Rössler attractor

continuous deterministic chaos

Rössler system

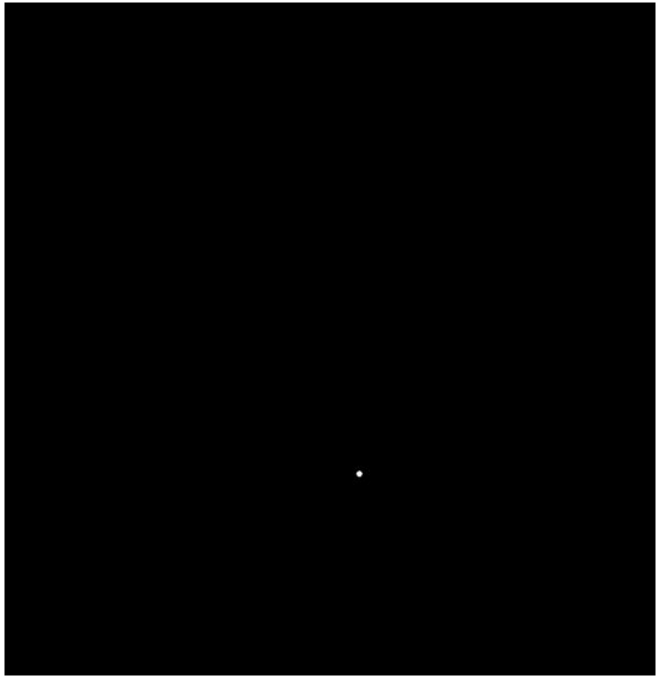
$$\begin{cases} \dot{x} = -y - z \\ \dot{y} = x + ay \\ \dot{z} = b + z(x - c) \end{cases}$$

$\{x, y, z\}$ system state

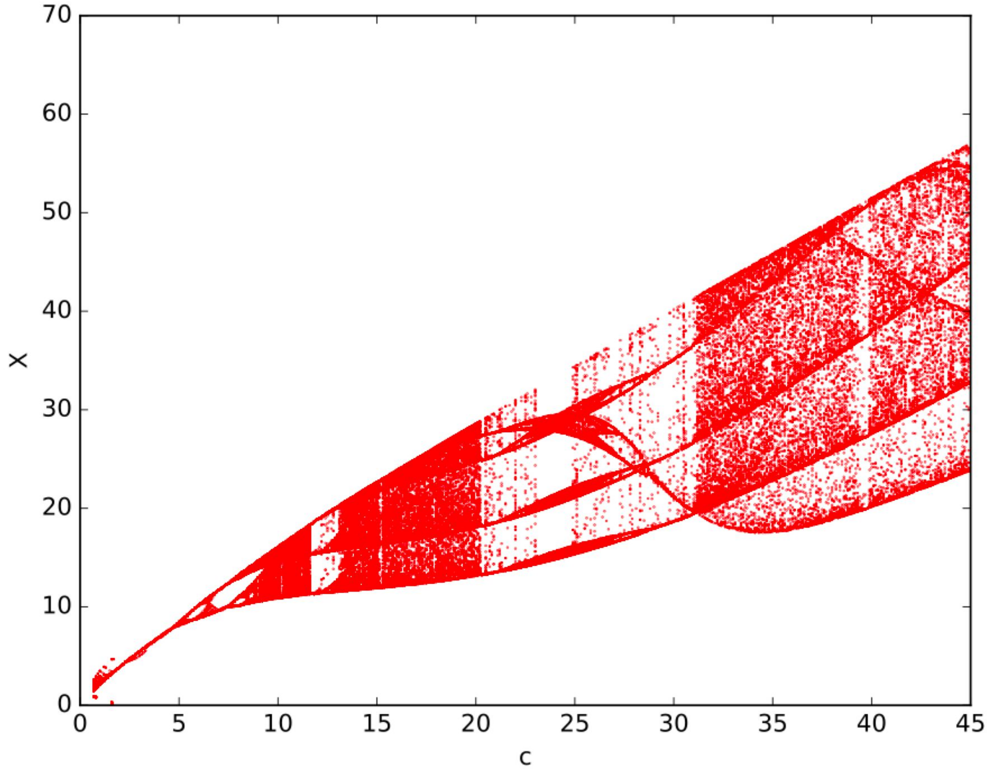
$\{a, b, c\}$ parameters

$$\left(\frac{c+\sqrt{c^2-4ab}}{2}, \frac{-c-\sqrt{c^2-4ab}}{2a}, \frac{c+\sqrt{c^2-4ab}}{2a}\right)$$

$$\left(\frac{c-\sqrt{c^2-4ab}}{2}, \frac{-c+\sqrt{c^2-4ab}}{2a}, \frac{c-\sqrt{c^2-4ab}}{2a}\right)$$



Bifurcation diagram



$$a = b = 0.1$$
$$c \in (0,45]$$

References

Rössler, Otto E. "An equation for continuous chaos." *Physics Letters A* 57.5 (1976): 397-398.

This is the original paper. The Rossler attractor can be seen as a Lorenz attractor with one lobe. Differently from Lorenz, in Rossler there is just one non-linearity.

Letellier, Christophe, and Valérie Messager. "Influences on Otto E. Rössler's earliest paper on chaos." International Journal of Bifurcation and Chaos 20.11 (2010): 3585-3616.

Historical overview on the Rossler system and its main influence in physics.

Delage, Olivier, and Alain Bourdier. "Selection of Optimal Embedding Parameters Applied to Short and Noisy Time Series from Rössler System." Journal of Modern Physics 8.09 (2017): 1607.

All you need to smartly succeed in this TP.

Observability

$\dot{W} = \mathbf{A}W$	$\mathbf{A} \in \mathbb{R}^{m \times m}$	Jacobian	$\mathbf{C} \in \mathbb{R}^{r \times m}$	measure matrix
$X = \mathbf{C}W$	$W \in \mathbb{R}^{m}$	State	$X \in \mathbb{R}^{r}$	measure

The system is observable in X if

$$\mathrm{rank}(\mathbf{Q}) = m$$

for Rossler *m* = 3 and *r* = 1
observing one coordinate

$$\mathbf{Q} = \begin{bmatrix} \mathbf{C} \\ \mathbf{CA} \\ \mathbf{CA}^2 \\ \vdots \\ \mathbf{CA}^{m-r} \end{bmatrix}$$

Observability in Rossler

$$\begin{cases} \dot{x} = -y - z \\ \dot{y} = x + ay \\ \dot{z} = b + z(x - c) \end{cases} \qquad \mathbf{A} = \begin{bmatrix} 0 & -1 & -1 \\ 1 & a & 0 \\ z & 0 & x - c \end{bmatrix} \qquad \mathbf{A}^2 = \begin{bmatrix} -1 - z & -a & c - x \\ a & a^2 - 1 & -1 \\ z(x - c) & -z & -z + (x - c)^2 \end{bmatrix}$$

$$\mathbf{C}_y = \begin{bmatrix} 0 & 1 & 0 \end{bmatrix} \qquad \mathbf{Q}_y = \begin{bmatrix} 0 & 1 & 0 \\ 1 & a & 0 \\ a & a^2 - 1 & -1 \end{bmatrix} \qquad \det(\mathbf{Q}_y) = 1$$

$$\mathbf{C}_x = \begin{bmatrix} 1 & 0 & 0 \end{bmatrix} \qquad \mathbf{Q}_x = \begin{bmatrix} 1 & 0 & 0 \\ 0 & -1 & -1 \\ -1 - z & -a & c - x \end{bmatrix} \qquad \det(\mathbf{Q}_x) = 0 \text{ if } x = a + c$$

$$\mathbf{C}_z = \begin{bmatrix} 0 & 0 & 1 \end{bmatrix} \qquad \mathbf{Q}_z = \begin{bmatrix} 0 & 0 & 1 \\ z & 0 & x - c \\ z(x - c) & -z & -z + (x - c)^2 \end{bmatrix} \qquad \det(\mathbf{Q}_z) = 0 \text{ if } z = 0$$

Observability in Rossler

- Generate the data (time series) with *a* = *b* = 0.2, *c* = 5.7
- Learn the discrete or continuous dynamical system from the time series

$$W_{n+1} = NN(W_n) \qquad \dot{W} = NN(W)$$

- 70% of the note: recover statistics
  - PDF
  - Time Correlations
  - Spectral density
  - ...

These analysis are enough to reach the 70% of the note but more analyses can make the difference!

- 30% of the note: recover dynamics
  - Equilibrium point (at least one)
  - Lyapunov exponents (the largest one) $\lambda \approx 7 \times 10^{-2}$

Differences:

Discrete		Continuous
$W_{t+\Delta t} = M(W_t)$	Dynamical system	$\dot{W} = F(W)$
$w_{t+\Delta t} = \mathbf{J} w_t$	Linearized system	$\dot{w} = \mathbf{A} w$
$\mathbf{J}$	Jacobian	$\mathbf{A}$
	Note that: $\mathbf{J} \approx e^{\mathbf{A}\Delta t}$	
$W_0 - M(W_0) = 0$	Equilibrium state	$F(W_0) = 0$

Constraints

One approach is enough

If you use temporal embedding or architectures with memory: **you can use just ONE coordinate (y of course!)**

$$\{y_1, y_0, y_{-1}, \dots\} = NN(\{y_0, y_{-1}, y_{-2}, \dots\})$$
$$\{y_1, \dot{y}_1, \ddot{y}_1, \dots\} = NN(\{y_0, \dot{y}_0, \ddot{y}_0, \dots\})$$
$$y_1, [h_1] = NN(y_0, [h_0])$$

h_0 memory in RNN

Without temporal embedding, discrete or continuous systems: **use the whole state**

$$W_1 = NN(W_0)$$

$$\dot{W} = NN(W)$$

Be smart

To find the equilibrium point and to evaluate the Lyapunov exponent, the Jacobian has to be computed. Introduce a penalization onto the sensibility of your model wrt the inputs!

$$loss = \|W - \hat{W}\| + \lambda(?) \longrightarrow \begin{array}{c} \|\dot{W} - \hat{\dot{W}}\| \\ \|A - \hat{A}\| \\ \|\hat{A}\|_F \end{array}$$

There is not a unique way to proceed! If you use a purely data driven approach (without explicitly introducing the true Jacobian in the loss function) will be appreciated.

Attention: with a RNN the **total Jacobian** needs to be computed!

Appendix D in: “Backpropagation Algorithms and Reservoir Computing in Recurrent Neural Networks for the Forecasting of Complex Spatiotemporal Dynamics”

Be Zen 1/2

Before start take some time to study the problem, read the papers. Take care to re-formulate the problem according to your choice. For instance, given $\{x_e, y_e, z_e\}$ the equilibrium point, if you choose:

1) temporal embedding $\{y_1, y_0, y_{-1}, \dots\} = NN(\{y_0, y_{-1}, y_{-2}, \dots\})$
   the new equilibrium point is $\{y_e, y_e, y_e, \dots\}$
2) derivative embedding $\{y_1, \dot{y}_1, \ddot{y}_1, \dots\} = NN(\{y_0, \dot{y}_0, \ddot{y}_0, \dots\})$
   the new equilibrium point is $\{y_e, 0, 0, \dots\}$

With temporal embedding the Lyapunov exponent might change but **still positive**. The reconstructed phase space is a diffeomorphism wrt the original phase space (Takens's theorem): one-to-one mapping between original and reconstructed phase space but the geometry might change!

Be Zen 2/2

Try to recover a pure data driven model. If you choose to recover a continuous system, time derivatives can be evaluated from the observed trajectory through finite differentiation.

Another possibility is to use ODE Net which solves the variational problem under continuous-in-time constraint [**Chen, Ricky TQ, et al. "Neural ordinary differential equations." arXiv:1806.07366 (2018)**]. ODE Net available at: https://github.com/rtqichen/torchdiffeq

The Jacobian can be computed through automatic differentiation librairies. Straightforward to use in Tensorflow or PyTorch. If you don't want to use automatic differentiation to recover the Jacobian, finite differentiation can do the job.

GOOGLE IS YOUR FRIEND

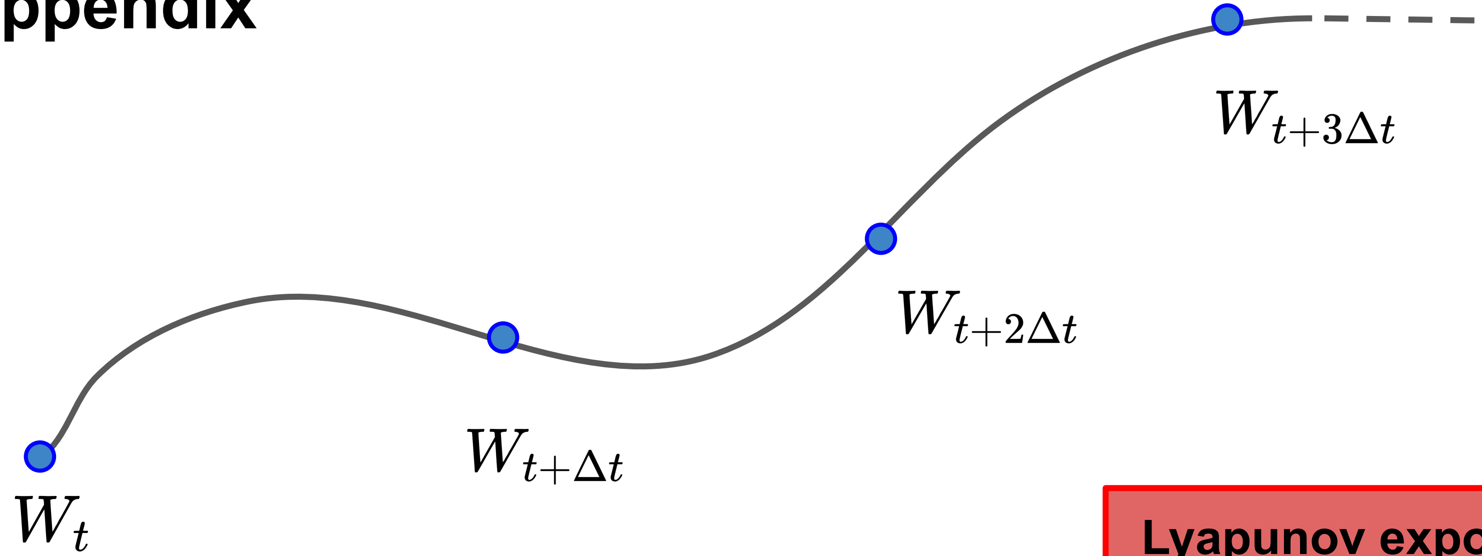
Deliver

1. small report (~4 pages max, 5 figures max)
   a. Justify the loss function
   b. Justify the regularization (if any)
   c. Justify the analysis you carried out to validate the model
2. codes to reproduce the pictures in your report (+ trained NN)
3. code to generate a new time series with your trained model, please complete the script **time_series.py**. I will choose a new initial condition with the SHELL command:

   ```
   $   python time_series.py --init  a  b  c
   ```

   where a b c are float numbers used as initial condition. Please follow the comment in the script without change the output file shape/name.

Appendix



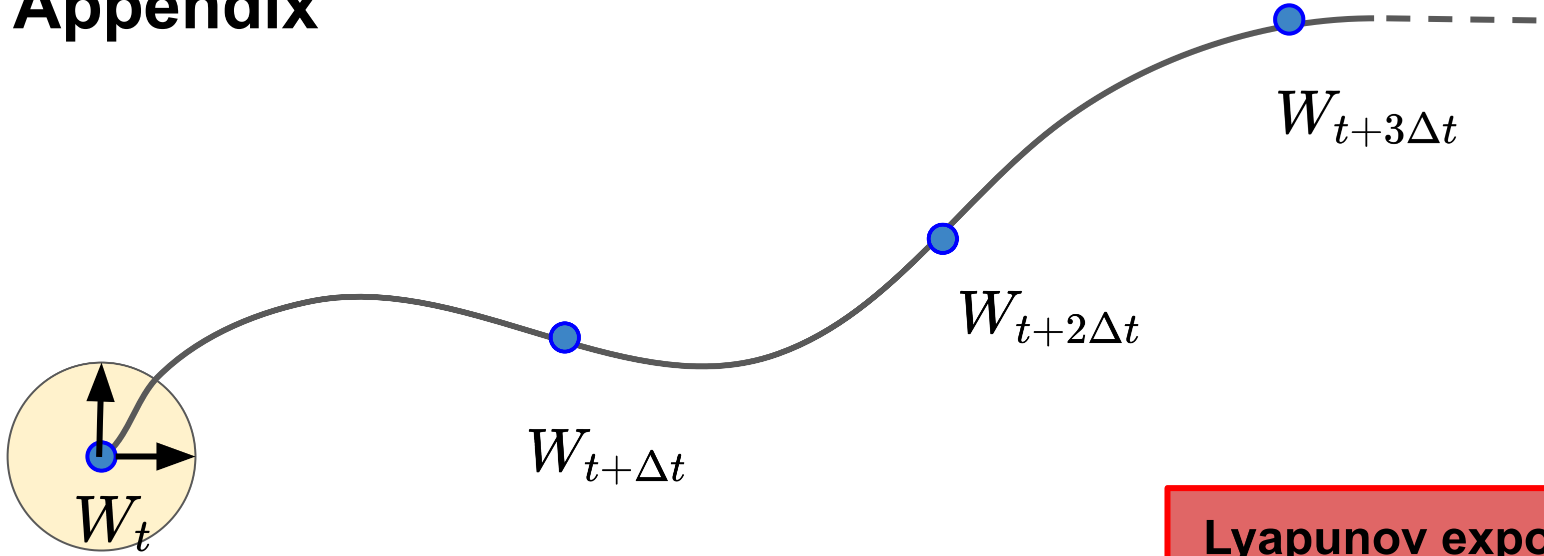
Lyapunov exponent:

$$|\delta W(t)| = e^{\lambda t}|\delta W_0|$$

where

$$|\delta W_0| = |W_0 - W_0'|$$

Appendix



Initial guess with unitary energy

```
w = np.eye(n)
```

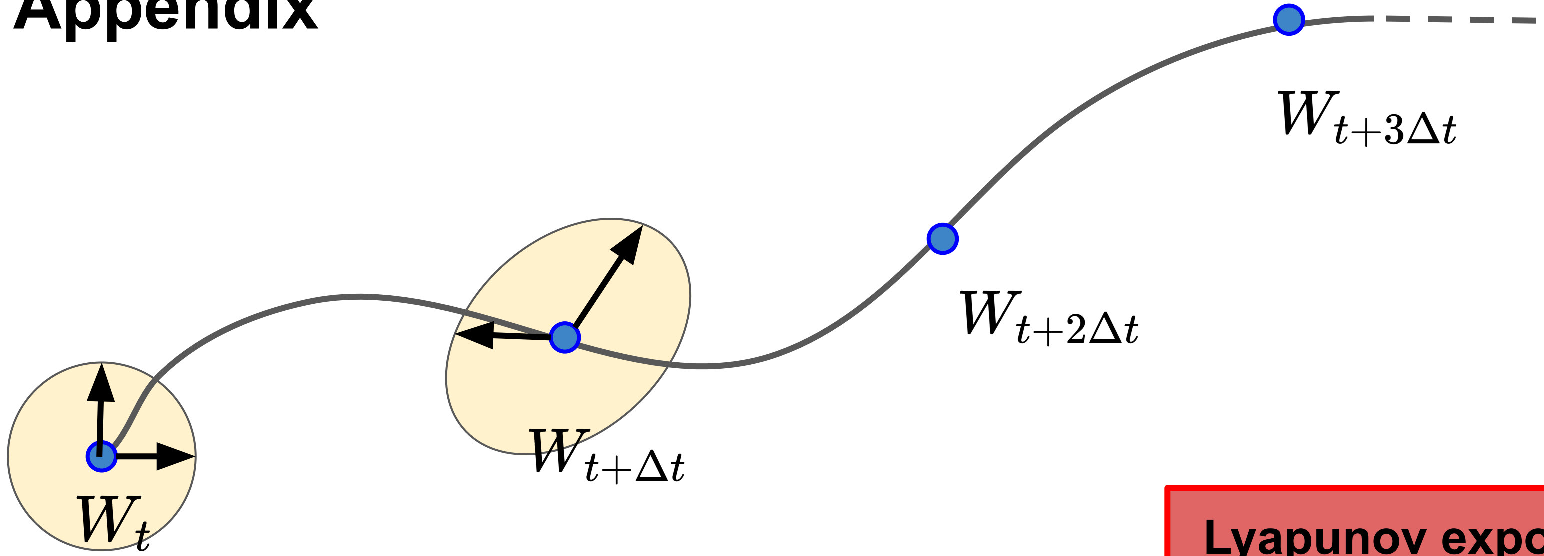
Lyapunov exponent:

$$|\delta W(t)| = e^{\lambda t}|\delta W_0|$$

where

$$|\delta W_0| = |W_0 - W_0'|$$

Appendix



Initial guess with unitary energy

```
w = np.eye(n)
```

Evolution of the initial guess following the tangent trajectory

```
w_next = np.dot(expm(jacob * delta_t),w)
```

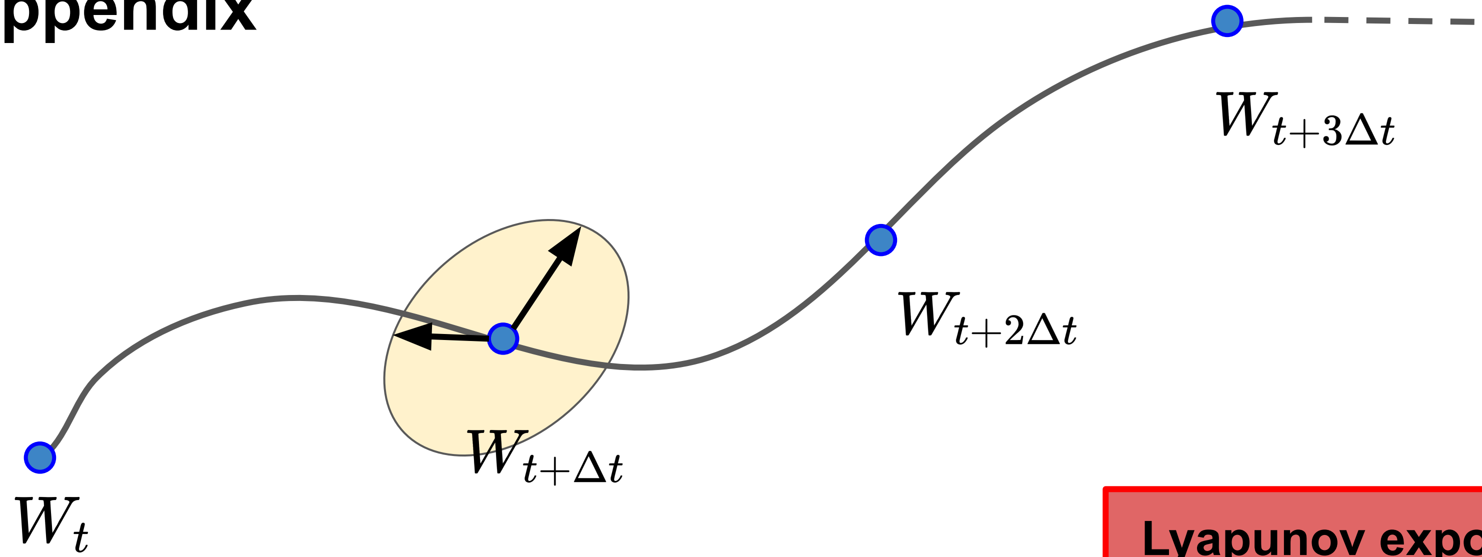
Lyapunov exponent:

$$|\delta W(t)| = e^{\lambda t}|\delta W_0|$$

where

$$|\delta W_0| = |W_0 - W_0'|$$

Appendix



Evaluate the stretching and the new orthonormal base

```
w_next, r_next = qr(w_next)
```

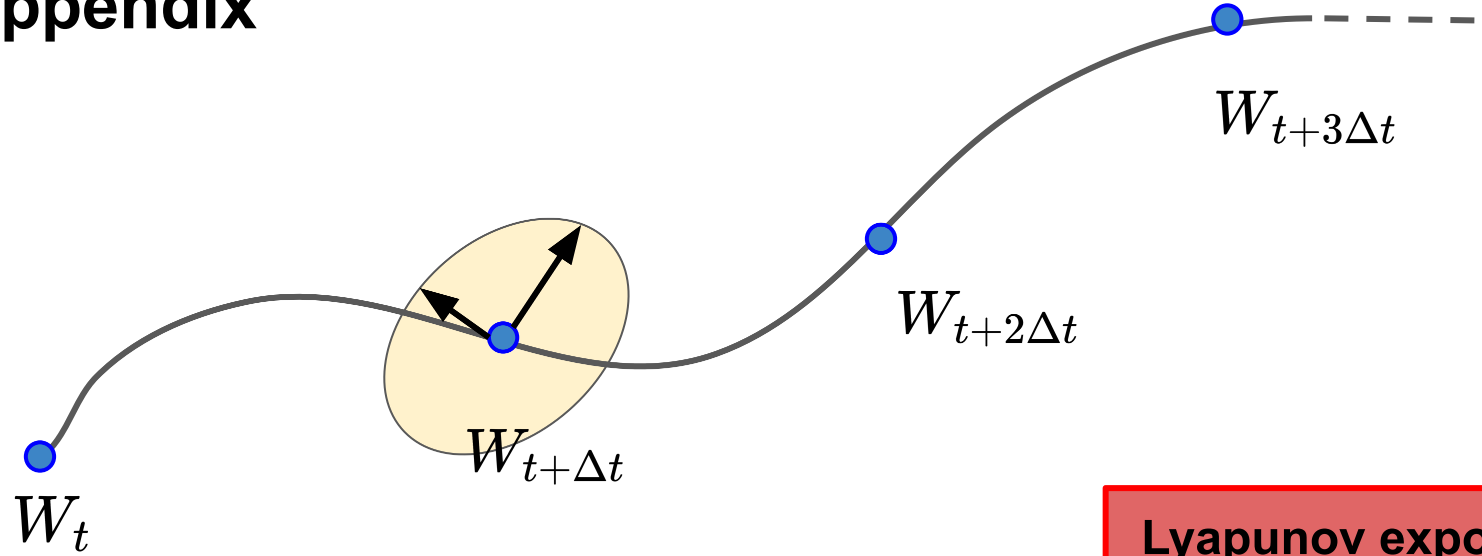
Lyapunov exponent:

$$|\delta W(t)| = e^{\lambda t}|\delta W_0|$$

where

$$|\delta W_0| = |W_0 - W_0'|$$

Appendix



Evaluate the stretching and the new orthonormal base

```
w_next, r_next = qr(w_next)
```

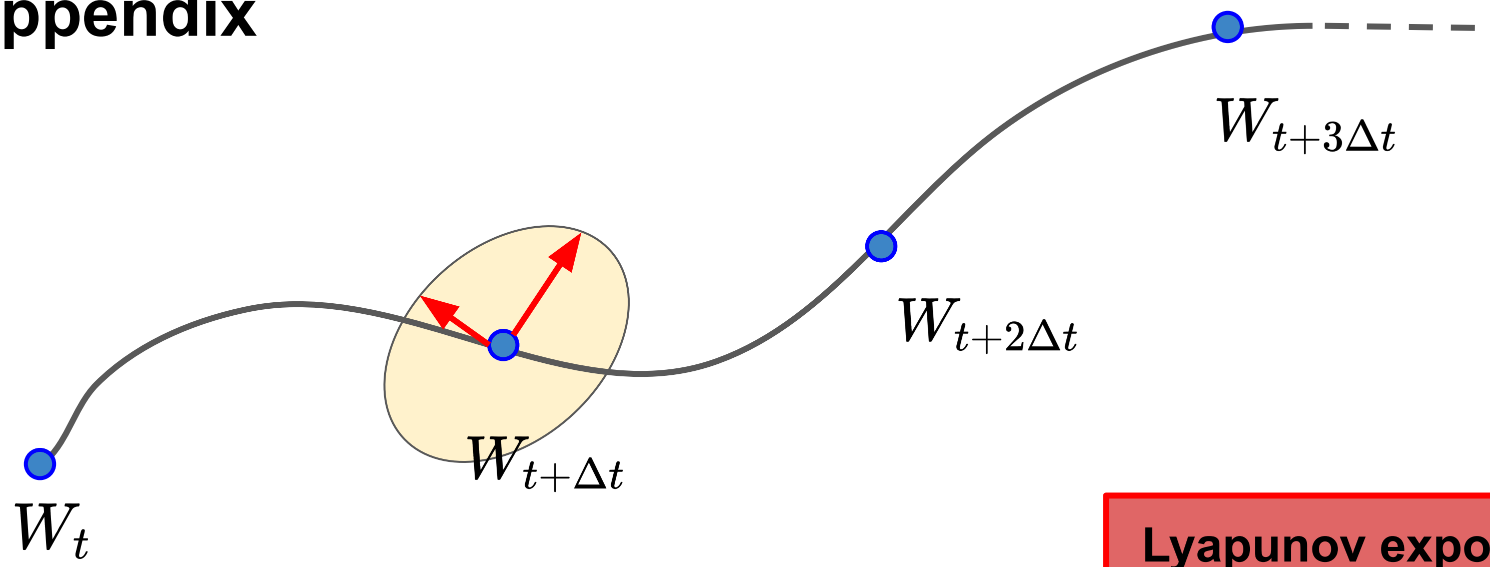
Lyapunov exponent:

$$|\delta W(t)| = e^{\lambda t}|\delta W_0|$$

where

$$|\delta W_0| = |W_0 - W_0'|$$

Appendix



Evaluate the stretching and the new orthonormal base

```
w_next, r_next = qr(w_next)
```

Store the amplification factors

```
rs.append(r_next)
```

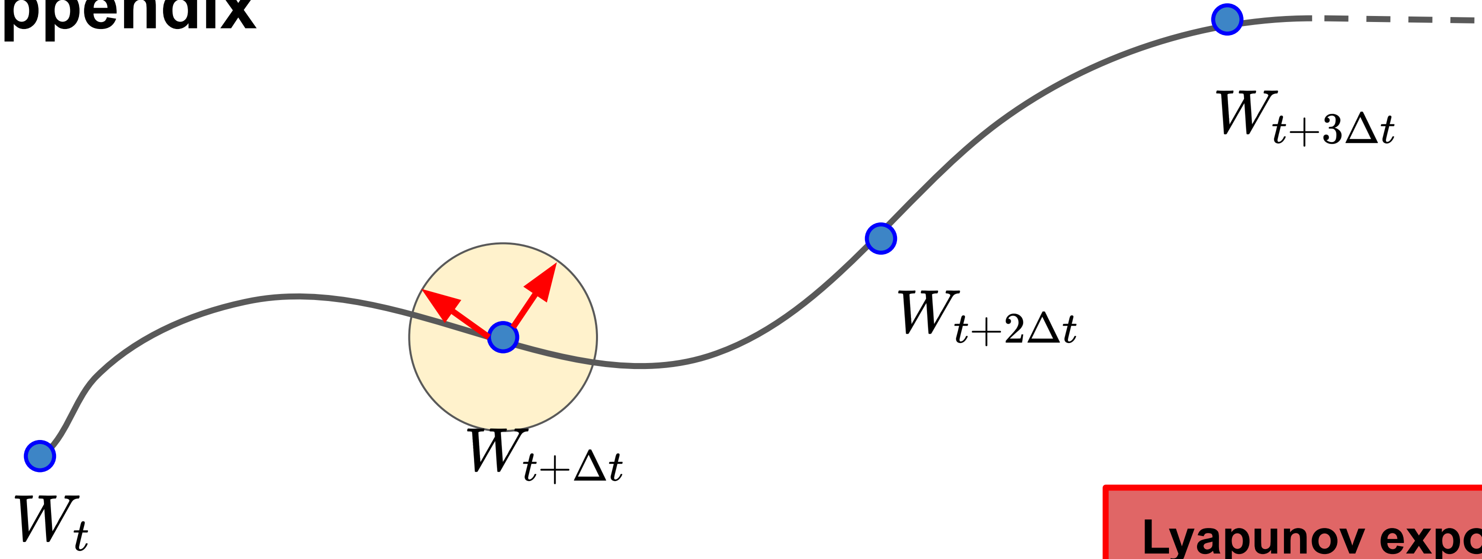
Lyapunov exponent:

$$|\delta W(t)| = e^{\lambda t} |\delta W_0|$$

where

$$|\delta W_0| = |W_0 - W_0'|$$

Appendix



Evaluate the mean amplification

`mean(rs)` $\approx e^{\lambda_i t}$

`mean(log(rs))/t` $\approx \lambda_i$

Lyapunov exponent:

$$|\delta W(t)| = e^{\lambda t}|\delta W_0|$$

where

$$|\delta W_0| = |W_0 - W_0'|$$

Personal considerations

Training a RNN for a discrete system using just y coordinate is fast and safe. On the other hand recover the Jacobian is not straightforward.

Training a Neural Network to emulate the discrete Rossler system without temporal embedding is fast but you need to accurately design the loss function and the penalization. Recover the Jacobian is easy with automatic differentiation.

Training a Neural Network to recover the continuous Rossler system is hard. Exponentially more complicated ($\mathbf{J} \approx e^{\mathbf{A}\Delta t}$), an error on A￼ is amplified in the solution propagation by ￼J. Moreover in the continuous space, cross trajectories are not allowed. BUT, if you are able to got it, the Jacobian is for free and almost exact.