealing the presence of chaotic dynamics.

This analysis provides insight into how sensitive the Lorenz system is to parameter changes—a key consideration when designing robust numerical solvers and interpreting chaotic behavior in engineering applications.

1.7 7. Largest Lyapunov Exponent: Benettin Algorithm

The Lyapunov exponent quantifies the average rate of divergence of nearby trajectories in a dynamical system. A positive value indicates that the system is chaotic (sensitive dependence on initial conditions). We focus on computing the **largest Lyapunov exponent (LLE)**.

1.7.1 Benettin Algorithm Overview

For a small perturbation $\delta(0)$ around an initial condition $\mathbf{u}(0)$, the algorithm is: 1. **Initialization:** Start with $\mathbf{u}_0$ and a perturbed state $\mathbf{u}_0 + \delta_0$, where $\|\delta_0\|$ is very small. 2. **Integration Step:** Integrate both trajectories for a short time interval Δt. 3. **Deviation Update:**
Compute the difference:

$$\delta = \mathbf{u}(t+\Delta t)_{\text{perturbed}} - \mathbf{u}(t+\Delta t)$$

4. **Normalization & Accumulation:**
Record the growth factor $\log\left(\frac{\|\delta\|}{\|\delta_0\|}\right)$, then renormalize δ back to the initial magnitude. 5. **Iteration:** Repeat for a total integration time T. 6. **Estimation:**
The LLE is estimated as:

$$\lambda \approx \frac{1}{T}\sum_i \log\left(\frac{\|\delta_i\|}{\|\delta_0\|}\right).$$

Below, you'll find a Python implementation of this algorithm for the Lorenz-63 system.

```
[12]: def compute_lyapunov(u0=(1.0, 1.0, 1.0), r=28.0, T=100.0, dt=0.1, delta0=1e-8):
          """
          Compute the largest Lyapunov exponent for the Lorenz-63 system using the␣
       ↪Benettin algorithm.

          Parameters:
              u0: tuple or array, initial condition.
              r: float, the parameter 'r' in the Lorenz system.
              T: float, total integration time.
              dt: float, time interval for each renormalization step.
              delta0: float, initial magnitude of the perturbation.

          Returns:
              lyapunov: float, the computed largest Lyapunov exponent.
          """
          N = int(T / dt)
          sum_log = 0.0

          # Convert initial condition to numpy array
          u = np.array(u0, dtype=float)
          # Initialize a small perturbation (you can choose the direction)
          delta = np.array([delta0, 0.0, 0.0], dtype=float)
```

```
    for i in range(N):
        # Integrate the original trajectory over dt
        sol1 = solve_lorenz(u0=u, r=r, t_end=dt, rtol=1e-8, atol=1e-10)
        # Integrate the perturbed trajectory over dt
        sol2 = solve_lorenz(u0=u + delta, r=r, t_end=dt, rtol=1e-8, atol=1e-10)

        # Extract the states at t+dt
        u_new = sol1.y[:, -1]
        u2_new = sol2.y[:, -1]

        # Compute the difference between trajectories
        d = u2_new - u_new
        norm_d = np.linalg.norm(d)

        # Accumulate the log of the divergence factor
        sum_log += np.log(norm_d / delta0)

        # Renormalize the difference to have the initial magnitude delta0
        delta = (delta0 / norm_d) * d

        # Update the base trajectory for the next interval
        u = u_new

    # The largest Lyapunov exponent is the time-averaged logarithmic growth rate
    lyapunov = sum_log / (N * dt)
    return lyapunov

# Compute and print the largest Lyapunov exponent for the Lorenz system
lambda_max = compute_lyapunov(u0=(1.0, 1.0, 1.0), r=28.0, T=100.0, dt=0.1,␣
 ↪delta0=1e-8)
print("Largest Lyapunov Exponent for r = 28.0: ", lambda_max)
```

```
Largest Lyapunov Exponent for r = 28.0:  0.812598508627019
```

1.7.2 Discussion of the Largest Lyapunov Exponent

The computed largest Lyapunov exponent, λ_{max}, quantifies the average rate at which nearby trajectories diverge: - **Positive λ_{max}:** Indicates chaos (sensitive dependence on initial conditions). - **Magnitude:** The reciprocal of λ_{max} gives an estimate of the predictability time scale (the "predictability horizon").

For the Lorenz-63 system at $r = 28$, we expect a positive Lyapunov exponent typically in the range of 0.8 to 1.2 (in appropriate units), confirming its chaotic behavior.

Adjusting the parameters (total integration time T, step time dt, or even the perturbation magnitude δ_0) can refine the accuracy of the estimation.

1.8 Saving Data as CSV Files

In order to facilitate further analysis and ensure reproducibility, we save the computed data to CSV files:

- **Bifurcation Data:**
  We have collected two lists during the bifurcation study:
  - `r_plot`: a list of (r) parameter values, and

  - `z_peak_plot`: the corresponding (z) peak values extracted from the time series.
    These are combined into a DataFrame and saved as `bifurcation_data.csv`.
- **Lyapunov Data:**
  After computing the largest Lyapunov exponent (stored in `lambda_max`) for a given (r) (here, (r = 28.0)), we store that information in another DataFrame and save it as `lyapunov_data.csv`.

Using pandas makes it easy to export the results, which you can then load into any data analysis or plotting tool for further investigation.

[13]:
```python
import pandas as pd

# Save bifurcation data (r values and corresponding z peaks)
# 'r_plot' and 'z_peak_plot' should have been generated in your bifurcation␣
↪loop.
df_bif = pd.DataFrame({'r': r_plot, 'z_peak': z_peak_plot})
df_bif.to_csv("bifurcation_data.csv", index=False)
print("Bifurcation data saved to 'bifurcation_data.csv'.")

# Save Lyapunov exponent data
# 'lambda_max' holds the computed largest Lyapunov exponent for a given r (e.g.
↪, r=28).
df_lyap = pd.DataFrame({'r': [28.0], 'lambda_max': [lambda_max]})
df_lyap.to_csv("lyapunov_data.csv", index=False)
print("Lyapunov data saved to 'lyapunov_data.csv'.")
```

```
Bifurcation data saved to 'bifurcation_data.csv'.
Lyapunov data saved to 'lyapunov_data.csv'.
```

1.9 8. Analysis and Discussion

In this phase, we analyze and interpret the data computed in previous phases.

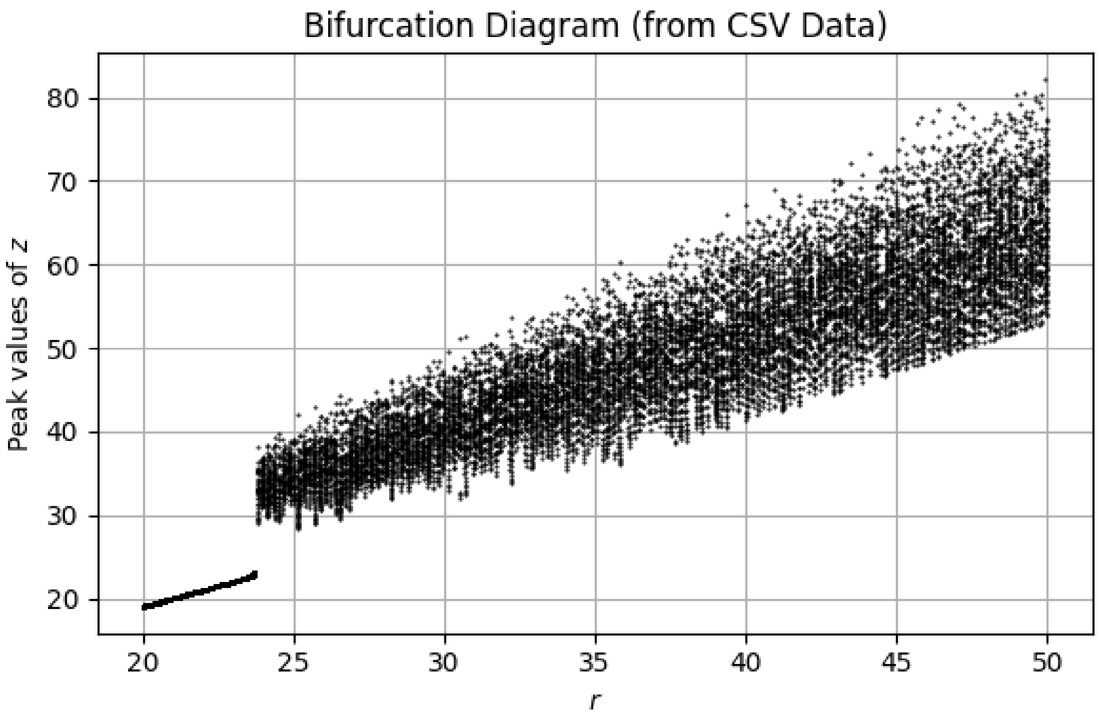
- **Bifurcation Diagram:**
  We re-load the data (stored as `bifurcation_data.csv`) and re-plot it to verify that our simulation of the Lorenz system captured the evolution of the peak values of (z) as (r) varied.

- **Lyapunov Exponents Sweep:**
  We then perform a parameter sweep over (r) to compute the largest Lyapunov exponent. A plot of the Lyapunov exponent versus (r) helps to identify transitions between regular and chaotic behavior.

The combined analysis helps us connect the bifurcation structure with the predictability (via the Lyapunov exponent) of the system.

```
[14]: import pandas as pd

      # Load the saved bifurcation data
      df_bif = pd.read_csv("bifurcation_data.csv")
      print("Bifurcation data preview:")
      print(df_bif.head())

      # Plot the bifurcation diagram
      plt.figure(figsize=(6,4))
      plt.plot(df_bif['r'], df_bif['z_peak'], 'k.', markersize=1)
      plt.xlabel('$r$')
      plt.ylabel('Peak values of $z$')
      plt.title('Bifurcation Diagram (from CSV Data)')
      plt.grid(True)
      plt.tight_layout()
      plt.show()
```

```
Bifurcation data preview:
      r     z_peak
0  20.0  19.000002
1  20.0  19.000002
2  20.0  19.000002
3  20.0  19.000002
4  20.0  19.000002
```



1.10 Lyapunov Exponent Analysis

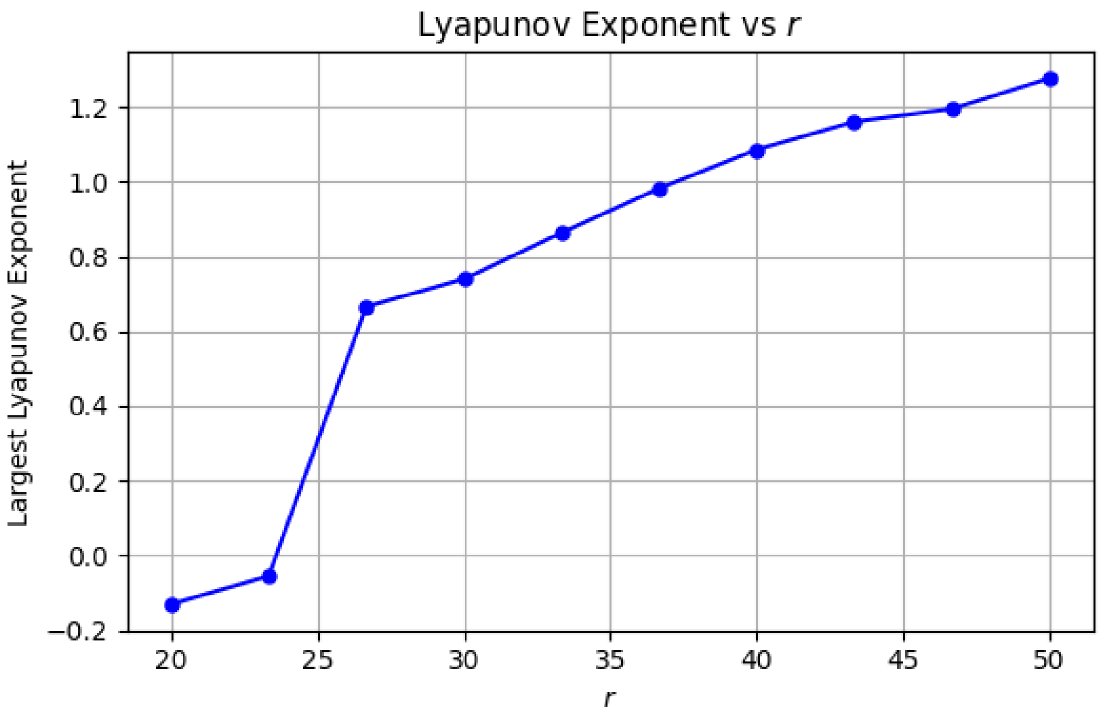
Next, we compute the largest Lyapunov exponent over a range of (r) values. The Lyapunov exponent indicates how fast nearby trajectories diverge—positive values mean the system is chaotic.

For this sweep: - We choose a small number of (r) values (for demonstration) from 20 to 50. - At each (r), we use our `compute_lyapunov` function with a shorter integration time (to speed up the computation during analysis). - The resulting plot of the largest Lyapunov exponent versus (r) will help us identify the threshold where the system becomes chaotic.

```
[15]: # Define a range of r values for the Lyapunov sweep (adjust number of points as␣
       ↪needed)
      r_range = np.linspace(20, 50, 10)  # you may increase the number of points for␣
       ↪higher resolution
      lyap_vals = []

      for r in r_range:
          # Here we reduce total time for faster computation; adjust T and dt for␣
       ↪more accuracy
          lam = compute_lyapunov(u0=(1.0, 1.0, 1.0), r=r, T=50.0, dt=0.1, delta0=1e-8)
          lyap_vals.append(lam)
          print(f"r = {r:.2f}, Largest Lyapunov Exponent = {lam:.4f}")

      # Save the Lyapunov sweep data (optional)
      df_lyap = pd.DataFrame({'r': r_range, 'lambda_max': lyap_vals})
      df_lyap.to_csv("lyapunov_sweep_data.csv", index=False)
      print("Lyapunov sweep data saved to 'lyapunov_sweep_data.csv'.")

      # Plot the results
      plt.figure(figsize=(6,4))
      plt.plot(r_range, lyap_vals, 'bo-', markersize=5)
      plt.xlabel('$r$')
      plt.ylabel('Largest Lyapunov Exponent')
      plt.title('Lyapunov Exponent vs $r$')
      plt.grid(True)
      plt.tight_layout()
      plt.show()
```

```
r = 20.00, Largest Lyapunov Exponent = -0.1300
r = 23.33, Largest Lyapunov Exponent = -0.0549
r = 26.67, Largest Lyapunov Exponent = 0.6656
r = 30.00, Largest Lyapunov Exponent = 0.7391
r = 33.33, Largest Lyapunov Exponent = 0.8631
r = 36.67, Largest Lyapunov Exponent = 0.9816
r = 40.00, Largest Lyapunov Exponent = 1.0854
r = 43.33, Largest Lyapunov Exponent = 1.1600
```

```
r = 46.67, Largest Lyapunov Exponent = 1.1944
r = 50.00, Largest Lyapunov Exponent = 1.2761
Lyapunov sweep data saved to 'lyapunov_sweep_data.csv'.
```



1.11 Discussion and Interpretation

- **Bifurcation Diagram:**
  The re-plotted bifurcation data shows the evolution of the system's peak (z) values as (r) increases. Notice how low (r) values yield a single (or tightly grouped) peak, while higher (r) values produce a scattered cloud of peak values, revealing chaotic behavior.

- **Lyapunov Exponent Sweep:**
  The plot of the largest Lyapunov exponent versus (r) typically shows that, for certain (r) values, the exponent becomes positive. A positive value indicates chaos, and its magnitude gives an idea about the divergence rate of nearby trajectories.

  For instance, for (r) values where the Lyapunov exponent exceeds zero, the system transitions into chaos, explaining the broad spread in the bifurcation diagram.

These plots provide a combined view that links the qualitative structure of the attractor (via peaks) with the quantitative measure of chaos (via the Lyapunov exponent).

1.11.1 Lorenz-63 System Animation (GIF)

To visualize the temporal evolution of the Lorenz-63 system, I animated the trajectory traced by the (x, y, z) components over time. This animation provides an intuitive representation of how the strange attractor unfolds dynamically in 3D space.

The animation is constructed by: - Solving the Lorenz-63 system using `solve_ivp` over a fine time grid. - Plotting the solution frame-by-frame to simulate motion through phase space. - Using Matplotlib's `FuncAnimation` and `PillowWriter` to save the animation as a `.gif`.

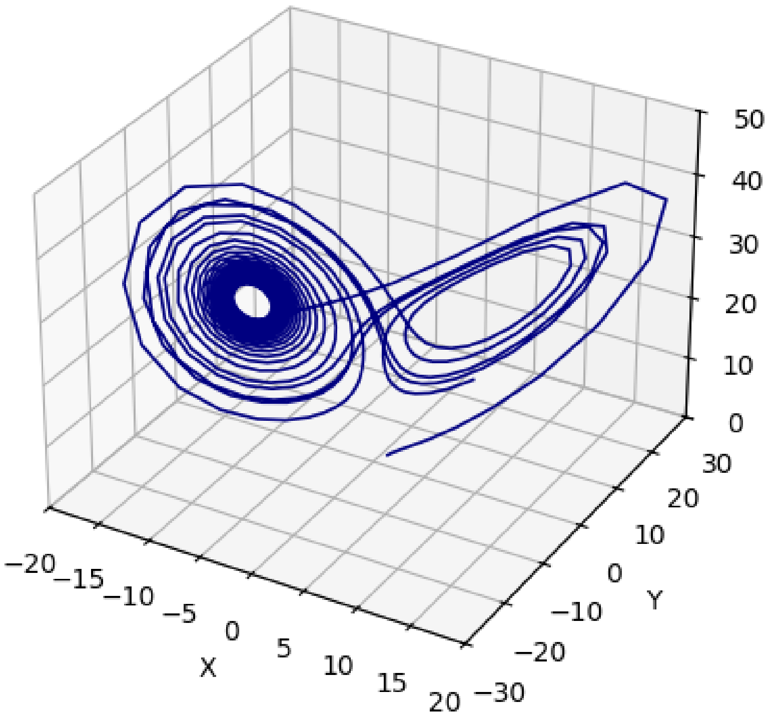
This visualization complements static plots by illustrating the flow of the trajectory, helping me appreciate the system's sensitivity to initial conditions and the nature of chaotic attractors.

[20]:
```python
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
from scipy.integrate import solve_ivp
from matplotlib import animation, rc
from IPython.display import HTML

# Raise embed limit (keep just above our GIF size)
rc('animation', embed_limit=52)

# Lorenz-63 system
def lorenz_rhs(t, u, sigma=10, beta=8/3, r=28):
    x, y, z = u
    return [sigma * (y - x), x * (r - z) - y, x * y - beta * z]

# Shorter simulation, fewer frames = smaller GIF
t_span = (0, 20)
t_eval = np.linspace(*t_span, 600)
u0 = [1.0, 1.0, 1.0]
sol = solve_ivp(lorenz_rhs, t_span, u0, t_eval=t_eval)

# Plot setup
fig = plt.figure(figsize=(7, 5))
ax = fig.add_subplot(111, projection='3d')
ax.set_xlim((-20, 20))
ax.set_ylim((-30, 30))
ax.set_zlim((0, 50))
ax.set_xlabel("X")
ax.set_ylabel("Y")
ax.set_zlabel("Z")

line, = ax.plot([], [], [], lw=1.2, color='navy')

# Init
def init():
    line.set_data([], [])
    line.set_3d_properties([])
    return line,
# Animate
def animate(i):
    line.set_data(sol.y[0][:i], sol.y[1][:i])
    line.set_3d_properties(sol.y[2][:i])
    return line,

# Create animation object
ani = animation.FuncAnimation(fig, animate, init_func=init,
                              frames=len(t_eval), interval=20, blit=True)

# Display in notebook
display(HTML(ani.to_jshtml()))

# Save as GIF
from matplotlib.animation import PillowWriter
ani.save("lorenz63.gif", writer=PillowWriter(fps=15))
print("GIF saved as lorenz63.gif")
```

```
Animation size has reached 54626527 bytes, exceeding the limit of 54525952.0. If
you're sure you want a larger animation embedded, set the animation.embed_limit
rc parameter to a larger value (in MB). This and further frames will be dropped.
```

```
<IPython.core.display.HTML object>
```

```
GIF saved as lorenz63.gif
```



```
[ ]:
```