Scientific Computation

Spring, 2019

Lecture 19

Notes

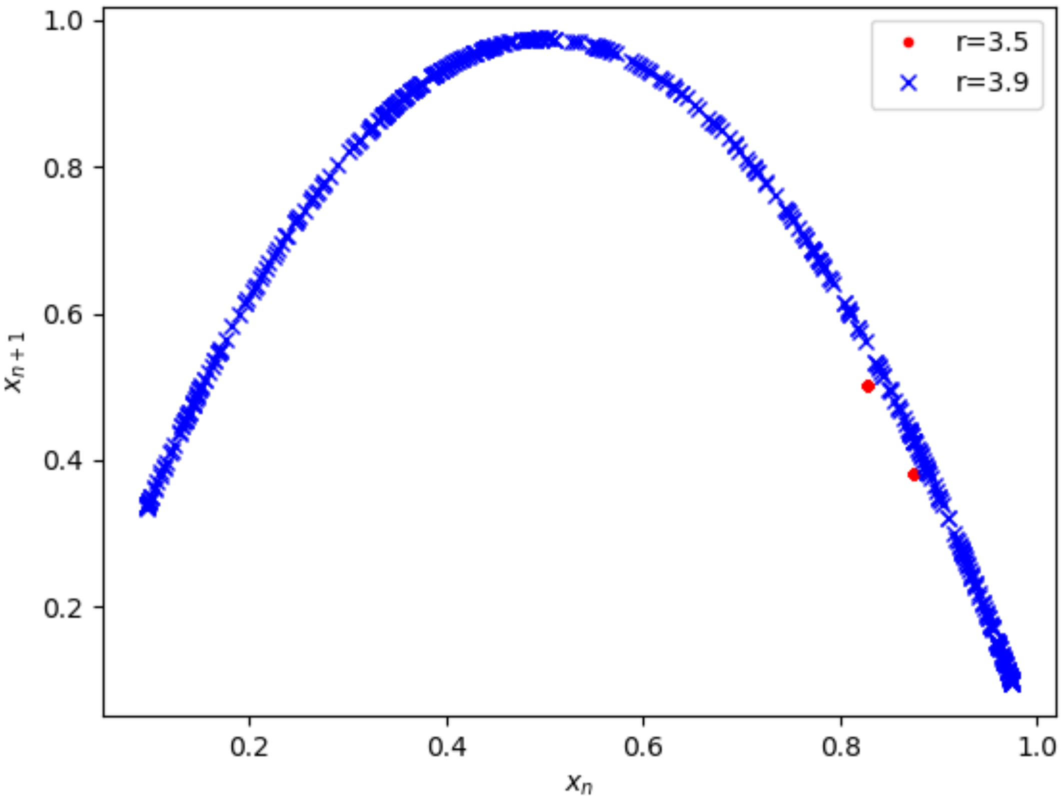
- No labs this week

- Extra office hours: Wednesday 10-11, Thursday 1-2, both in MLC

Today

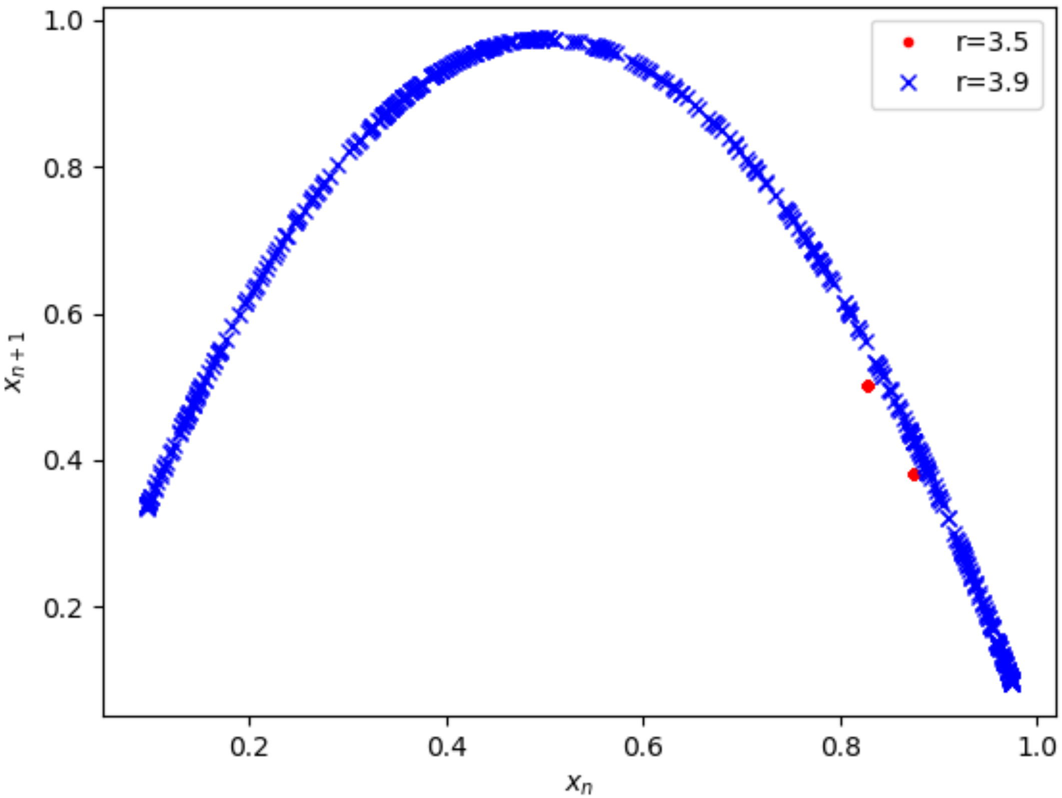
- Investigating nonlinear dynamics and chaos
  - Transitions between dynamical states
  - Qualitative similarities between difference and differential equations
  - Sensitivity to initial conditions

Logistic map



- **At** r=3.5**, there are *period-2* oscillations**
- $\mathbf{x_{n+4} = x_n}$
- **At** r=3.9**, we have chaos**
- **What other states are there?**
- **For what values of** r **are there transitions?**

Logistic map



- At r=3.5, there are *period-2* oscillations
- $x_{n+4} = x_n$
- At r=3.9, we have chaos
- What other states are there?
- For what values of r are there transitions?

Simple approach: compute "all" values of x visited for a range of r over a sufficiently large number of interations

Bifurcation diagram

- **Basic solution is straightforward:**

```
for i in range(N):
    x[i+1] = r*x[i]*(1-x[i])
```

- **Discard influence of initial condition:**

```
y = x[N//2:]
n = y.size
```

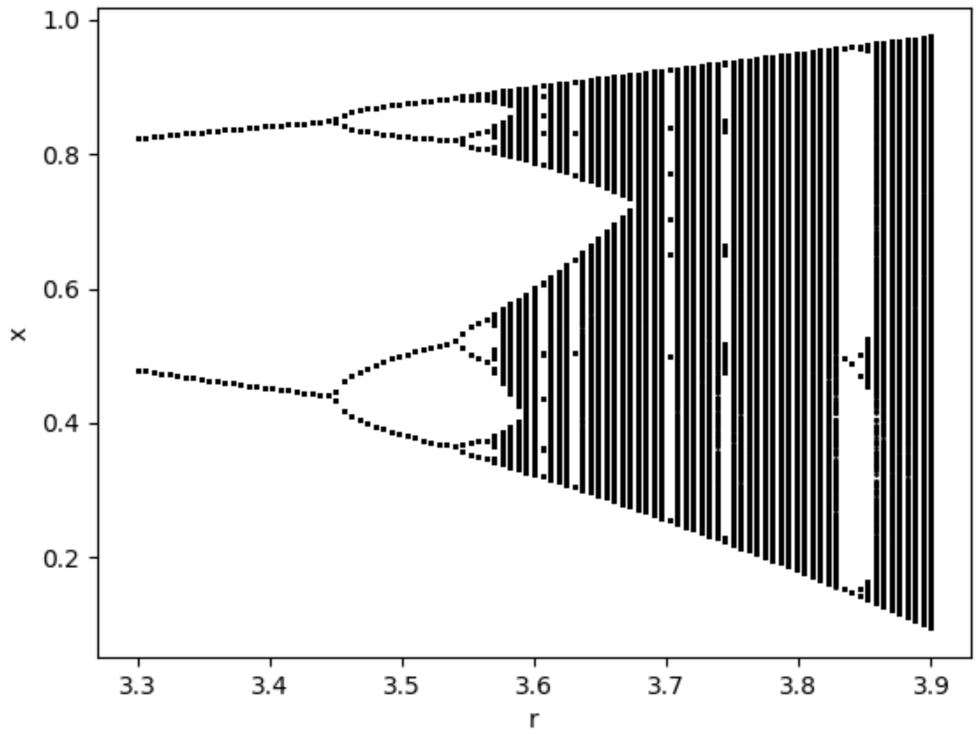
- **Loop over a range of r, plotting all x as we go along:**

```
for r in rarray:
    #Setup
    x = np.zeros(N+1)
    x[0]=x0
    #Main calculation
    for i in range(N):
        x[i+1] = r*x[i]*(1-x[i])
    x = x[N//2:]
    rplot = np.ones_like(x)*r
    plt.plot(rplot,x,'k.',markersize=2)
```

Bifurcation diagram

- Provides concise global view of dynamics

- What is it showing:
  - $r < 3.45$: period-1 oscillation
  - $3.45 < r < 3.545$: period-2
  - Then period 4, period 8, …
  - Until at around $r=3.5699$ we have aperiodic, chaotic dynamics
  - But for larger r there are periodic “windows”!



Differential equations

- A range of difference equations (maps) show similar behavior
  - The map should look like an "upside-down U" (or V) in some sense
- What about differential equations?
  - We have seen that ideas on fractal dimension can carry over
  - Are bifurcation diagrams similar in any sense?
  - Can nonlinear ODEs be viewed as maps in some way?

Rossler system

$$\frac{dx}{dt} = -y - z$$

$$\frac{dy}{dt} = x + ay$$

$$\frac{dz}{dt} = b + z(x - c)$$

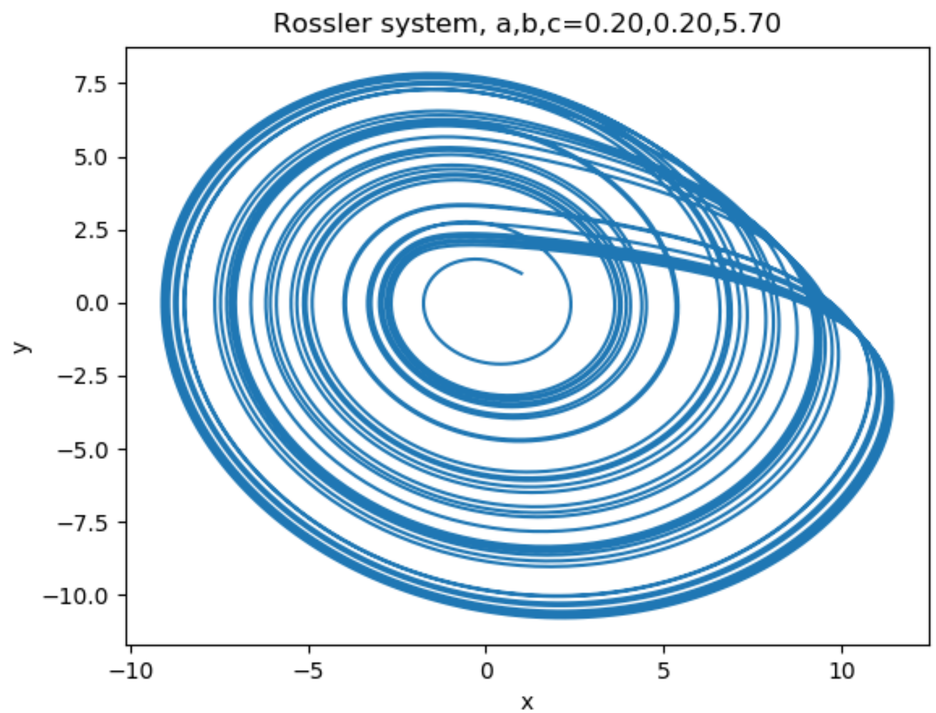
- **"Simplest" system of ODEs which produce chaos**
- a, b**, and** c **are parameters. In certain parameter ranges, chaotic dynamics are generated**
- **Motivated by Lorenz system and consideration of chaotic maps**
- **Can integrate these equations using** odeint**…**

Rossler system

$$\frac{dx}{dt} = -y - z$$

$$\frac{dy}{dt} = x + ay$$

$$\frac{dz}{dt} = b + z(x - c)$$



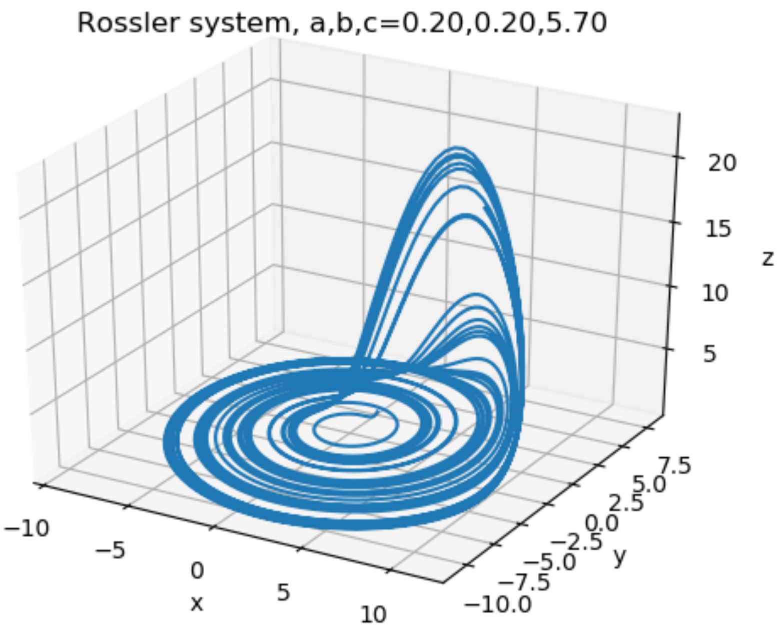
- These are trajectories in the x-y plane (a phase plane)
- It looks like trajectories are crossing (suggests periodic dynamics), but they are in fact avoiding each other…

Rossler system

$$\frac{dx}{dt} = -y - z$$

$$\frac{dy}{dt} = x + ay$$

$$\frac{dz}{dt} = b + z(x - c)$$



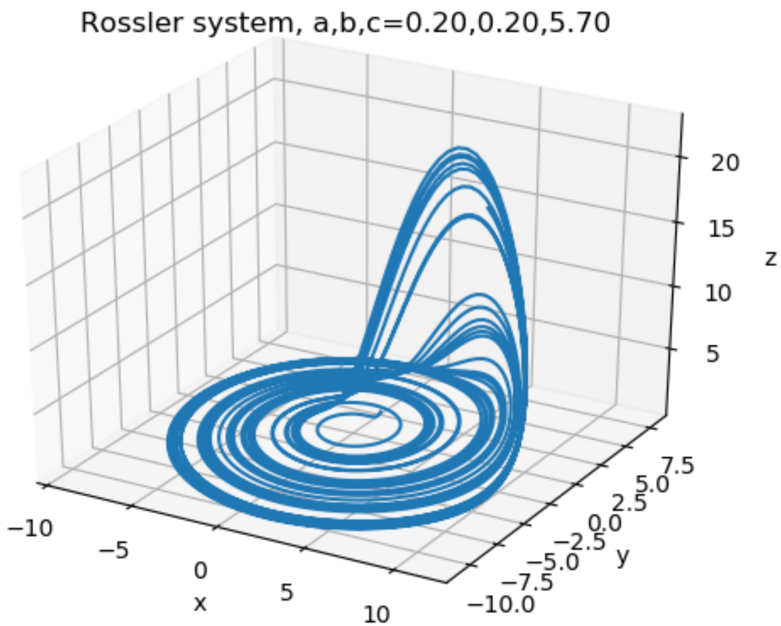
- These are trajectories in the x-y plane (a phase plane)
- It looks like trajectories are crossing (suggests periodic dynamics), but they are in fact avoiding each other…

Rossler system

$$\frac{dx}{dt} = -y - z$$

$$\frac{dy}{dt} = x + ay$$

$$\frac{dz}{dt} = b + z(x - c)$$



- These are trajectories in the x-y plane (a phase plane)
- It looks like trajectories are crossing (suggests periodic dynamics), but they are in fact avoiding each other…
- We can again consider transitions between states, this time varying c

Bifurcation diagram

- **Basic solution is straightforward:**

```
f = odeint(RHS,f0,t,args=(a,b,c))
```

- **Discard influence of initial condition:**

```
f = f[iskip:,:]
```

- **Loop over a range of** c, **but now, we extract local maxima of** x:

```
dx = np.diff(x)
    d2x = dx[:-1]*dx[1:]
    ind = np.argwhere(d2x<0) #locations of extrema
```

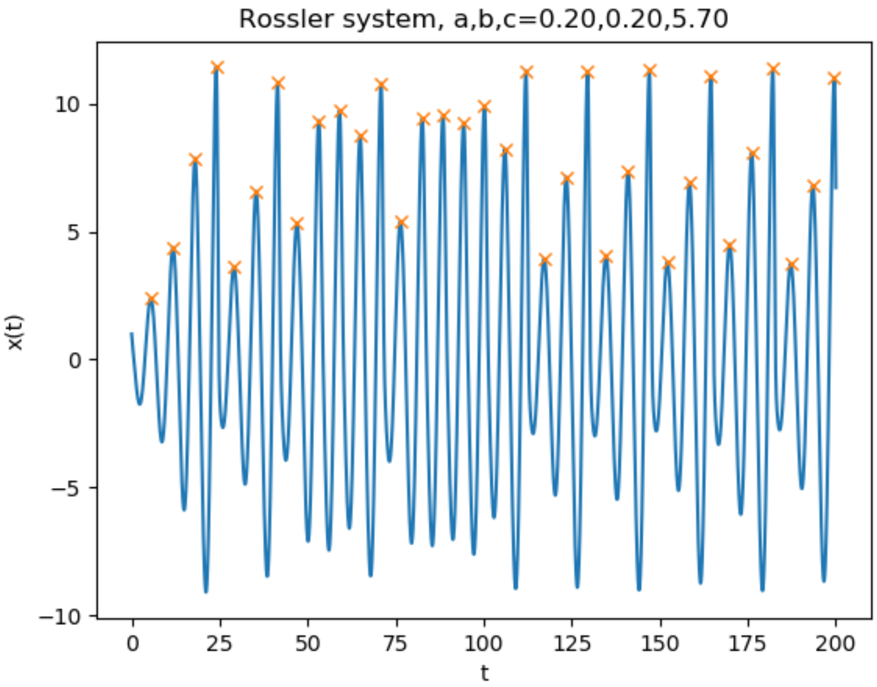
- ind+1 **contains locations of both maxima and minima, just need to separate even and odd indices**

Bifurcation diagram

- **Loop over a range of** c**, but now, we extract local maxima of** x:

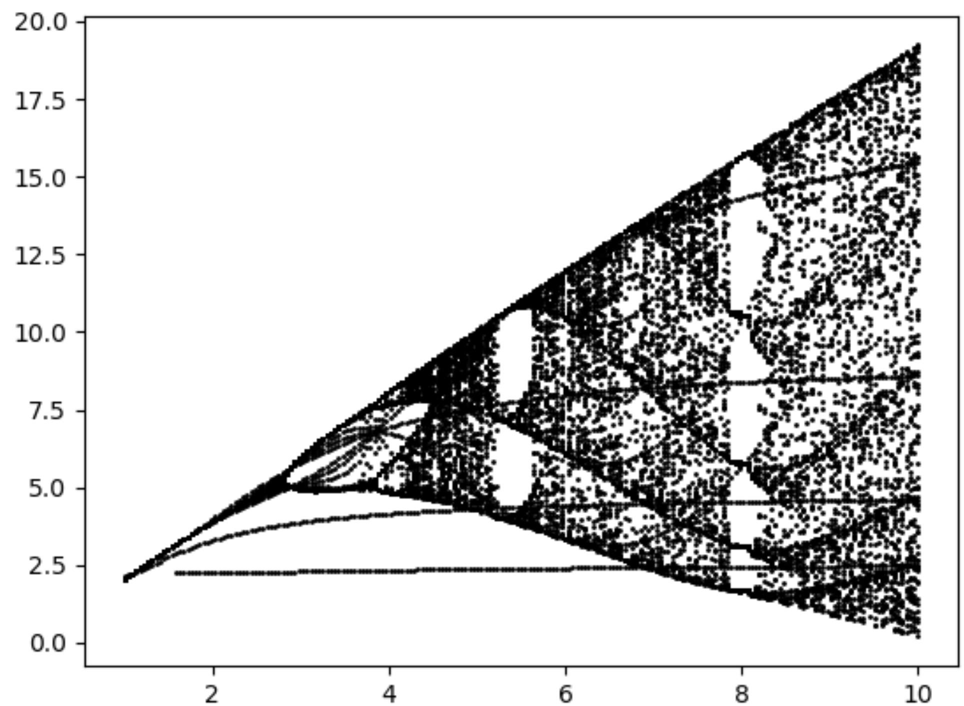
```
dx = np.diff(x)
    d2x = dx[:-1]*dx[1:]
    ind = np.argwhere(d2x<0) #locations of extrema
```

- ind+1 **contains locations of both maxima and minima, just need to separate even and odd indices:** `plot(t[ind[1::2]+1],x[ind[1::2]+1],'x')`



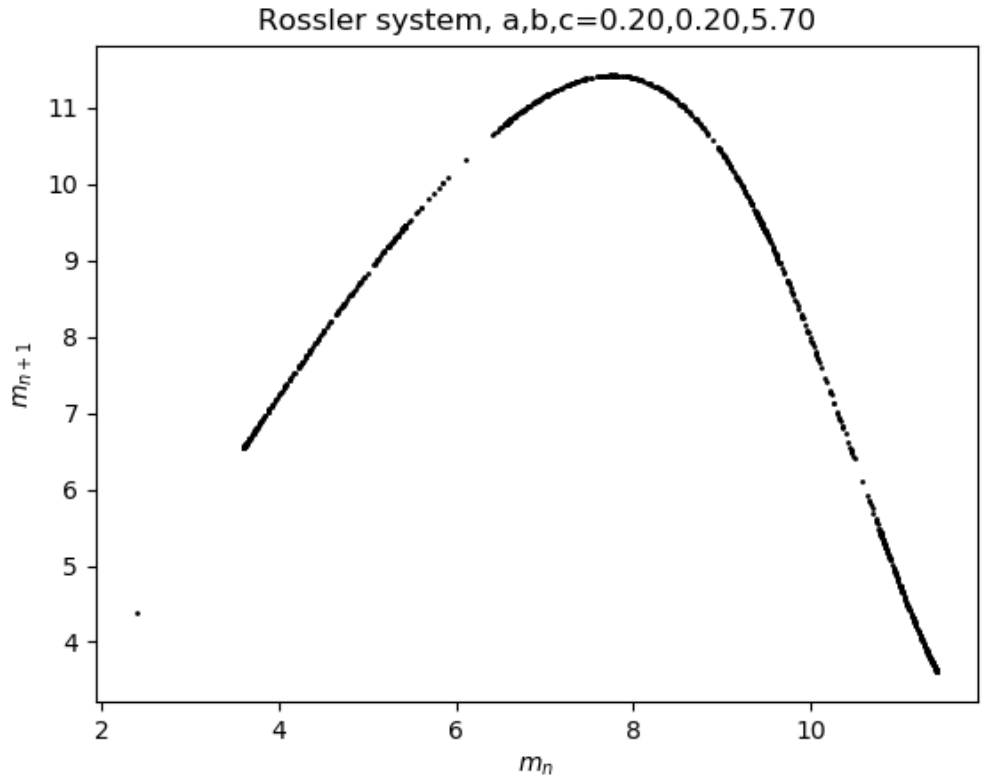
Bifurcation diagram

- Again, we have:
  - a sequence of period doublings
  - leading to chaos
  - with subsequent periodic windows
- Can be computed with ~20 lines of code in 2-3 minutes



1-D maps

- Given the qualitative similarities in the bifurcation diagrams, is there an underlying map that can be extracted for the Rossler system?
- This idea comes from Lorenz:
- Let f_n represent the nth maximum of x(t) (already constructed for bifurcation diagram)
- Then plot f_{n+1} vs f_n:



- We expect *unimodal maps* to be "hiding" behind chaotic dynamics
- V-map for Lorenz eqns.
- Similar maps have been Constructed from measurements as well!

Lyapunov exponents

- Last point to consider is arguably the most important
- For (continuous) chaotic systems, we expect:
  – Fractal dimension > 2
  – Aperiodic dynamics
  – Sensitivity to initial conditions

- This last point is characterized by *Lyapunov exponents*
  – Given two initial conditions, (x,y,z) and $(x+\varepsilon,y,z)$, consider the distance between the subsequent solutions
  – In a linear system, the distance will remain small
  – However in a chaotic system, the distance will increase exponentially
  – The rate of exponential growth is the Lyapunov exponent.

Lyapunov exponent

- **A precise computation requires consideration of several initial conditions**
- **We will just work through a simple example using the Lorenz system:**

$$\frac{dx}{dt} = \sigma(y - x)$$

$$\frac{dy}{dt} = rx - y - xz$$

$$\frac{dz}{dt} = xy - bz$$

- **First run simulation from arbitrary initial condition up to t=1 and take solution at t=1 as initial condition:**

```
T = 10
Nt = 10
t = np.linspace(0,T,Nt+1)
f0 = [1,1,1] #arbitrary initial condition

f = odeint(RHS,f0,t,args=(r,s,b))
```

Lyapunov exponent

- **First run simulation from arbitrary initial condition up to t=1 and take solution at t=1 as initial condition:**

```
T = 1
Nt = 10
t = np.linspace(0,T,Nt+1)
f0 = [1,1,1] #arbitrary initial condition
f = odeint(RHS,f0,t,args=(r,s,b))
```

- **Next, integrate forward using the solution at t=1 as an initial condition:**

```
T=40
Nt=2000
t = np.linspace(0,T,Nt+1)
f0 = f[-1,:] #initial condition taken from solution above
f = odeint(RHS,f0,t,args=(r,s,b))
x,y,z = f[:,0],f[:,1],f[:,2]
```

Lyapunov exponent

- **First run simulation from arbitrary initial condition up to** t=1 **and take solution at t=1 as initial condition:**

```
T = 10
Nt = 10
t = np.linspace(0,T,Nt+1)
f0 = [1,1,1] #arbitrary initial condition

#call odeint and rearrange output
f = odeint(RHS,f0,t,args=(r,s,b))
```

- **Repeat using "perturbed" initial condition:**

```
f0[0]+=1e-6 #perturbed initial condition
f2 = odeint(RHS,f0,t,args=(r,s,b))
x2,y2,z2 = f2[:,0],f2[:,1],f2[:,2]
```

- **And compute distance^2 between the two solutions:**

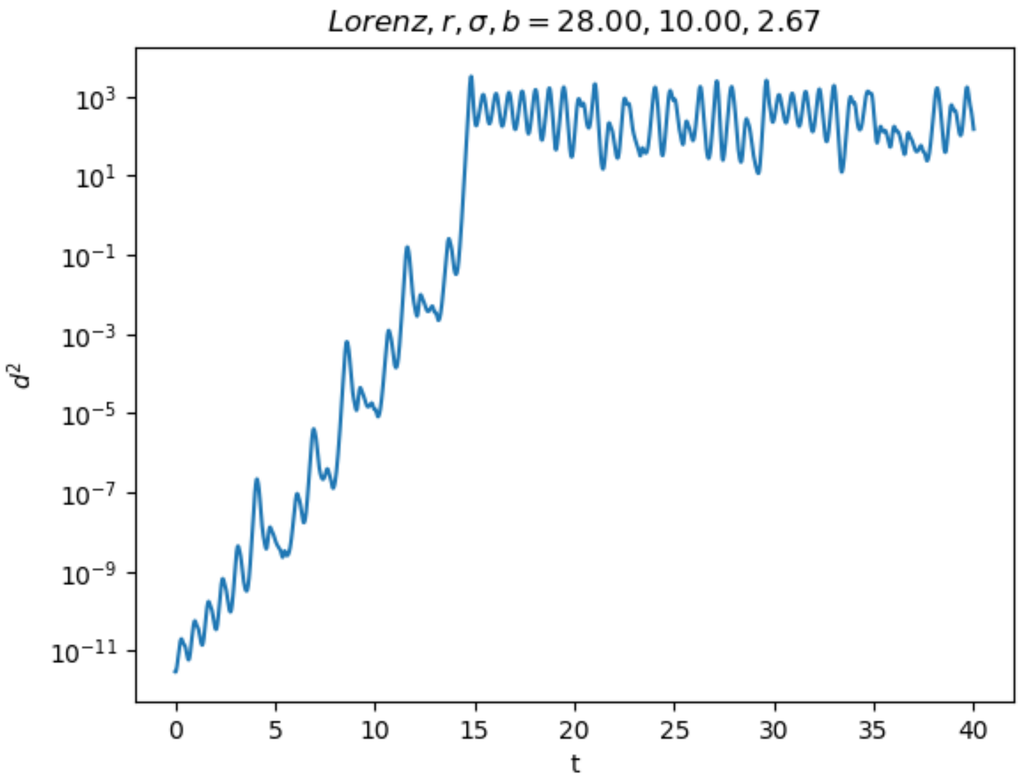
```
d2 = (x2-x)**2 + (y2-y)**2 + (z2-z)**2
```

Lyapunov exponent

- **Note that the distance initially grows exponentially:**

- **From previous plots, we know that there will be an upper bound**

Can estimate Lyapunov exponent using least-sqaures fit…

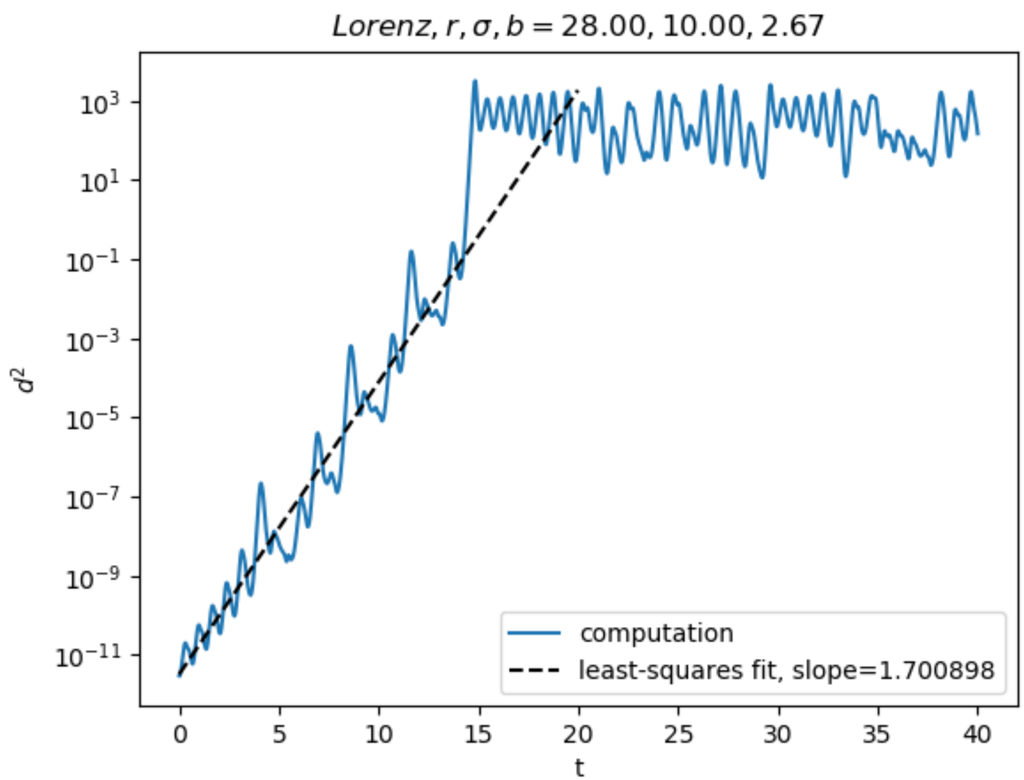


Lyapunov exponent

- Note that the distance initially grows exponentially:

- From previous plots, we know that there will be an upper bound

Can estimate Lyapunov exponent using least-sqaures fit…



The exponent is half of the slope (because distance squared is plotted)

More careful calculations estimate the exponent as ~0.9 for the Lorenz eqns.

Lyapunov exponents

- Last point to consider is arguably the most important

- Why is this important?
  - It has important consequences for forecasting
  - Small initial errors can lead to substantial errors in predictions at later times
  - This is (partly) why weather forecasts more than 10 days ahead are not very accurate

- Often misinterpreted in terms of “butterfly effect”
  - Probability of butterfly initiating a storm is infinitesimally small!