4.3 Lyapunov exponents for the Lorenz model

The three-dimensional Lorenz flow is given by

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

a) Write a code that numerically computes the Lyapunov exponents for the Lorenz attractor. You may find the following hints helpful:

Use the “cooking recipe” given in the lecture notes (Section 11.5) to calculate Lyapunov exponents. Following this method, you iterate the discretized equation for the deformation matrix. At each time step you evaluate the Jacobian along your solution trajectory $(x(t), y(t), z(t))$. The Lyapunov exponents are obtained from the diagonal elements of the R-matrix using the QR-decomposition method.

You may want to use the built-in function “QRDecomposition[]” in Mathematica. Read the documentation of that function carefully. In particular, note that (at least in Mathematica Version 13 and below) it returns for a given matrix M the pair $\{Q^T, R\}$.

For the parameter values studied by Lorenz and discussed in the exercise “Introduction to the Lorenz model”, $\sigma = 10$, b= 8 / 3, r = 28, the Lyapunov exponents are approximately $\lambda_1 \approx 0.91$, $\lambda_2 \approx 0$, $\lambda_3 \approx -14, 57$.

In the exercise “Introduction to the Lorenz model” you have derived an expression for the sum of the Lyapunov exponents, $\lambda_1 + \lambda_2 + \lambda_3$, for general parameter values. This expression is numerically more stable than the results for the individual Lyapunov exponents and it is very useful for checking that your code works properly.

Once you have a working code, do test runs by solving the equations for a couple of hundred time units.

Discard the initial transient of the solution as you did when plotting the Lorenz attractor in the exercise “Introduction to the Lorenz model”.

When your test runs work well for the sum of exponents and ok for individual exponents, you can try to get better results by iterating for long times with short time

step
$dt \approx 10^{-3}$.

High precision for the individual Lyapunov exponents is hard to achieve and you will probably have to be satisfied with 2 – 3 digits of accuracy.

Read the hints!
Also section 11.5, 11.7

```
ClearAll["Global`*"]

(* Lorenz variables *)
σ = 10;
r = 28;
b = 8/3;

(* Differential equation system *)
Equation1 = x'[t] == σ*(y[t] - x[t]);
Equation2 = y'[t] == r*x[t] - y[t] - x[t]*z[t];
Equation3 = z'[t] == x[t]*y[t] - b*z[t];
System = {Equation1, Equation2, Equation3};

(* Fixed points of the differential equation system *)
FixedPoint1 = {0, 0, 0};

eta = 0.001;
t0 = 0;
tMax = 1000;
stepSize = 0.001;

(* Starting point of the trajectory (distance from fixed points handled via eta) *)
StartingPoint = {x[0] == FixedPoint1[[1]] + eta, y[0] == FixedPoint1[[2]] + eta, z[0] == Fix

(* Solution using NDSolve with specified step size *)
Solution = NDSolve[{System, StartingPoint}, {x, y, z}, {t, t0, tMax}];

xFunction = x /. First[Solution];
yFunction = y /. First[Solution];
zFunction = z /. First[Solution];

(*Initialize parameters*)
lambda1 = 0;
lambda2 = 0;
lambda3 = 0;
```

```
Qold = IdentityMatrix[3];

(* Preallocate lambdaValues matrix *)
numIterations = Ceiling[(tMax - 0.001)/stepSize] + 1;
lambdaValues = ConstantArray[0, {numIterations, 4}];

i = 1;
(* Create the loop with Monitor *)
Monitor[
    For[t = 0.001, t ≤ tMax, t += stepSize,
        (* Jacobian matrix *)
        jacobianMatrix = {
            {-σ, σ, 0},
            {r - zFunction[t], -1, -xFunction[t]},
            {yFunction[t], xFunction[t], -b}
        };
        (* M matrix *)
        M = IdentityMatrix[3] + jacobianMatrix * stepSize;
        (* QR decomposition *)
        valuesQR = QRDecomposition[M . Qold];
        Qold = Transpose[valuesQR⟦1⟧];
        Rold = valuesQR⟦2⟧;
        (* Update Lyapunov exponents *)
        lambda1 += Log[Abs[Rold⟦1, 1⟧]];
        lambda2 += Log[Abs[Rold⟦2, 2⟧]];
        lambda3 += Log[Abs[Rold⟦3, 3⟧]];
        (* Assign values to lambdaValues matrix *)
        lambdaValues⟦i⟧ = {t, lambda1/t, lambda2/t, lambda3/t};
        i++;
    ],
    ProgressIndicator[t, {0.001, tMax}]
]

(* Output results *)
{lambda1/tMax, lambda2/tMax, lambda3/tMax}
```

Out[ ]=

```
{0.942875, 0.029703, -14.676}
```

b) Make a plot that shows the evolution of $\lambda_1(t)$, $\lambda_2(t)$ and $\lambda_3(t)$ as a function of time t, using a logarithmic scaling of the t-axis. First choose $t \in [1, 10^3]$. On the basis of this plot, decide whether or not your Lyapunov exponents are converged. Let the dynamics run for longer time if you don’t see convergence.

Upload your final plot as .pdf or .png.

In[ ]:=
```
(* Filter values where t ≥ 1 *)
filteredLambdaValues = Select[lambdaValues, First[#] ≥ 1 &];

(* Extracting lambda values *)
lambda1Values = filteredLambdaValues⟦All, {1, 2}⟧;  (* t and lambda1/t *)
lambda2Values = filteredLambdaValues⟦All, {1, 3}⟧;  (* t and lambda2/t *)
lambda3Values = filteredLambdaValues⟦All, {1, 4}⟧;  (* t and lambda3/t *)

(* Plotting *)
ListLogLinearPlot[
    {lambda1Values, lambda2Values, lambda3Values},
    PlotLegends → {"λ1", "λ2", "λ3"},
    AxesLabel → {"Log(t)", "Lambda values"},
    PlotLabel → "For e) Lambda values vs. Log(Time)",
    PlotRange → All,
    PlotStyle → {Red, Green, Blue}
]
```

We now compute the Lyapunov exponents for a parameter set slightly different from that of Lorenz. Set your parameters according to $\sigma = 10$, b = 19/6 and r = 28.

---

c) Make the same plot as in (b) for the new parameter set. On the basis of this plot, decide whether or not your Lyapunov exponents are converged. Let the dynamics run for longer time if you don’t see convergence.

Upload your final plot as .pdf or .png.

```
ClearAll["Global`*"]

(* Lorenz variables *)
σ = 10;
r = 28;
b = 19/6;

(* Differential equation system *)
```

```
Equation1 = x'[t] == σ*(y[t] - x[t]);
Equation2 = y'[t] == r*x[t] - y[t] - x[t]*z[t];
Equation3 = z'[t] == x[t]*y[t] - b*z[t];
System = {Equation1, Equation2, Equation3};

(* Fixed points of the differential equation system *)
FixedPoint1 = {0, 0, 0};

eta = 0.001;
t0 = 0;
tMax = 1000;
stepSize = 0.001;

(* Starting point of the trajectory (distance from fixed points handled via eta) *)
StartingPoint = {x[0] == FixedPoint1⟦1⟧ + eta, y[0] == FixedPoint1⟦2⟧ + eta, z[0] == Fix

(* Solution using NDSolve with specified step size *)
Solution = NDSolve[{System, StartingPoint}, {x, y, z}, {t, t0, tMax}];

xFunction = x /. First[Solution];
yFunction = y /. First[Solution];
zFunction = z /. First[Solution];

(*Initialize parameters*)
lambda1 = 0;
lambda2 = 0;
lambda3 = 0;
Qold = IdentityMatrix[3];

(* Preallocate lambdaValues matrix *)
numIterations = Ceiling[(tMax - 0.001)/stepSize] + 1;
lambdaValues = ConstantArray[0, {numIterations, 4}];

i = 1;
(* Create the loop with Monitor *)
Monitor[
    For[t = 0.001, t ≤ tMax, t += stepSize,
        (* Jacobian matrix *)
        jacobianMatrix = {
            {-σ, σ, 0},
            {r - zFunction[t], -1, -xFunction[t]},
            {yFunction[t], xFunction[t], -b}
        };
        (* M matrix *)
        M = IdentityMatrix[3] + jacobianMatrix * stepSize;
        (* QR decomposition *)
        valuesQR = QRDecomposition[M . Qold];
        Qold = Transpose[valuesQR⟦1⟧];
        Rold = valuesQR⟦2⟧;
        (* Update Lyapunov exponents *)
        lambda1 += Log[Abs[Rold⟦1, 1⟧]];
        lambda2 += Log[Abs[Rold⟦2, 2⟧]];
        lambda3 += Log[Abs[Rold⟦3, 3⟧]];
        (* Assign values to lambdaValues matrix *)
        lambdaValues⟦i⟧ = {t, lambda1/t, lambda2/t, lambda3/t};
        i++;
    ],
    ProgressIndicator[t, {0.001, tMax}]
]

(* Output results *)
{lambda1/tMax, lambda2/tMax, lambda3/tMax}
```

Out[∘]=

```
{0.983782, 0.0318873, -15.2203}
```

In[∘]:=

```
(* Filter values where t ≥ 1 *)
filteredLambdaValues = Select[lambdaValues, First[#] ≥ 1 &];

(* Extracting lambda values *)
lambda1Values = filteredLambdaValues⟦All, {1, 2}⟧;  (* t and lambda1/t *)
lambda2Values = filteredLambdaValues⟦All, {1, 3}⟧;  (* t and lambda2/t *)
lambda3Values = filteredLambdaValues⟦All, {1, 4}⟧;  (* t and lambda3/t *)

(* Plotting *)
ListLogLinearPlot[
    {lambda1Values, lambda2Values, lambda3Values},
    PlotLegends → {"λ1", "λ2", "λ3"},
    AxesLabel → {"Log(t)", "Lambda values"},
    PlotLabel → "For e) Lambda values vs. Log(Time)",
    PlotRange → All,
    PlotStyle → {Red, Green, Blue}
]
```

d) Give your (converged) result for the Lyapunov exponents with the new parameters as the ordered vector [λ_1, λ_2, λ_3] with $\lambda_1 \geq \lambda_1 \geq \lambda_3$. Provide two decimal digits of accuracy.

See output of c).

We change the parameters again, now to very different values. Set your parameters according to σ=16, b=5 and r=330. You may want to plot the attractor for the new

parameters in order to know, what the attracting set looks like now, but you do not need to upload that plot.

---

e) Make the same plot as in (b) and (c) for the third parameter set. On the basis of this plot, decide whether or not your Lyapunov exponents are converged. Let the dynamics run for longer time if you don’t see convergence.

```
ClearAll["Global`*"]

(* Lorenz variables *)
σ = 16;
r = 330;
b = 5;

(* Differential equation system *)
Equation1 = x'[t] == σ*(y[t] - x[t]);
Equation2 = y'[t] == r*x[t] - y[t] - x[t]*z[t];
Equation3 = z'[t] == x[t]*y[t] - b*z[t];
System = {Equation1, Equation2, Equation3};

(* Fixed points of the differential equation system *)
FixedPoint1 = {0, 0, 0};

eta = 0.001;
t0 = 0;
tMax = 1000;
stepSize = 0.0001;

(* Starting point of the trajectory (distance from fixed points handled via eta) *)
StartingPoint = {x[0] == FixedPoint1⟦1⟧ + eta, y[0] == FixedPoint1⟦2⟧ + eta, z[0] == Fix

(* Solution using NDSolve with specified step size *)
Solution = NDSolve[{System, StartingPoint}, {x, y, z}, {t, t0, tMax}];

xFunction = x /. First[Solution];
yFunction = y /. First[Solution];
zFunction = z /. First[Solution];

(*Initialize parameters*)
lambda1 = 0;
lambda2 = 0;
lambda3 = 0;
```

```
Qold = IdentityMatrix[3];

(* Preallocate lambdaValues matrix *)
numIterations = Ceiling[(tMax - 0.001)/stepSize] + 1;
lambdaValues = ConstantArray[0, {numIterations, 4}];

i = 1;
(* Create the loop with Monitor *)
Monitor[
    For[t = 0.001, t ≤ tMax, t += stepSize,
        (* Jacobian matrix *)
        jacobianMatrix = {
            {-σ, σ, 0},
            {r - zFunction[t], -1, -xFunction[t]},
            {yFunction[t], xFunction[t], -b}
        };
        (* M matrix *)
        M = IdentityMatrix[3] + jacobianMatrix * stepSize;
        (* QR decomposition *)
        valuesQR = QRDecomposition[M . Qold];
        Qold = Transpose[valuesQR⟦1⟧];
        Rold = valuesQR⟦2⟧;
        (* Update Lyapunov exponents *)
        lambda1 += Log[Abs[Rold⟦1, 1⟧]];
        lambda2 += Log[Abs[Rold⟦2, 2⟧]];
        lambda3 += Log[Abs[Rold⟦3, 3⟧]];
        (* Assign values to lambdaValues matrix *)
        lambdaValues⟦i⟧ = {t, lambda1/t, lambda2/t, lambda3/t};
        i++;
    ],
    ProgressIndicator[t, {0.001, tMax}]
]

(* Output results *)
{lambda1/tMax, lambda2/tMax, lambda3/tMax}
```

Out[31]=

```
{0.123096, -10.9908, -11.0184}
```

In[64]:=
```
filteredLambdaValues = Select[lambdaValues, First[#] ≥ 1 &];

(* Extract every 100th value starting from the first *)
lambda1ValuesEvery10th = filteredLambdaValues⟦1 ;; -1 ;; 100, {1, 2}⟧;  (* t and lambd
lambda2ValuesEvery10th = filteredLambdaValues⟦1 ;; -1 ;; 100, {1, 3}⟧;  (* t and lambd
lambda3ValuesEvery10th = filteredLambdaValues⟦1 ;; -1 ;; 100, {1, 4}⟧;  (* t and lambd

(* Plotting *)
ListLogLinearPlot[
    {lambda1ValuesEvery10th, lambda2ValuesEvery10th, lambda3ValuesEvery10th},
    PlotLegends → {"λ1", "λ2", "λ3"},
    AxesLabel → {"Log(t)", "Lambda values"},
    PlotLabel → "For e) Lambda values vs. Log(Time)",
    PlotRange → All,
    PlotStyle → {Red, Green, Blue}
]
```

f) Give your (converged) result for the Lyapunov exponents with the new parameters as the ordered vector [λ_1, λ_2, λ_3] with $\lambda_1 \geq \lambda_1 \geq \lambda_3$. Provide two decimal digits of accuracy.

See output of e).