L2 Computational Physics
Week 8 – Chaos and Fractals

Graph Formatting

- Almost all plots and axes are labelled...
- Basic exam technique: If you're stuck on a hard part of a problem, don't neglect the easy parts!

Graphs beyond this course

- Concise, Precise
- Axes labels – **quantity** and **unit**
- Legend – is one required?
- How to plot data
  - Discrete points – **data markers**
  - Continuous functions, model fits – **plot lines**
- Titles – not usual as CAPTION figure!

Exponent notation

- I saw a lot of code like this in the GD assessment

```
if abs(stuff) < 0.0000000012
```

- It's hard as a human to read that number precisely
  - Easy to introduce bugs
    - e.g. `0.000000012 vs 0.0000000012`
- Python supports exponent notation
  - A lower case "e" after a number means "multiply by ten to the power of …"

```
if abs(stuff) < 1.2e-9
```

Chaos and Fractals

Chaotic Systems

- A Chaotic System has several properties:
  - a *deterministic* system
  - that is *highly sensitive* to *initial conditions*
  - That exhibits *topological mixing*

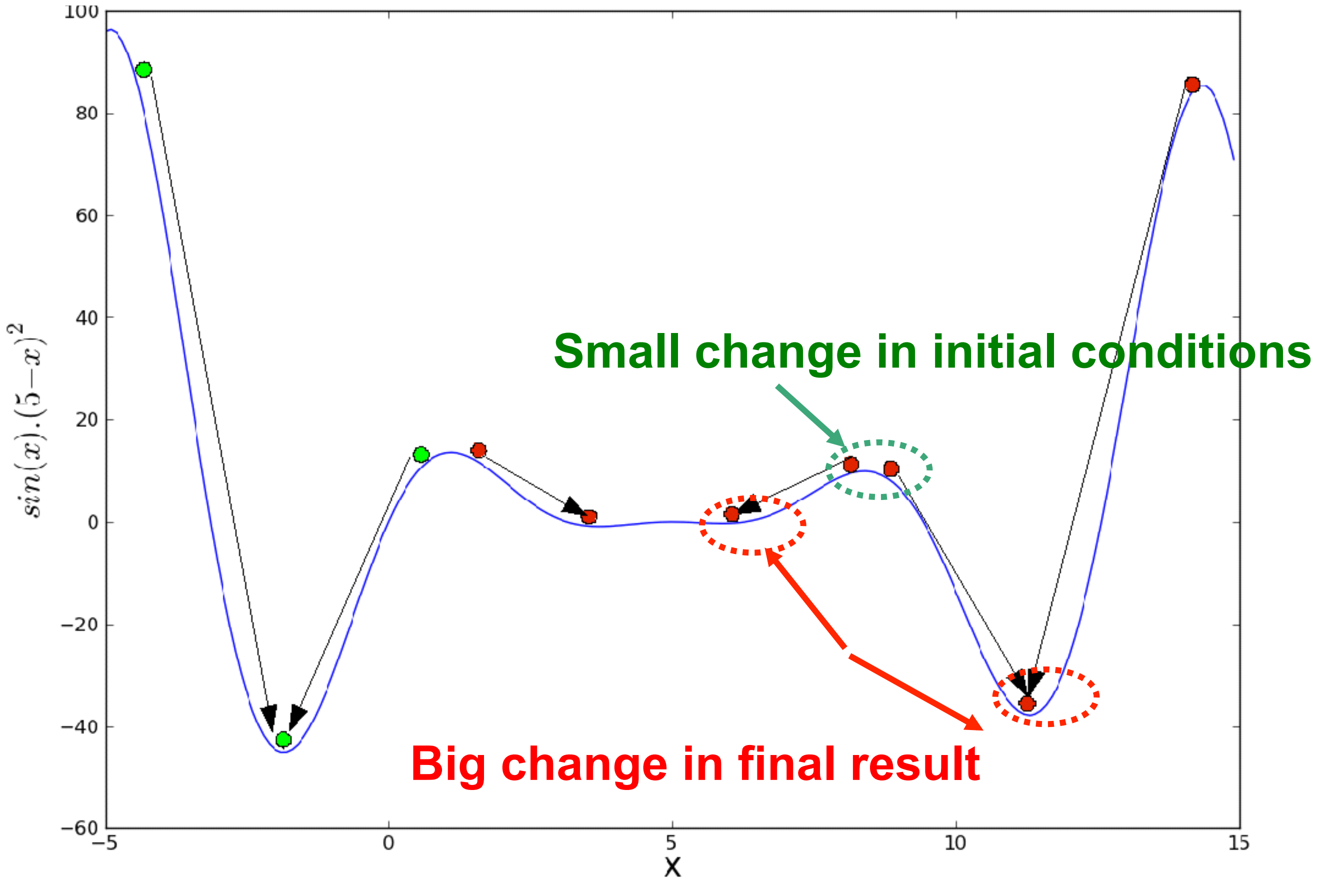
- Random and chaotic are not the same

Deterministic

- Some future state is a function of the current state
- This should be very familiar by now
- `state_t1 = f(initial_conditions)`
- `state_t2 = f(state_t1)`
- `state_t3 = f(state_t2)`
- `etc.`

Sensitive to Initial Conditions

A well tuned gradient descent is sensitive to initial conditions but **Not** chaotic



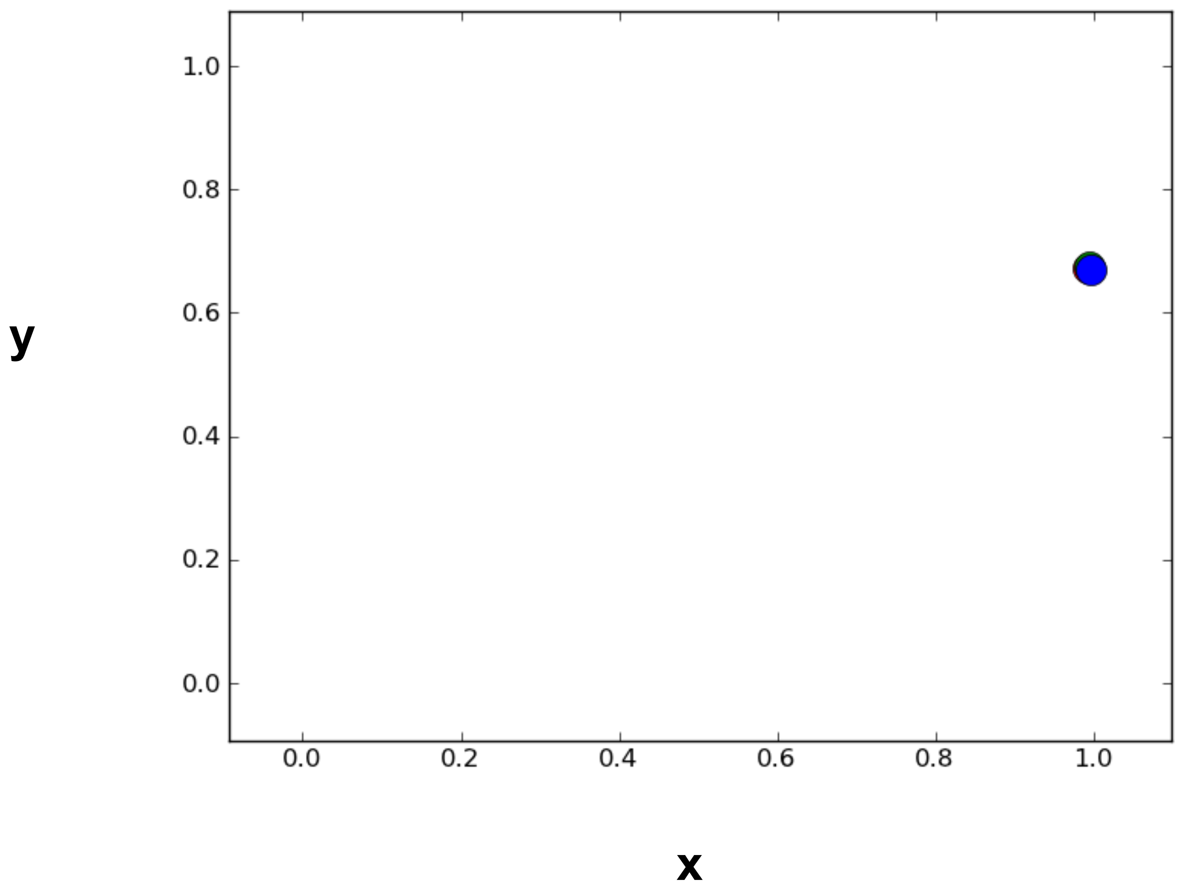
Topological Mixing

- Gradient Descent is not chaotic with small gamma
  - simple mapping of initial position to final position - go downhill

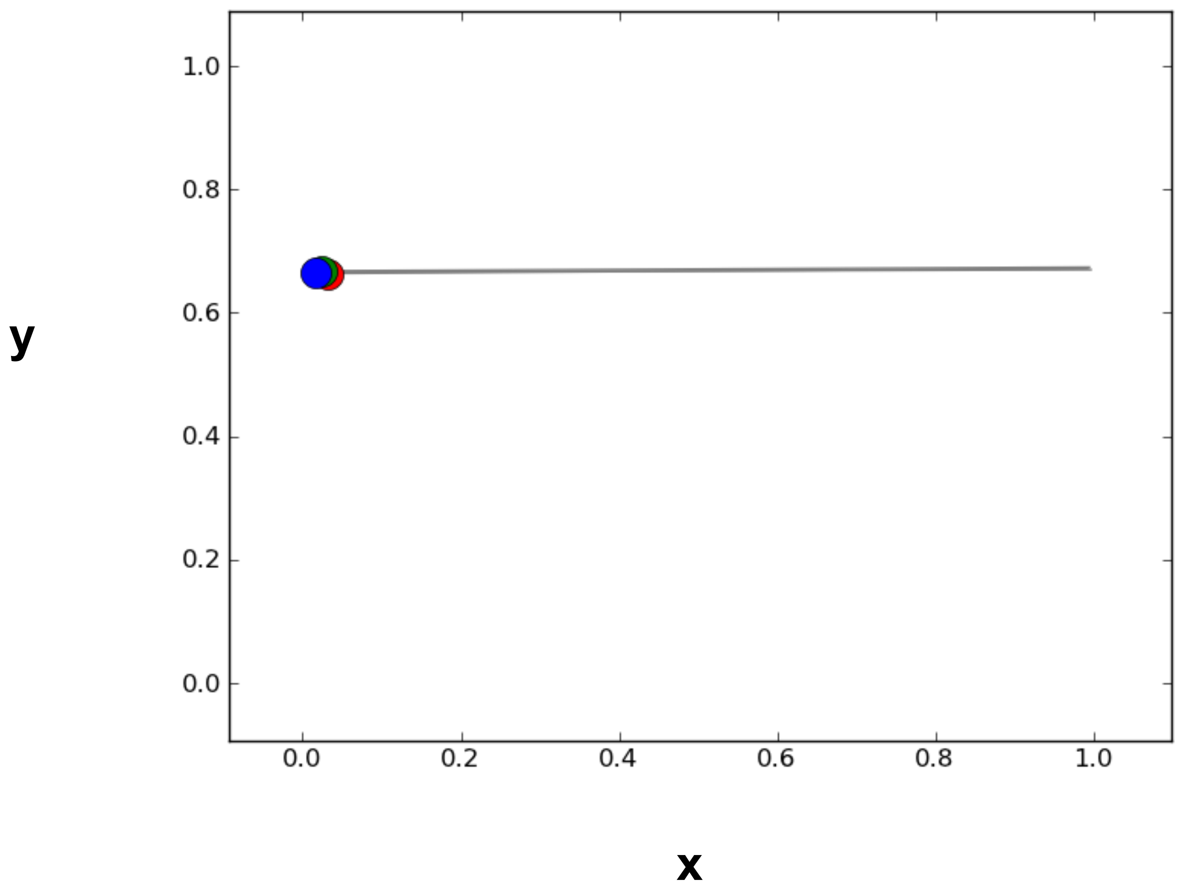
- Topological mixing
  - Complicated mathematical definition
  - Basically it means that things 'jump all over the place' – dimensions mix apparently randomly

Mixing Example
logistics equation

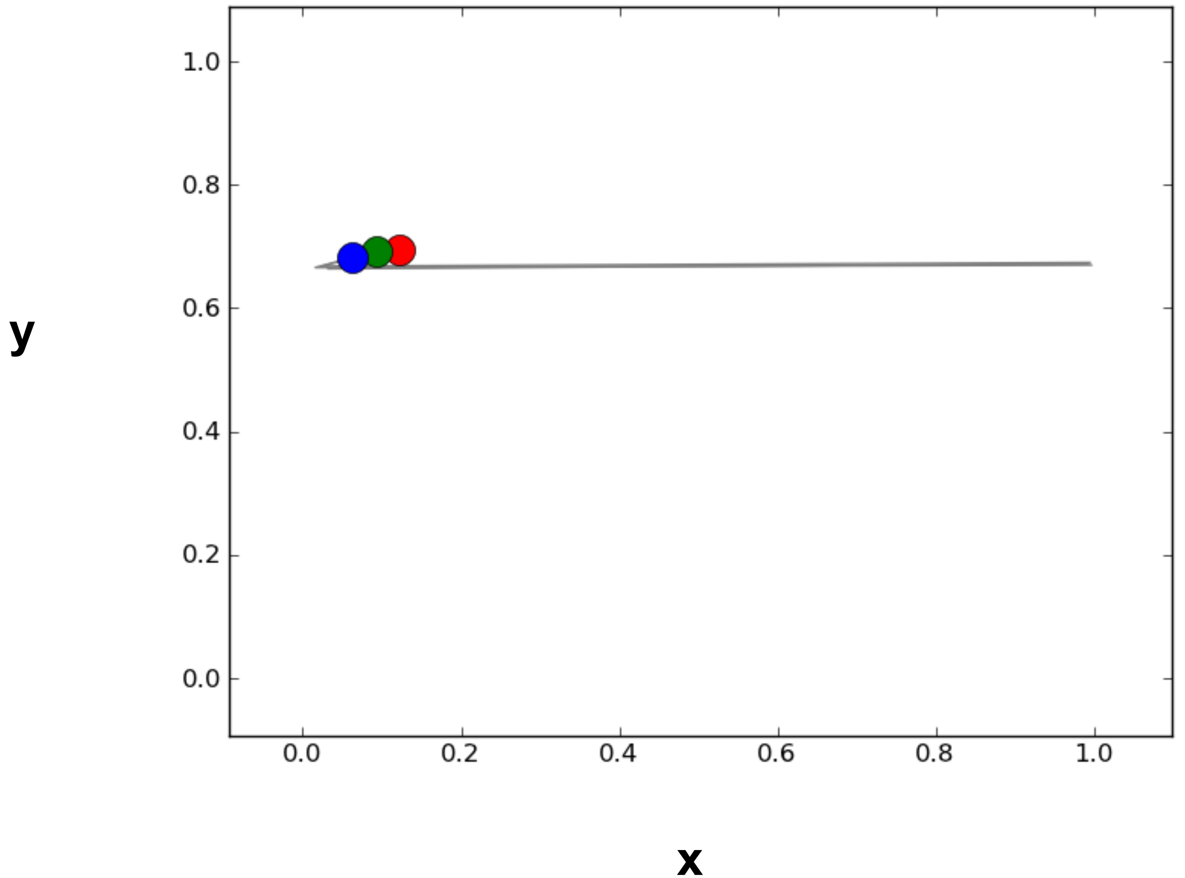
```
def f(x, y):
  x1 = 4 * x * (1-x)
  if x + y < 1:
    y1 = x + y
  else:
    y1 = x + y - 1
  return x1, y1
```



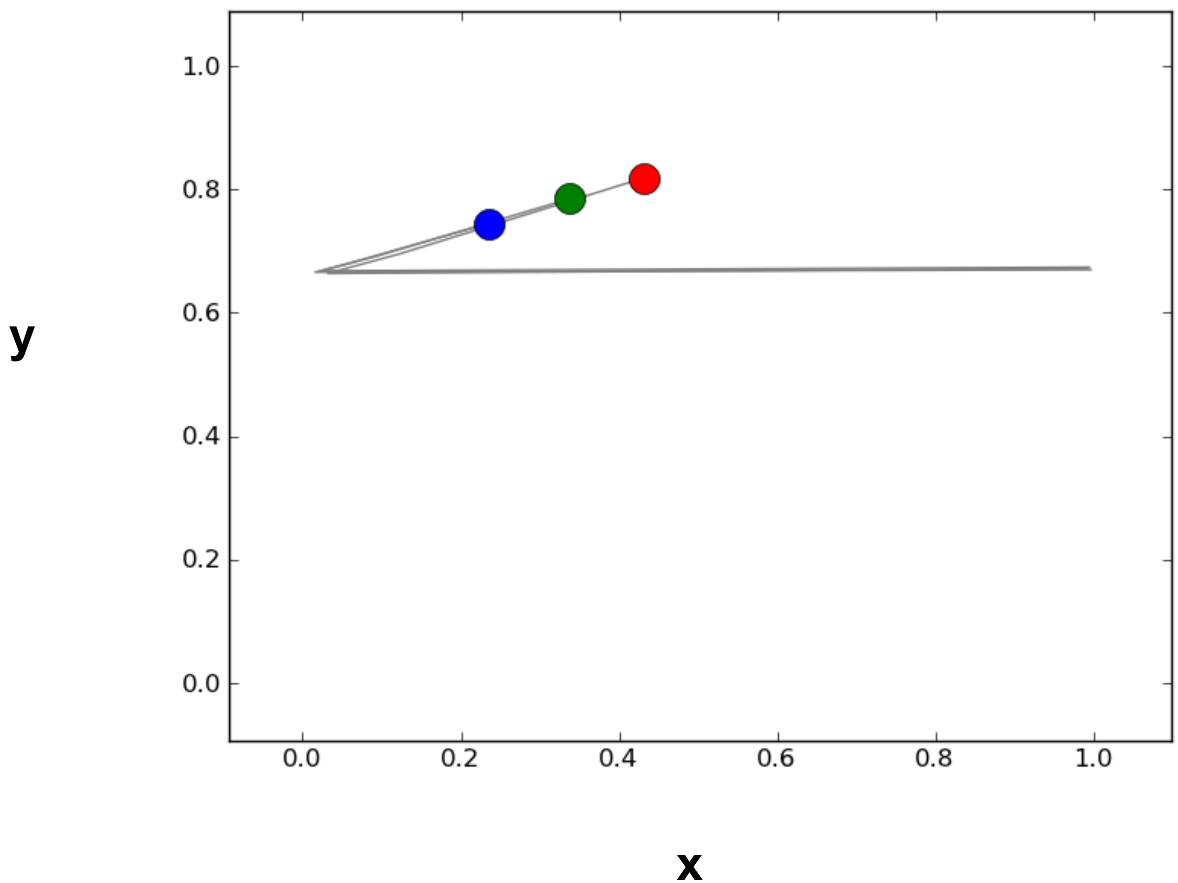
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
y
x



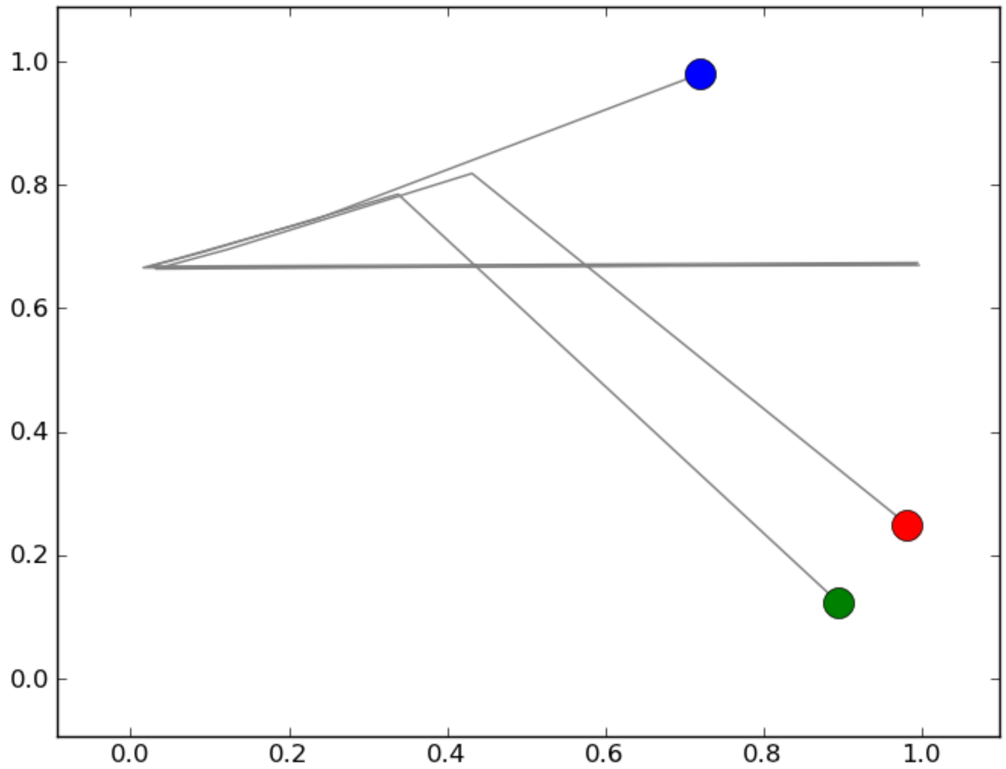
y
x
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0



y
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
x

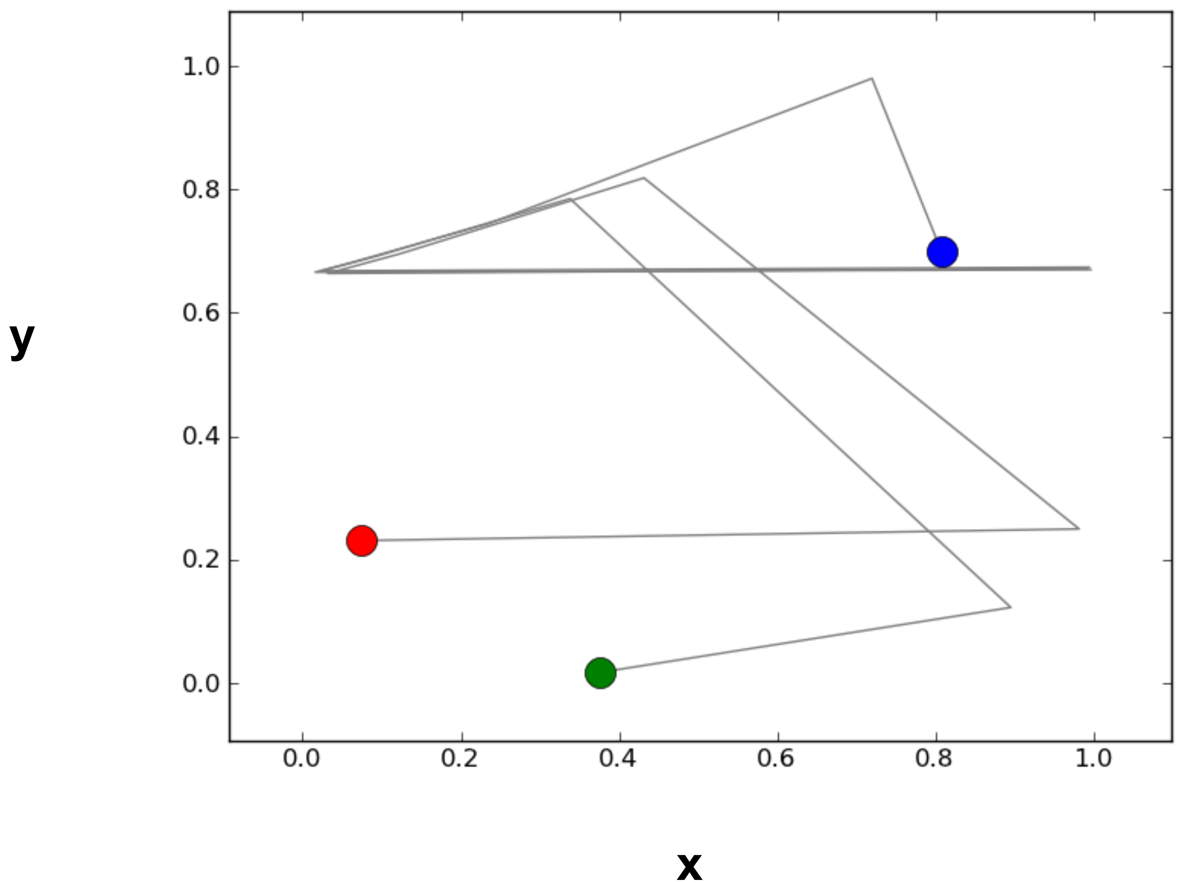


1.0
0.8
0.6
0.4
0.2
0.0
y
0.0
0.2
0.4
0.6
0.8
1.0
x

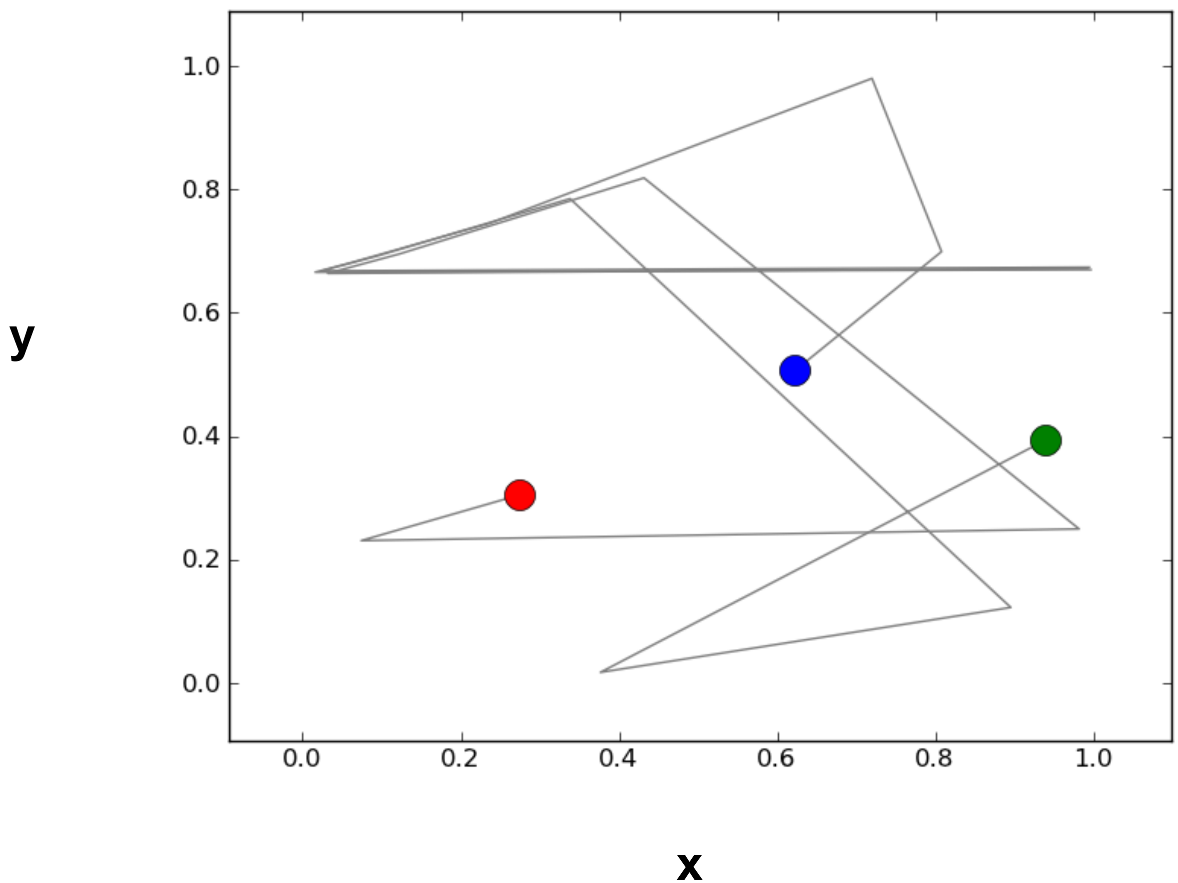


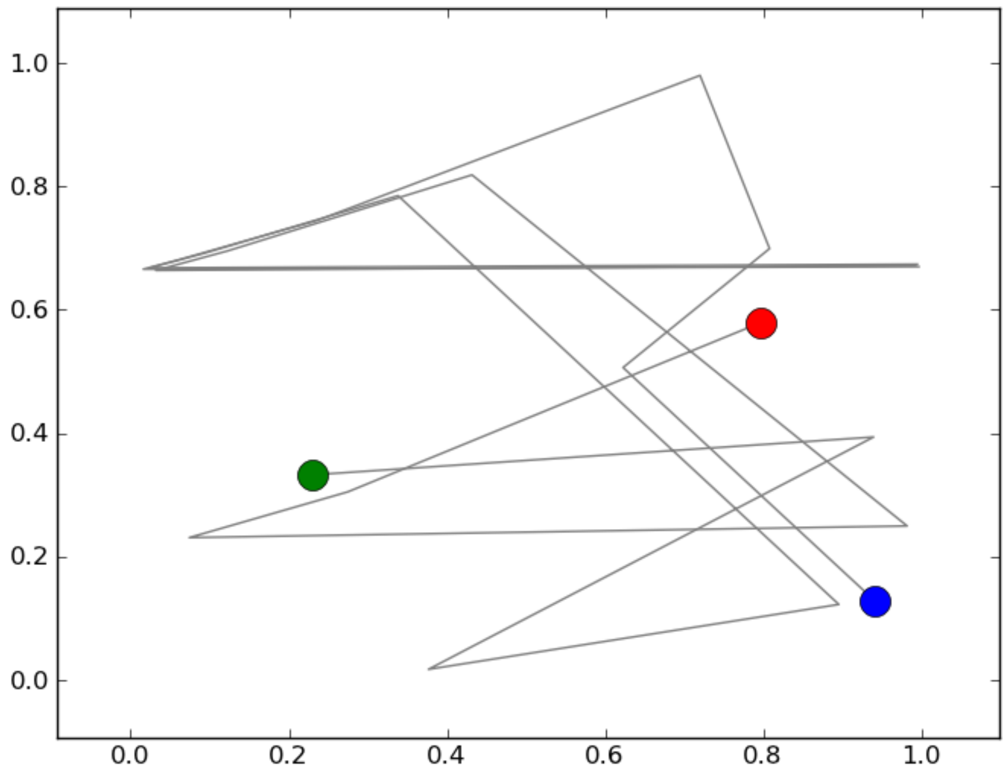
y

x



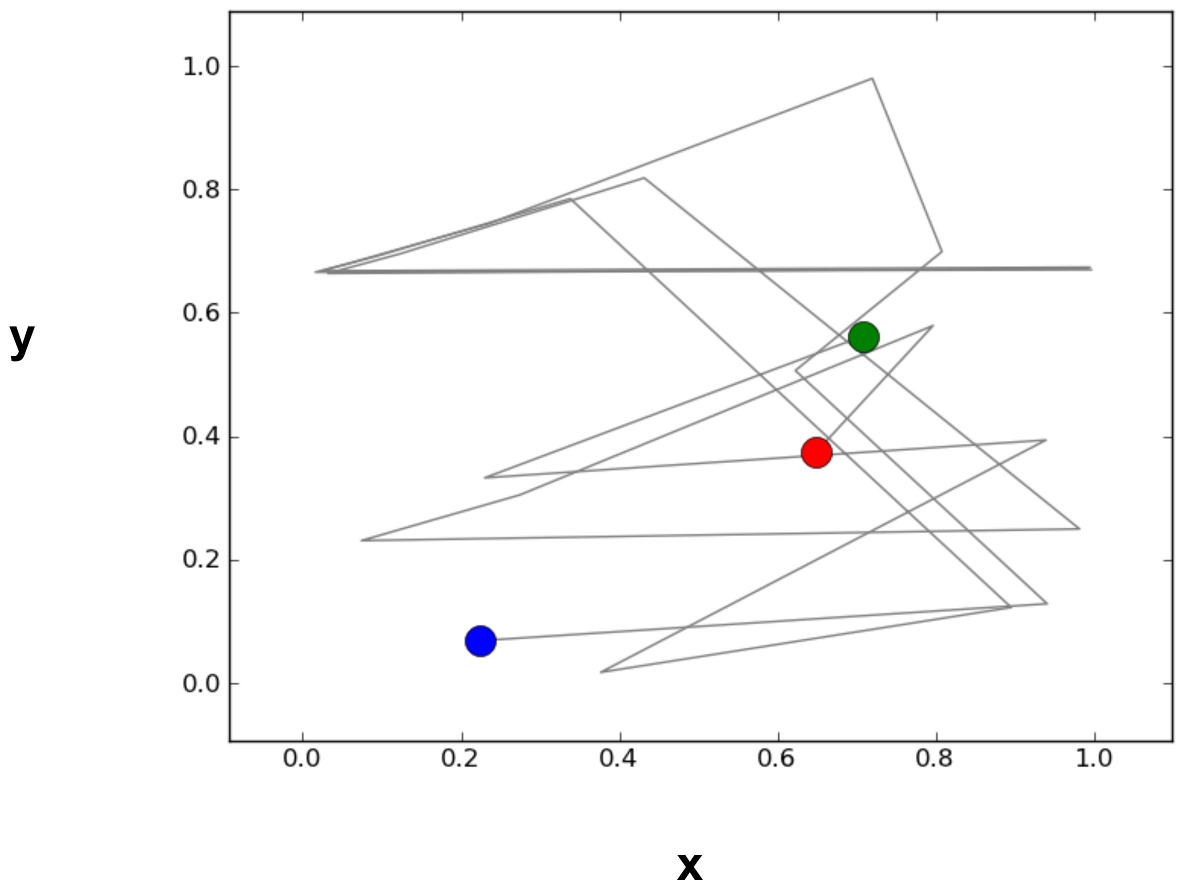
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
y
x


y
x
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

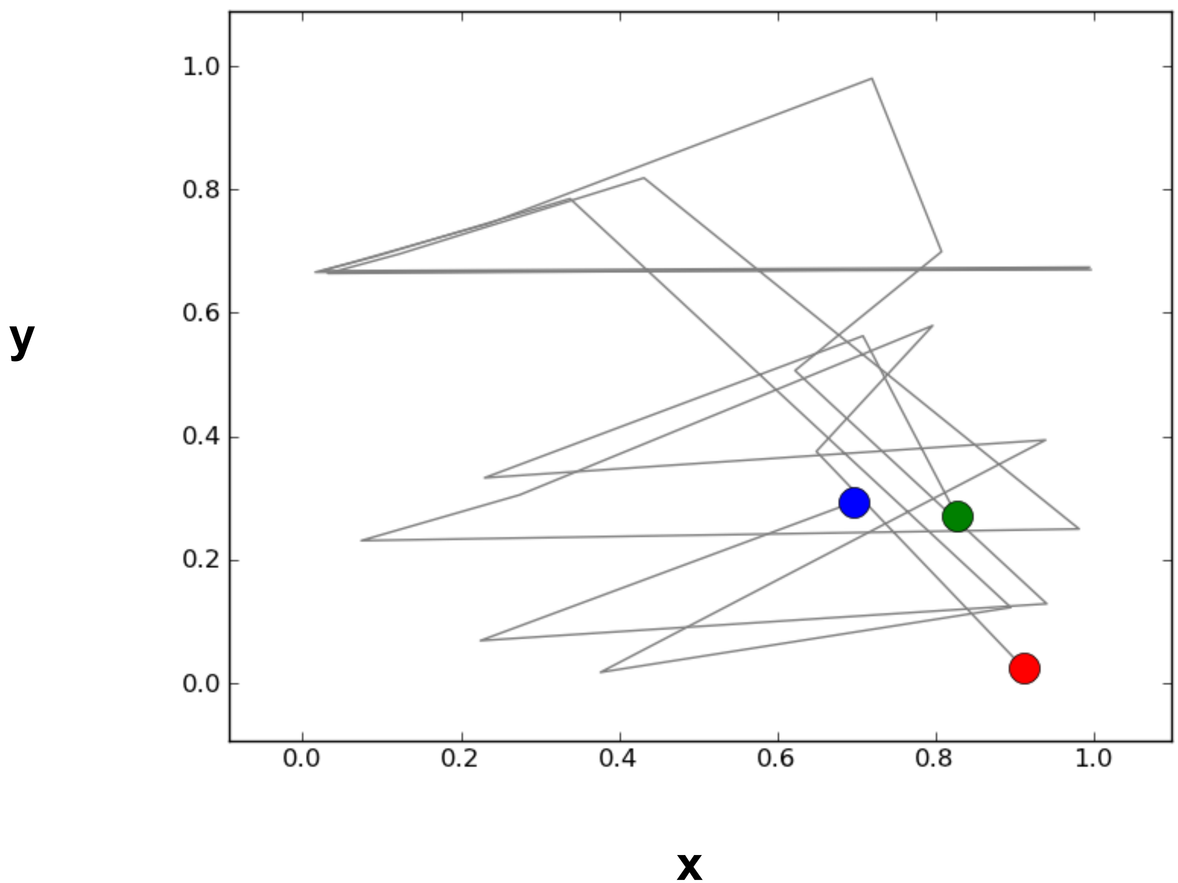


y

x



1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
y
x



y
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0
x

Chaotic System : Recap

- Deterministic
- Sensitive to initial conditions
- Topological mixing

The driven pendulum

- Pendulum

$$\frac{d^2\theta}{dt^2} = -\frac{g}{l}\sin(\theta)$$

- Now add a periodic driving force of frequency ω_d

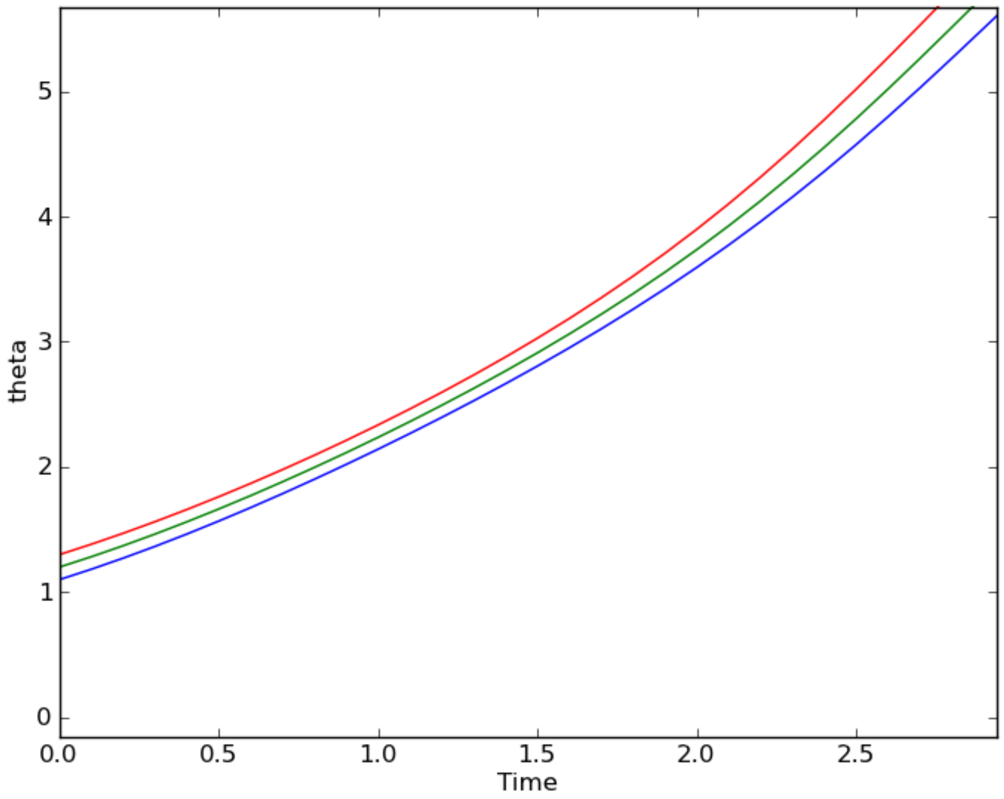
$$\frac{d^2\theta}{dt^2} = -\frac{g}{l}\sin(\theta) + A\cos(\omega_d t)$$

driven1.py – /Users/cds/cds/Teaching/2014-2015/L2_CompPhys/Week 8 – Ch...

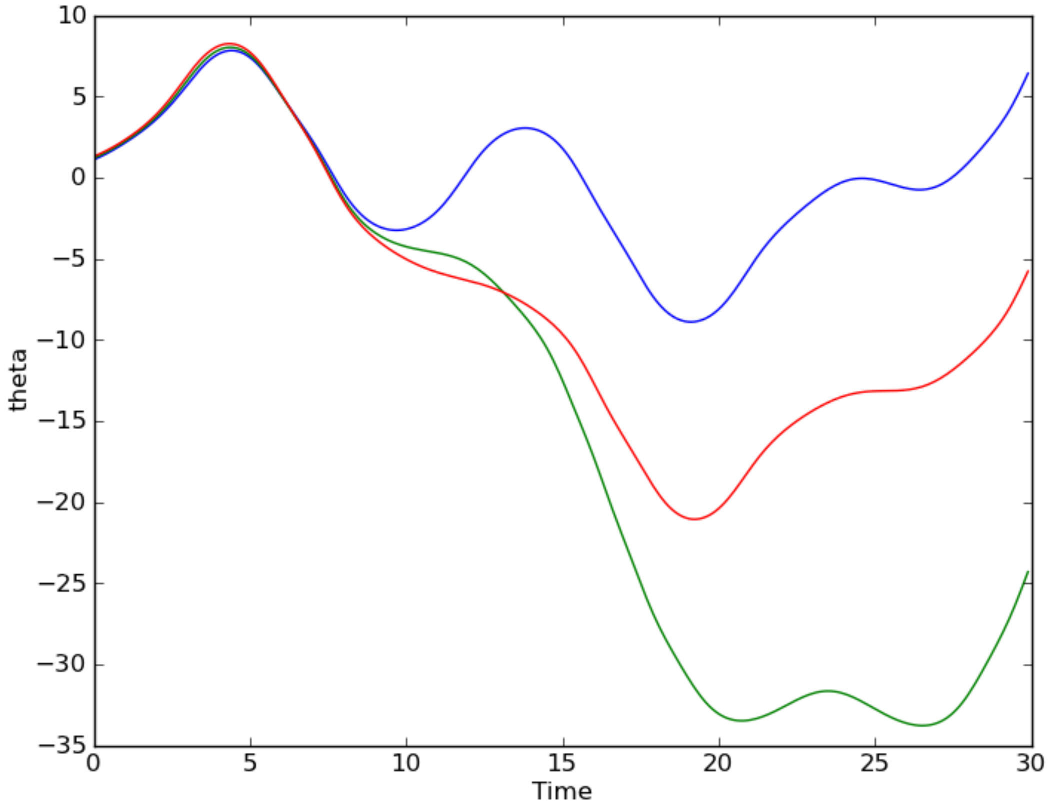
```python
from __future__ import division
import numpy
import scipy.integrate
import matplotlib.pyplot as pyplot


A = 1.5 # 1.5 is good
omega_d = 2/3
def f((theta, omega), t):
    ''' DEQ for a driven pendulum '''
    domega = - numpy.sin(theta) + A*numpy.cos(omega_d*t)
    dtheta = omega
    return dtheta, domega

def sim(theta0, omega0):
    ''' Wrapper function for the pendulum DEQ that generates
    a timebase, invokes ODEINT and seperates the return
    variables '''
    timebase = numpy.arange(0, 30, 0.1)

    x = scipy.integrate.odeint(f, (theta0, omega0), timebase)
    theta, omega = x[:,0], x[:,1]

    return timebase, theta, omega

t0, theta0, omega0 = sim(1.10, 0.8)
t1, theta1, omega1 = sim(1.20, 0.8)
t2, theta2, omega2 = sim(1.30, 0.8)

def modfunc(x): # Phase wrapping
    return (x + numpy.pi) % (2*numpy.pi) - numpy.pi

pyplot.figure()
pyplot.plot(t0, modfunc(theta0))
pyplot.plot(t1, modfunc(theta1))
pyplot.plot(t2, modfunc(theta2))
pyplot.xlabel('Time')
pyplot.ylabel('theta')
pyplot.show()
```

Ln: 30 Col: 32

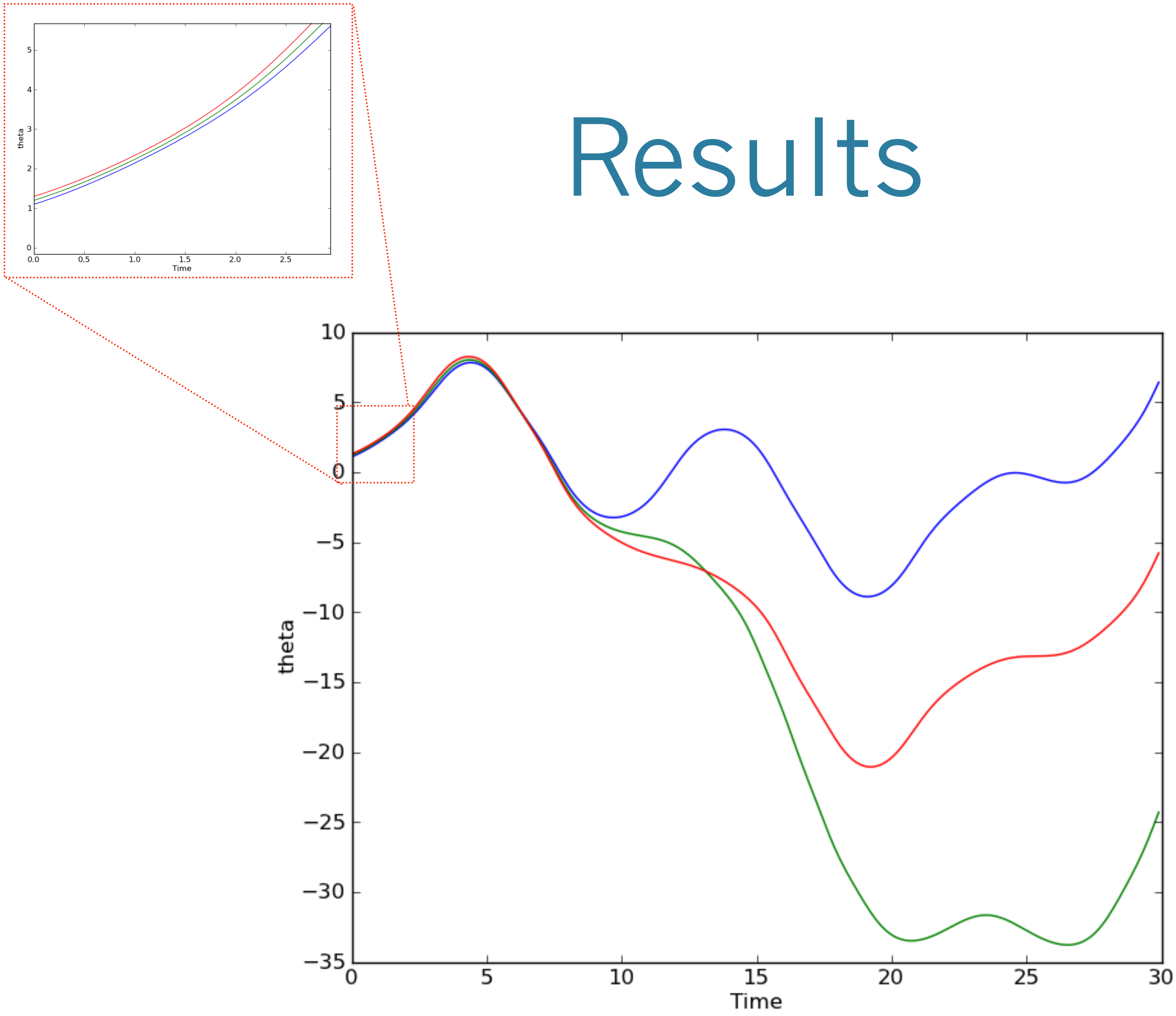
Results



Results

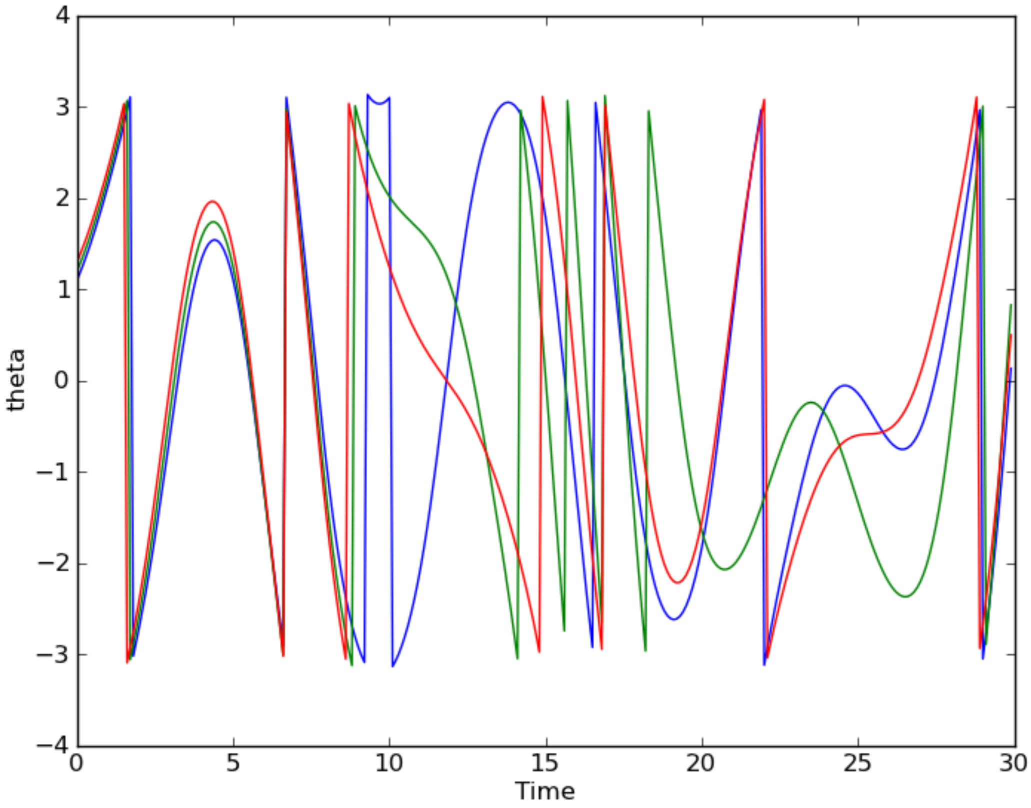


Results



theta
Time
10
5
0
−5
−10
−15
−20
−25
−30
−35
0
5
10
15
20
25
30
0.0
0.5
1.0
1.5
2.0
2.5

Phase Wrapping



‘Butterfly Effect’

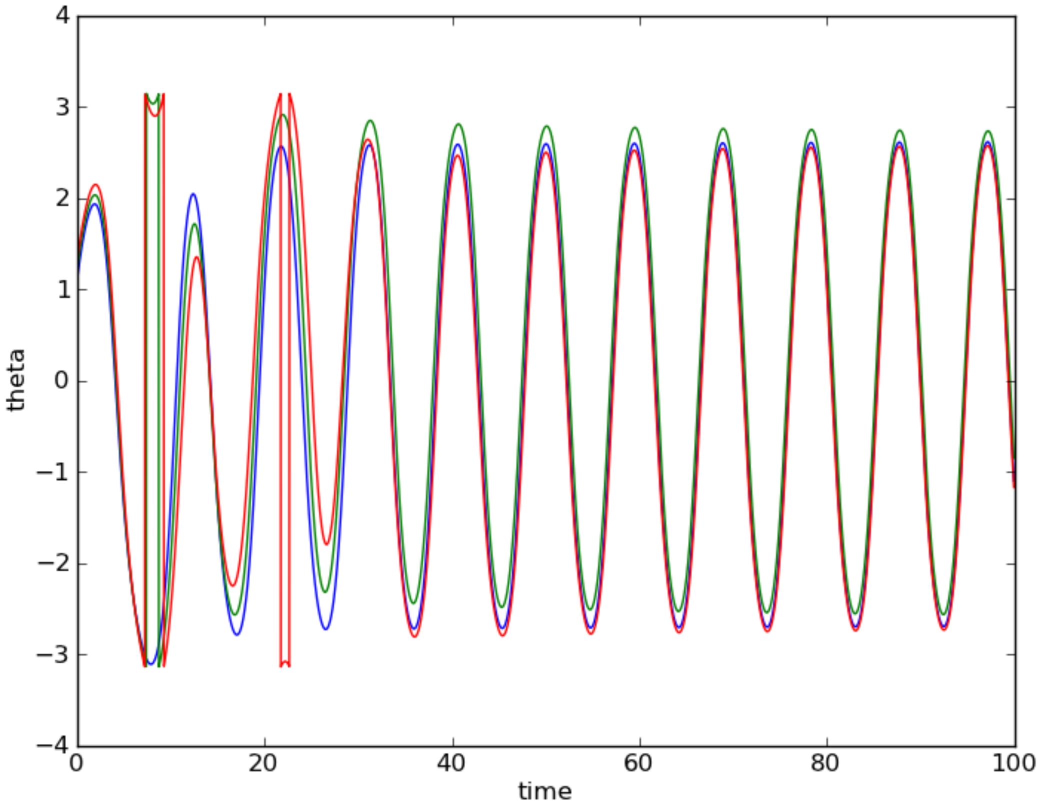
- A tiny change in the initial conditions has a large impact on the state at later times
- Dramatic implications for numerical modelling
  - Weather
  - Stock Market
  - N-body dynamics

Damped, Driven Pendulum

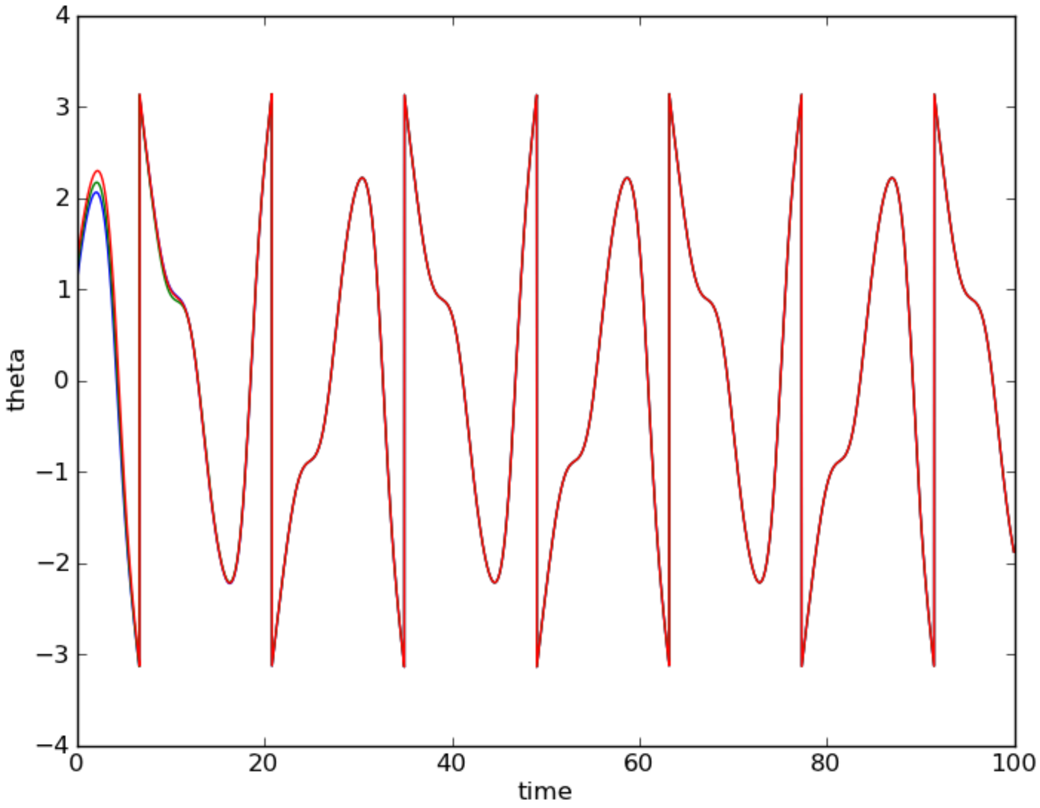
- periodic driving force
  - frequency ω_d
  - amplitude A
- Linear damping, q

$$\frac{d^2\theta}{dt^2} = -\frac{g}{l}\sin(\theta) + A\cos(\omega_d t) - \frac{1}{q}\frac{d\theta}{dt}$$

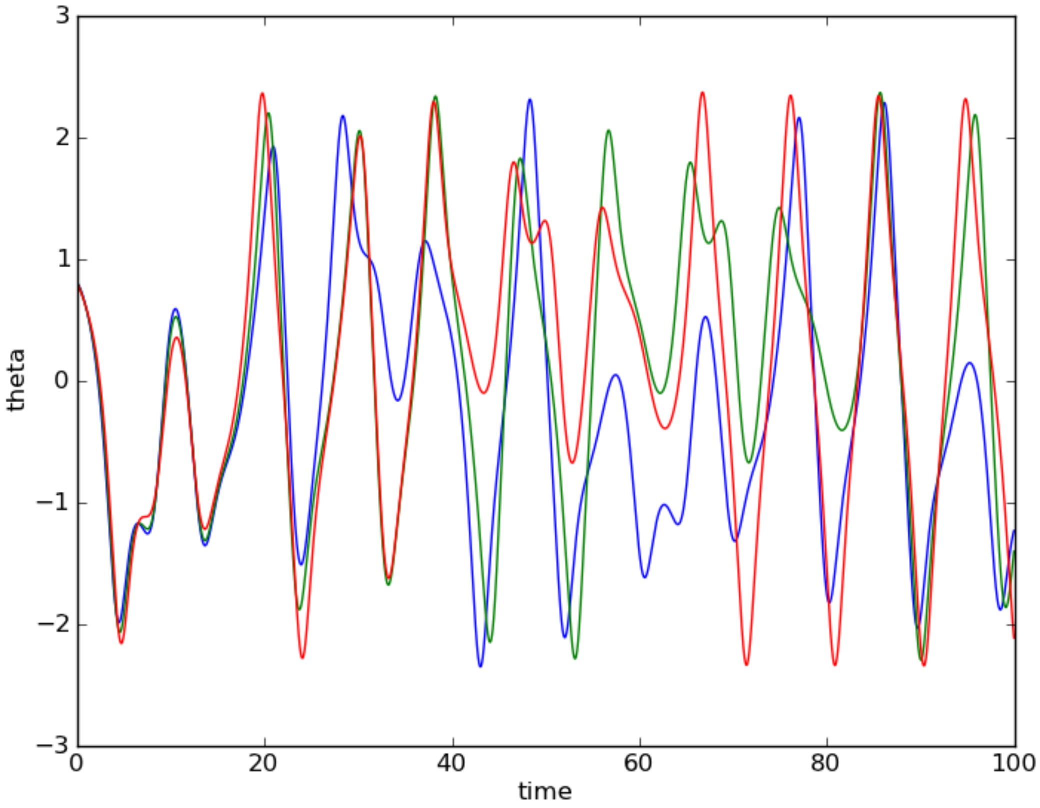
A=1.0



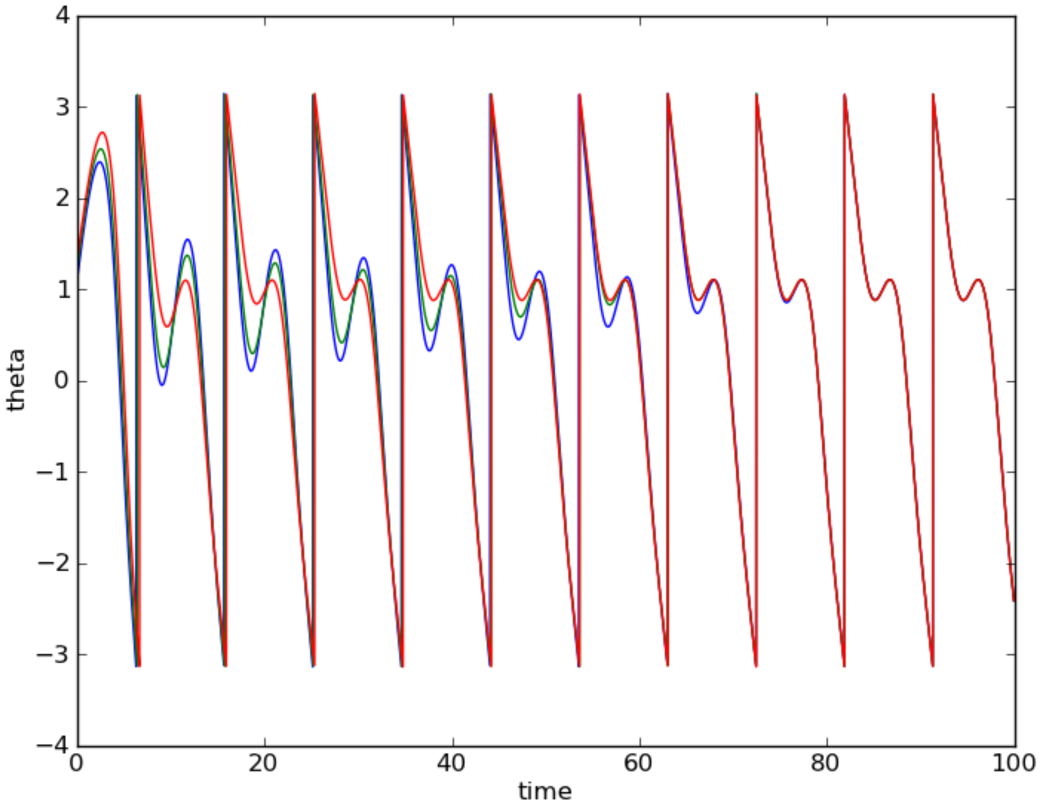
A=1.1



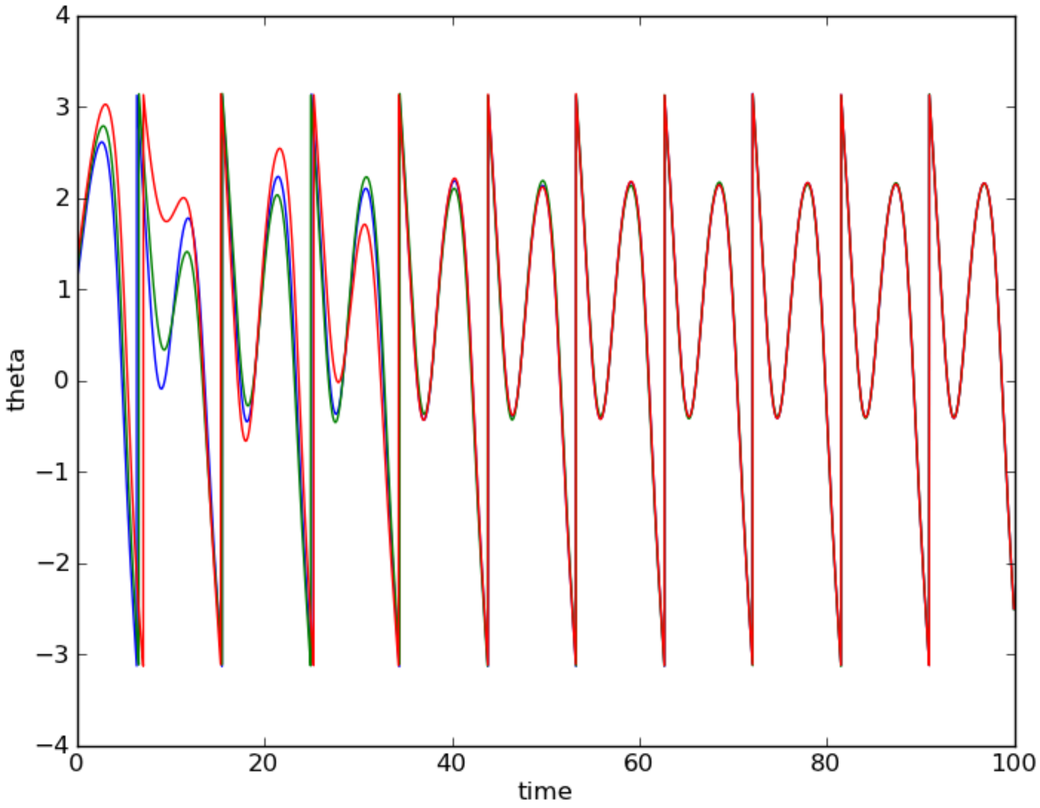
A=1.2



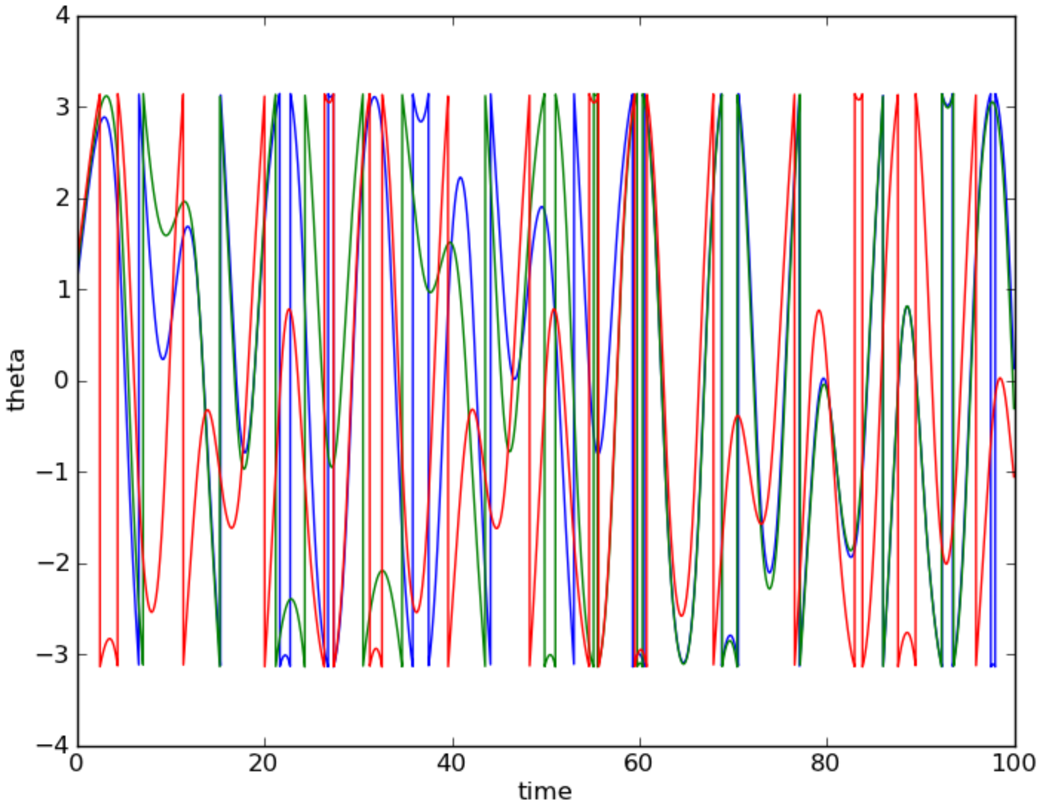
A=1.3



A=1.4



A=1.5



Butterfly Effect

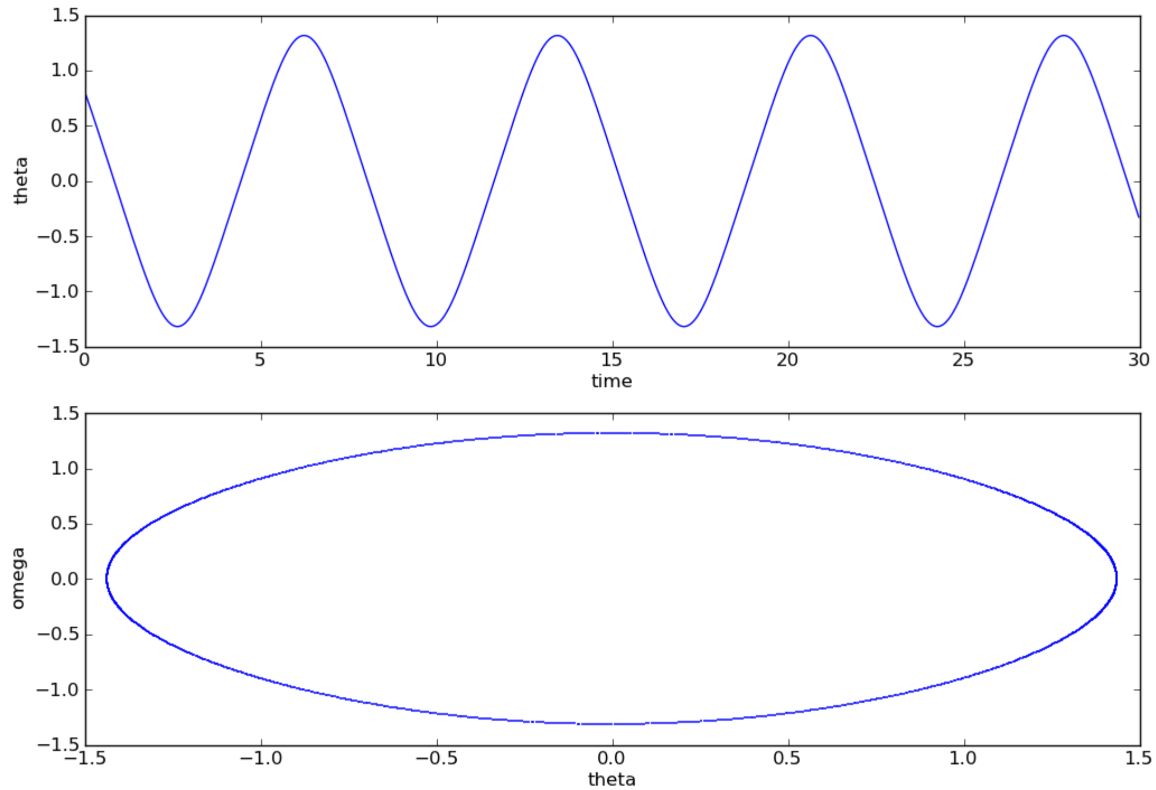
- Butterfly Effect is still present with damping!

- Even though you might naively expect damping would decrease the importance of initial conditions over time

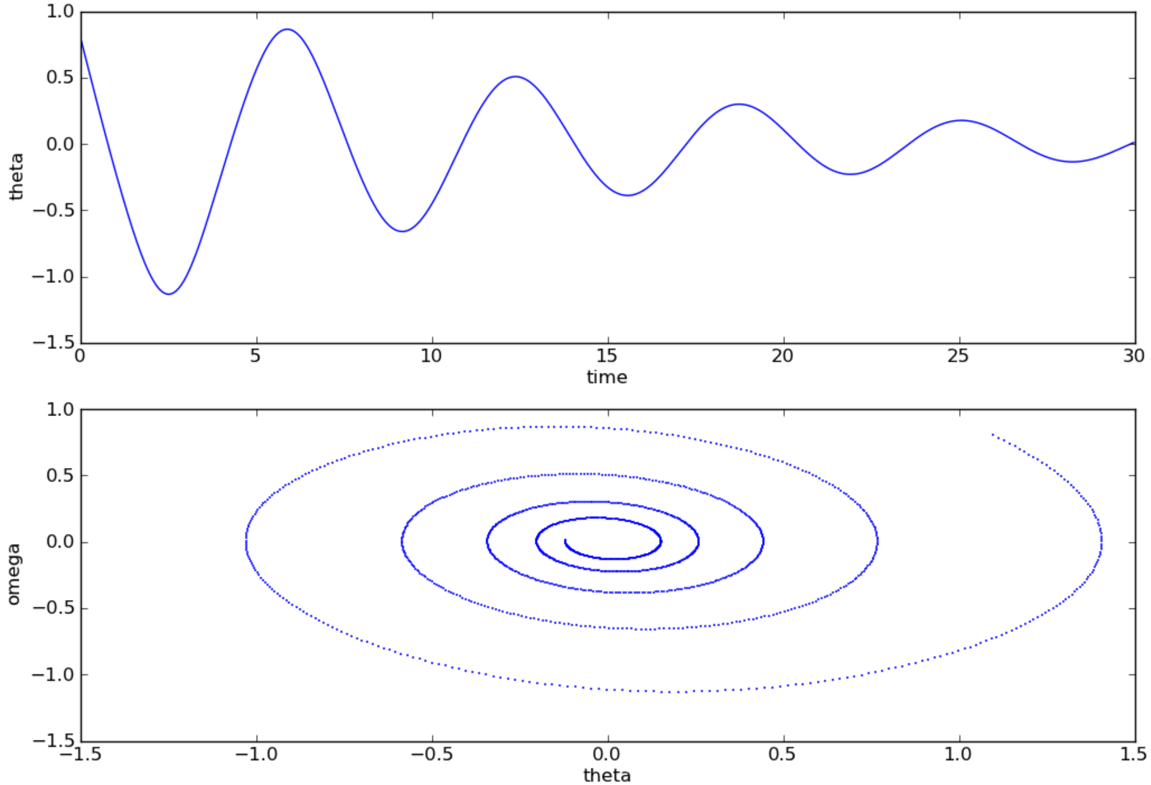
Recap: Phase Space

- Multidimensional space
- One dimension for each variable composing system state
- Typically position & momentum/velocity
- Pendulum
  - Angular displacement
  - Angular velocity

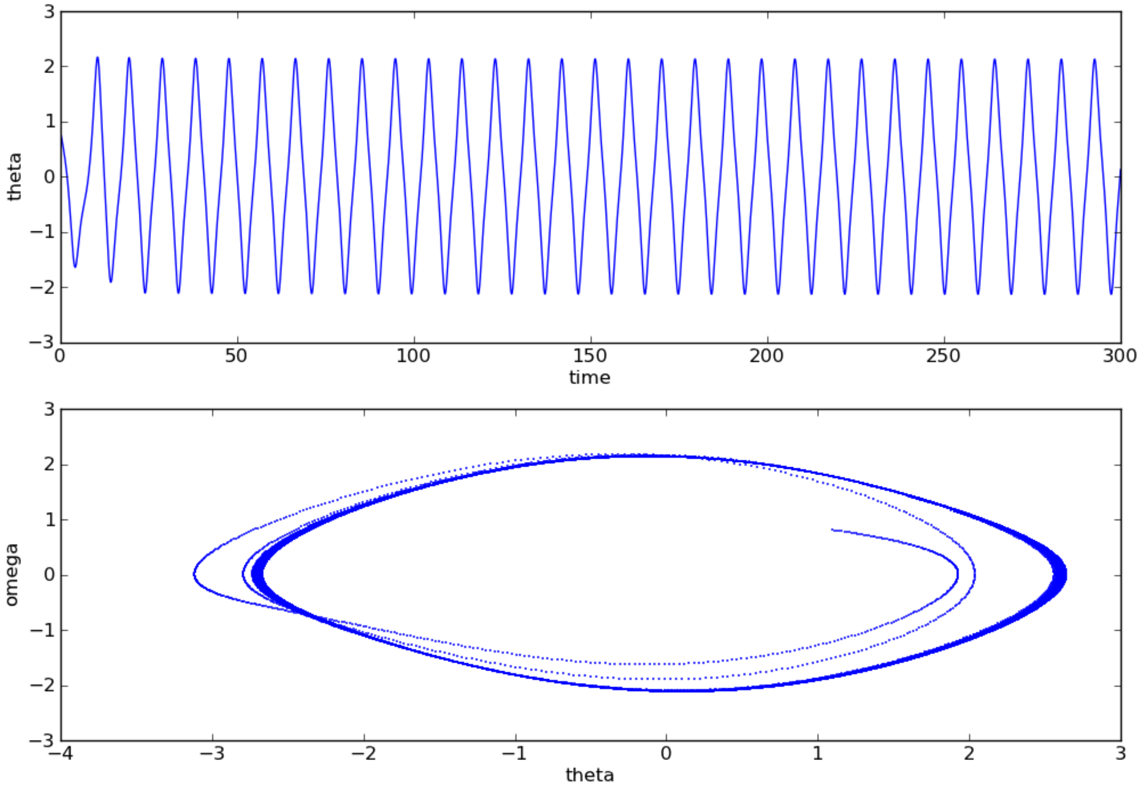
Phase Space: Pendulum



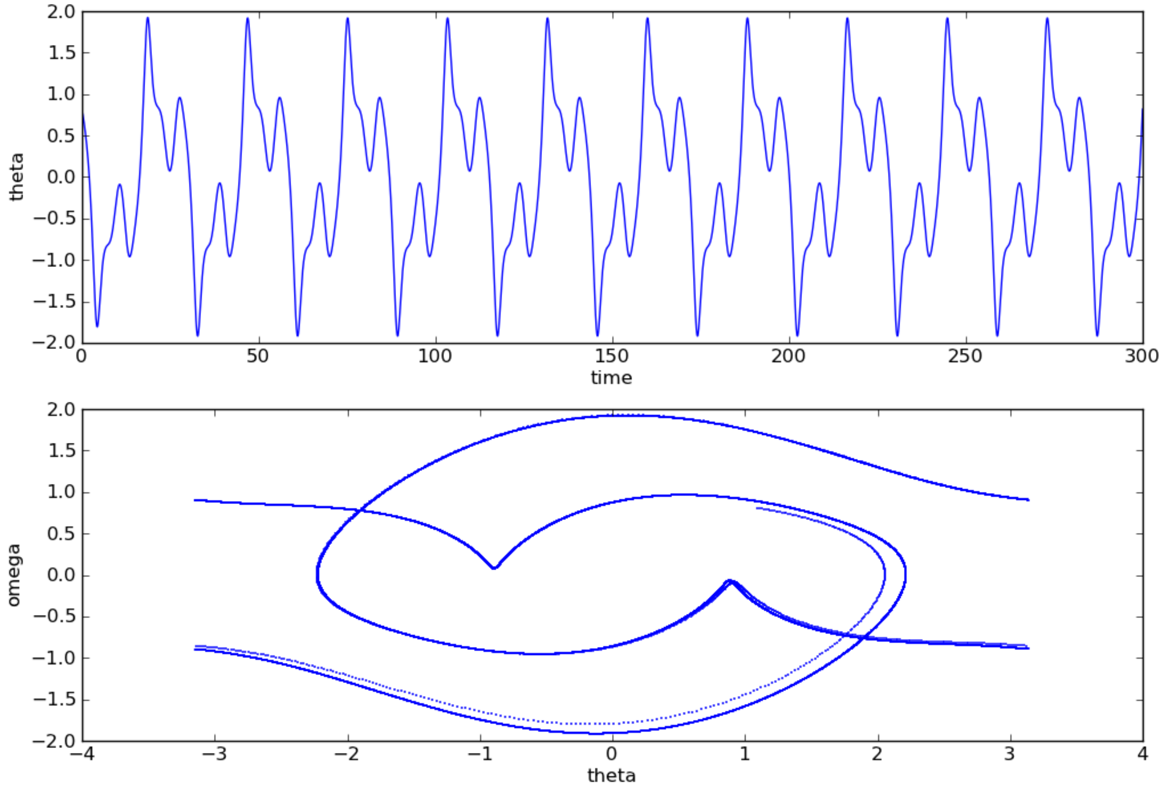
Phase Space – Damped Pendulum



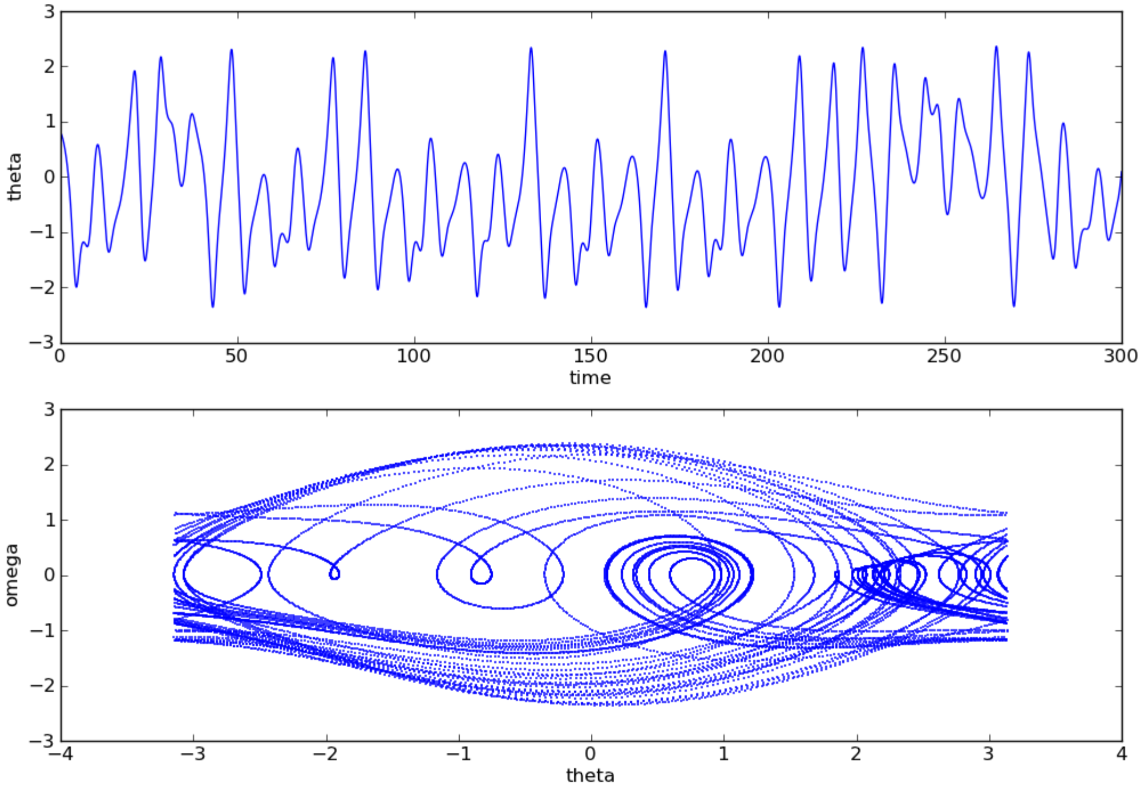
Damped, Driven – A = 1.0



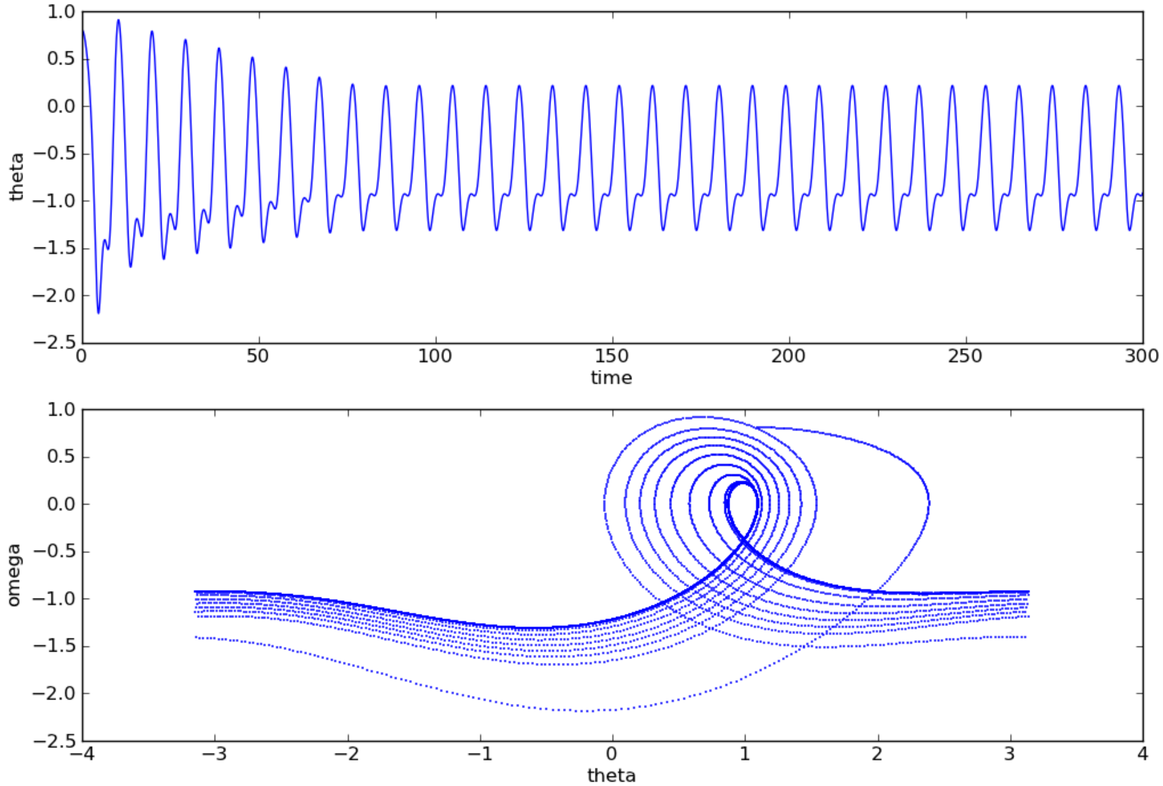
Damped, Driven – A = 1.1



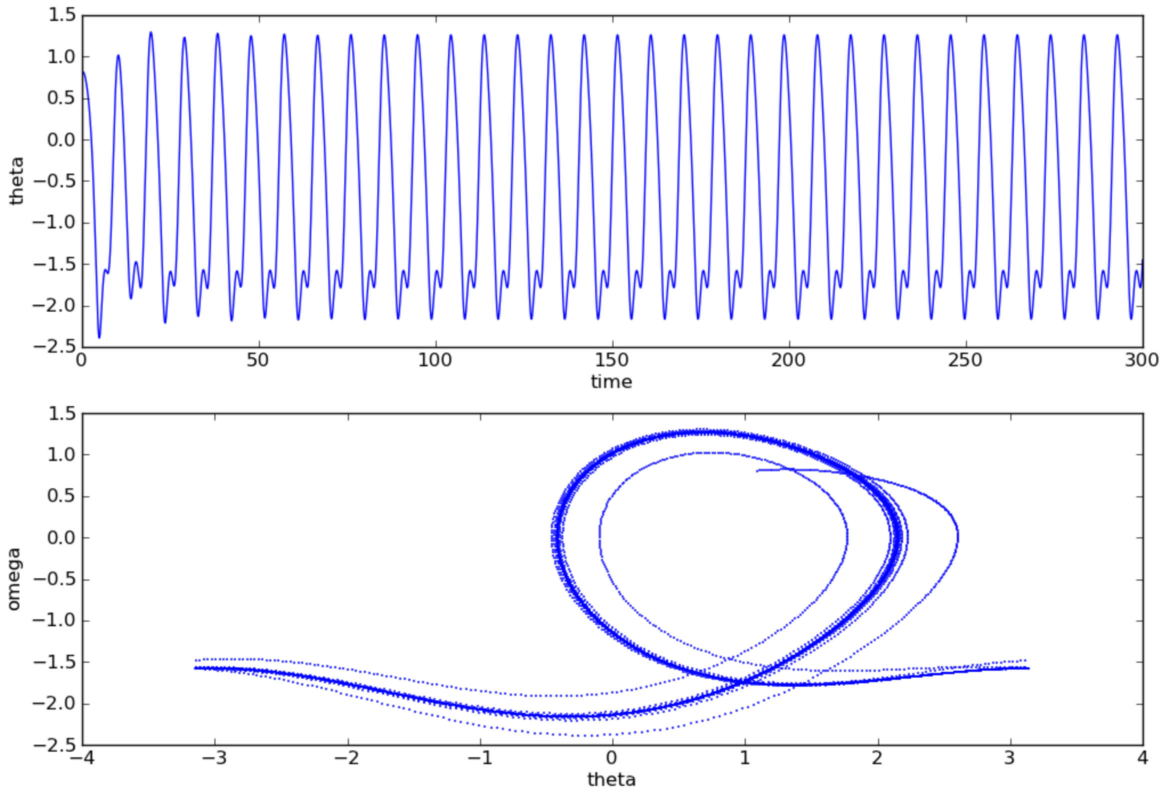
Damped, Driven – A = 1.2



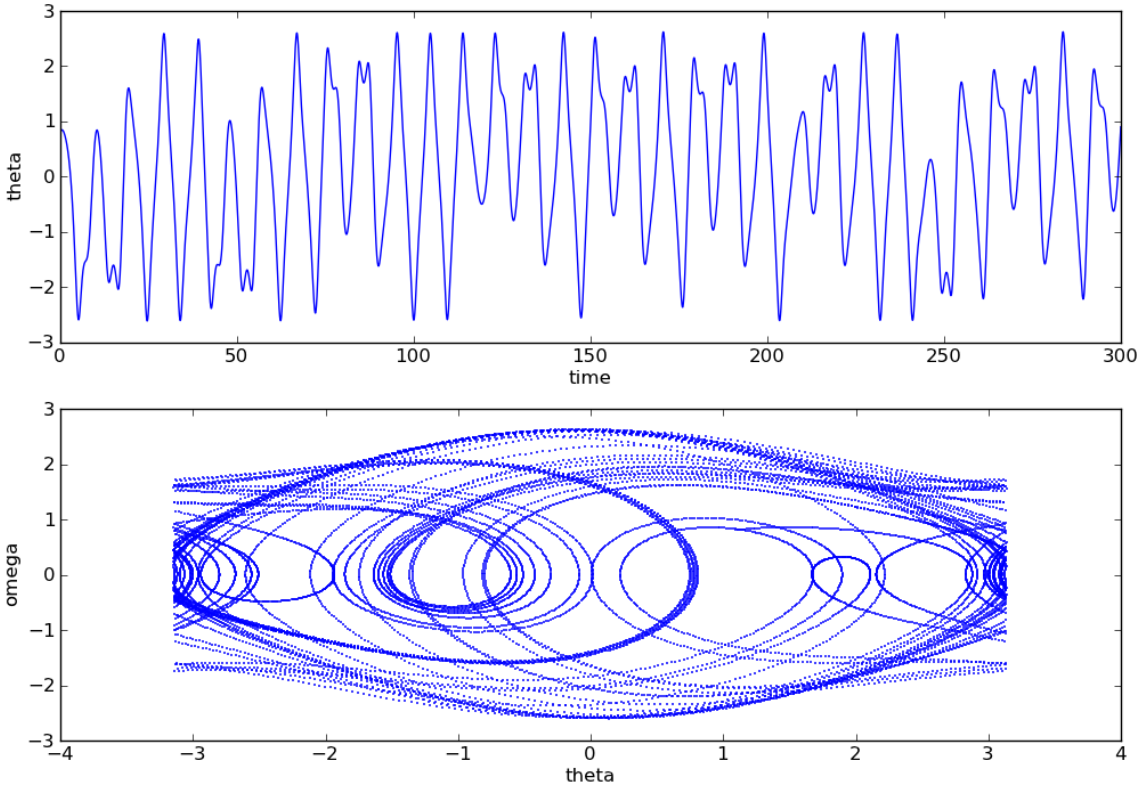
Damped, Driven – A = 1.3



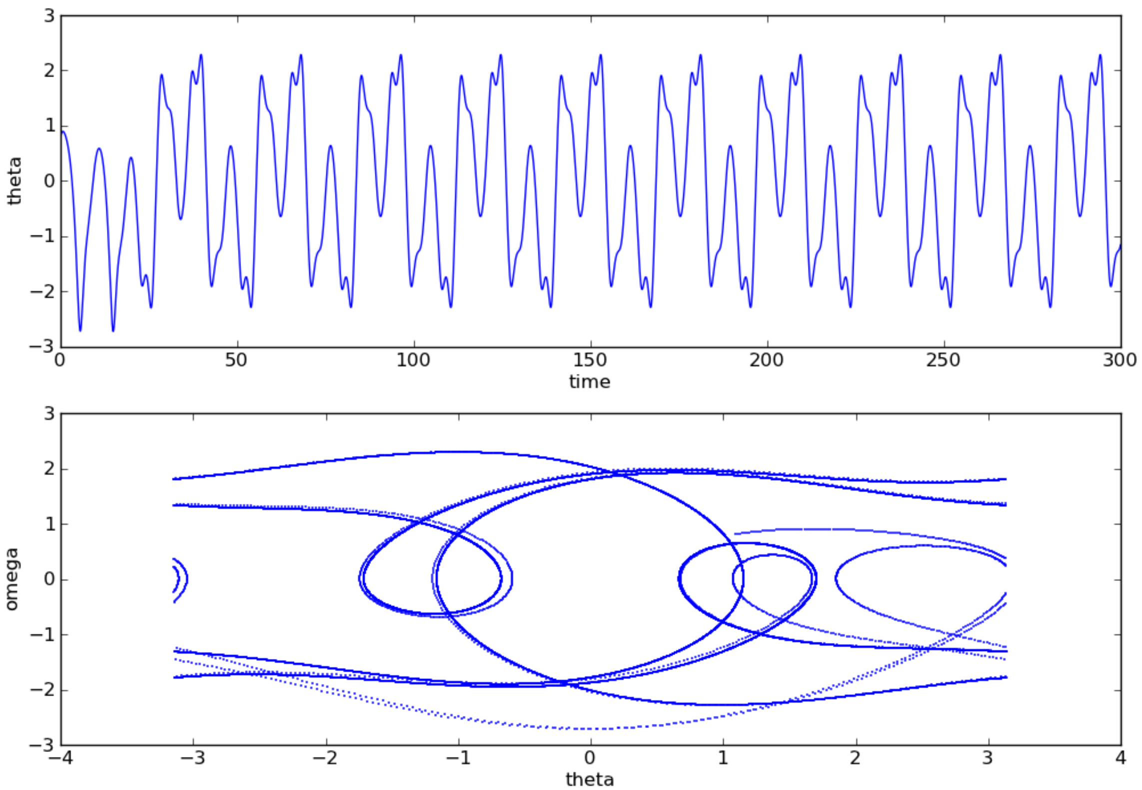
Damped, Driven – A = 1.4



Damped, Driven – A = 1.5



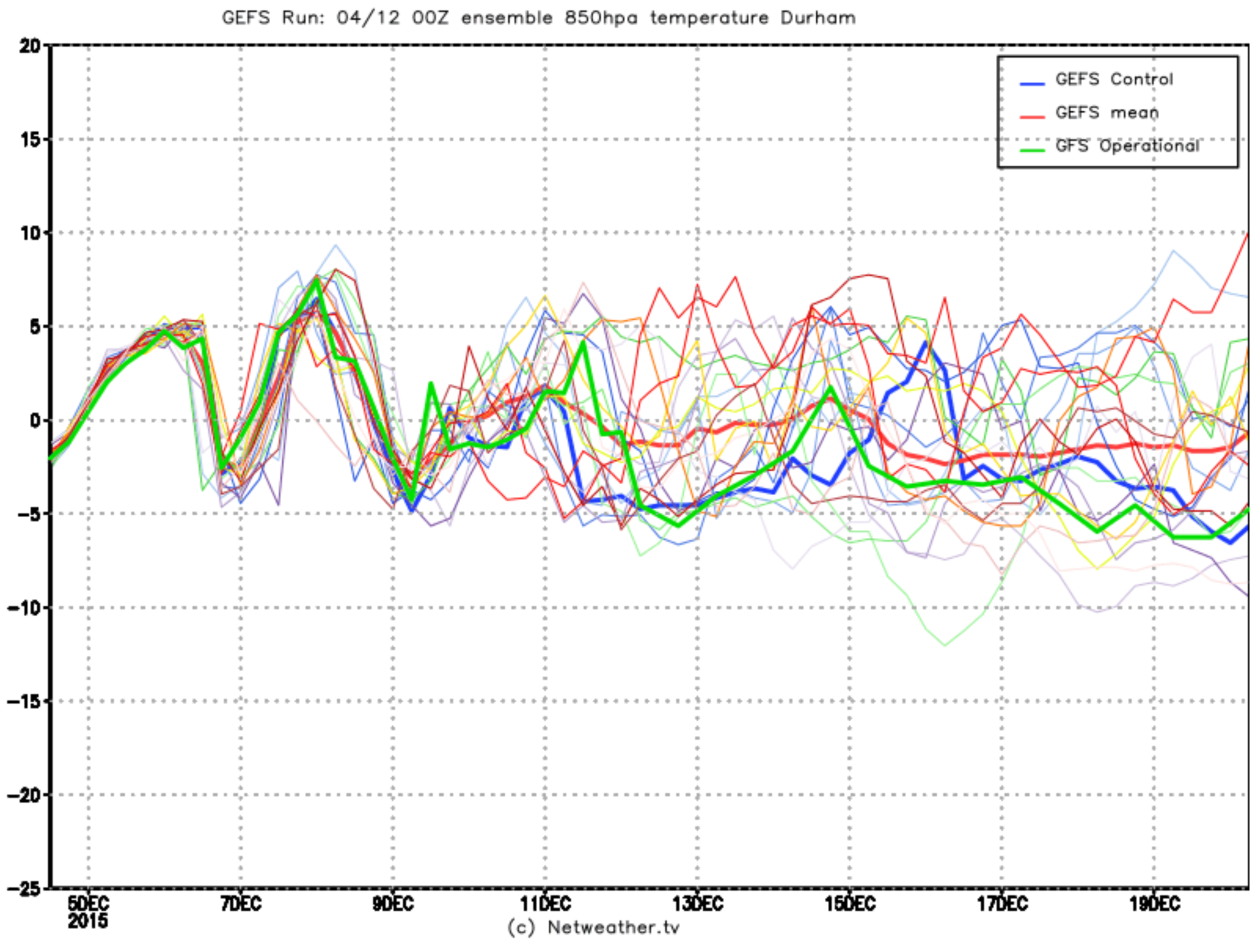
Damped, Driven – A = 1.6



Chaotic Weather

- NWP: Numerical Weather Prediction
  - Divide atmosphere into 3D grid of points
  - For each point write DEQs:
    - Heat transfer (conduction)
    - Solar irradiance / absorption
    - Mass flow (wind, moisture)
    - Etc. etc.
  - Solve numerically
- Global forecast models on an XY grid of ~40km
- Small changes in inputs -> chaos

US GFS temperature at 850hpa line



Fractals

Fractal

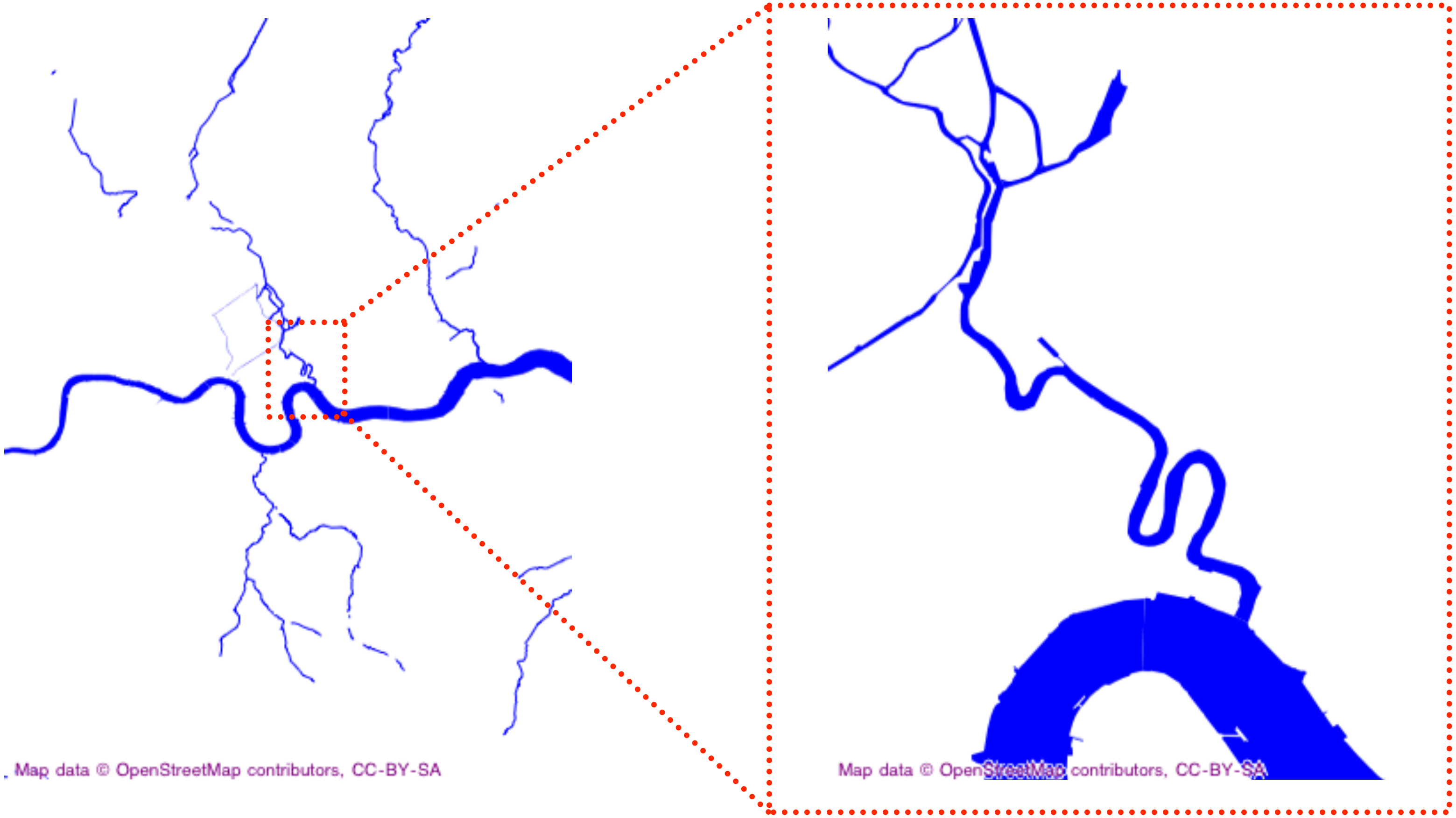
- A fractal is a specific class of geometric shapes

- A shape is deemed fractal if it has *self similarity*
  - Scale invariant
  - It looks the same at different scales

- It may be self similarity off
  - Appearance
  - Statistics
  - Some other feature

Fractals and Nature

- Many natural processes form shapes with fractal properties
  - River systems
  - Coast lines
  - Lightning
  - Snow flakes
  - Many more



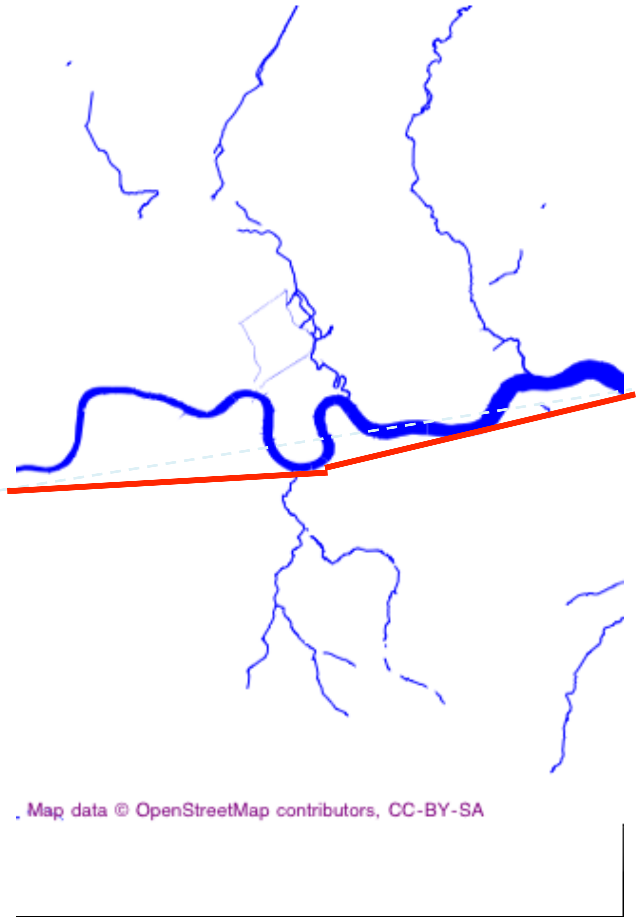
Map data © OpenStreetMap contributors, CC-BY-SA
Map data © OpenStreetMap contributors, CC-BY-SA

Fractal Dimension

- The self-similarity of fractals means they contain endless detail – the more you zoom in, the more detail there is
- This means there is no answer to questions like "How long is my fractal"
- The measured length depends upon the scale you measure on

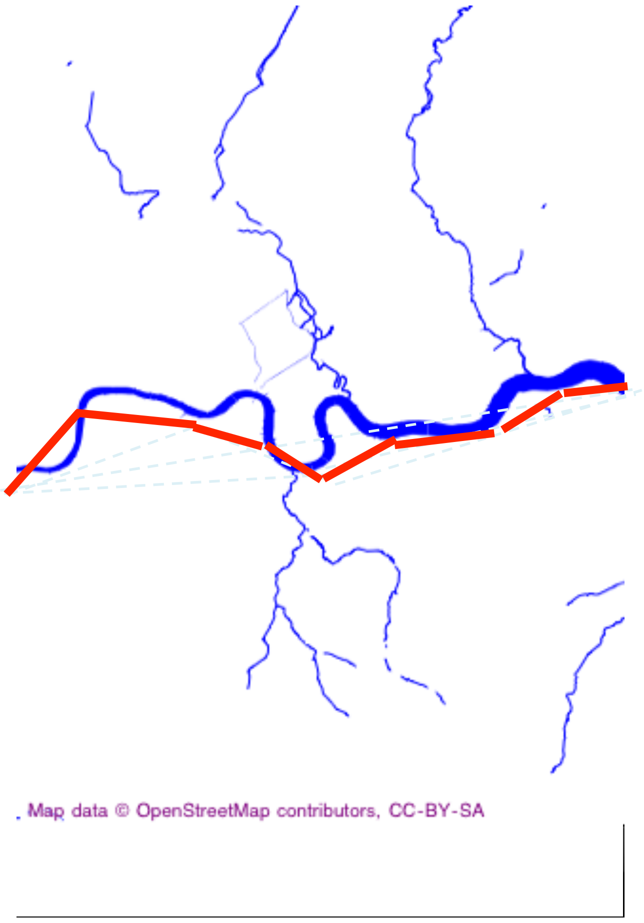
Example: Rivers

Example: Rivers



Example: Rivers

Example: Rivers



Fractal Dimension

- The smaller our measurement scale, 'G', the greater our measured length, 'L'

$$L(G) \propto G^{1-D}$$

- 'D' is the 'Hausdorff dimension' or 'fractal dimension' – it is some constant for a given geometry that characterises its scaling

Random Fractals

Escape-time fractals

Iterated function systems

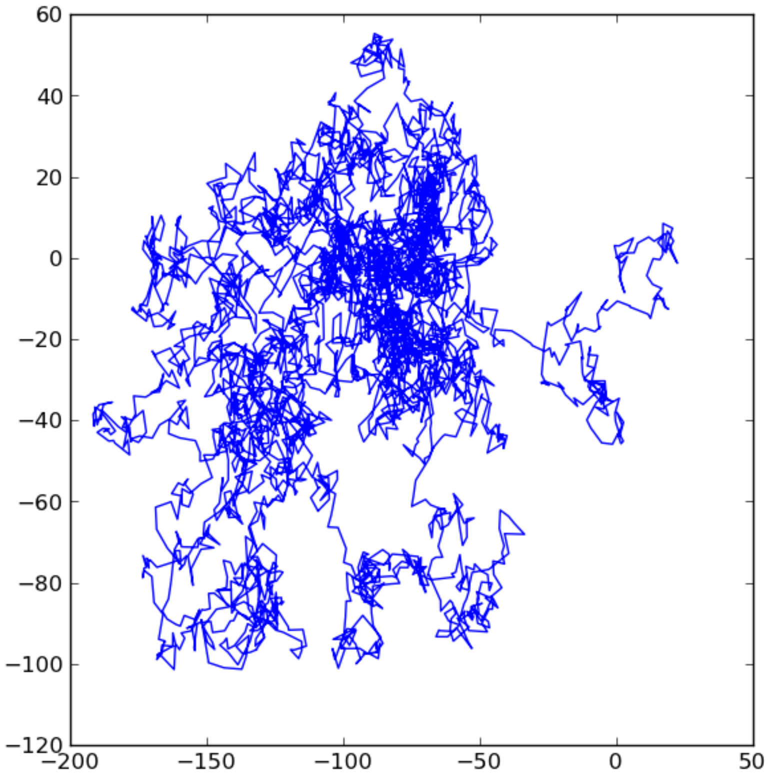
Strange Attractors

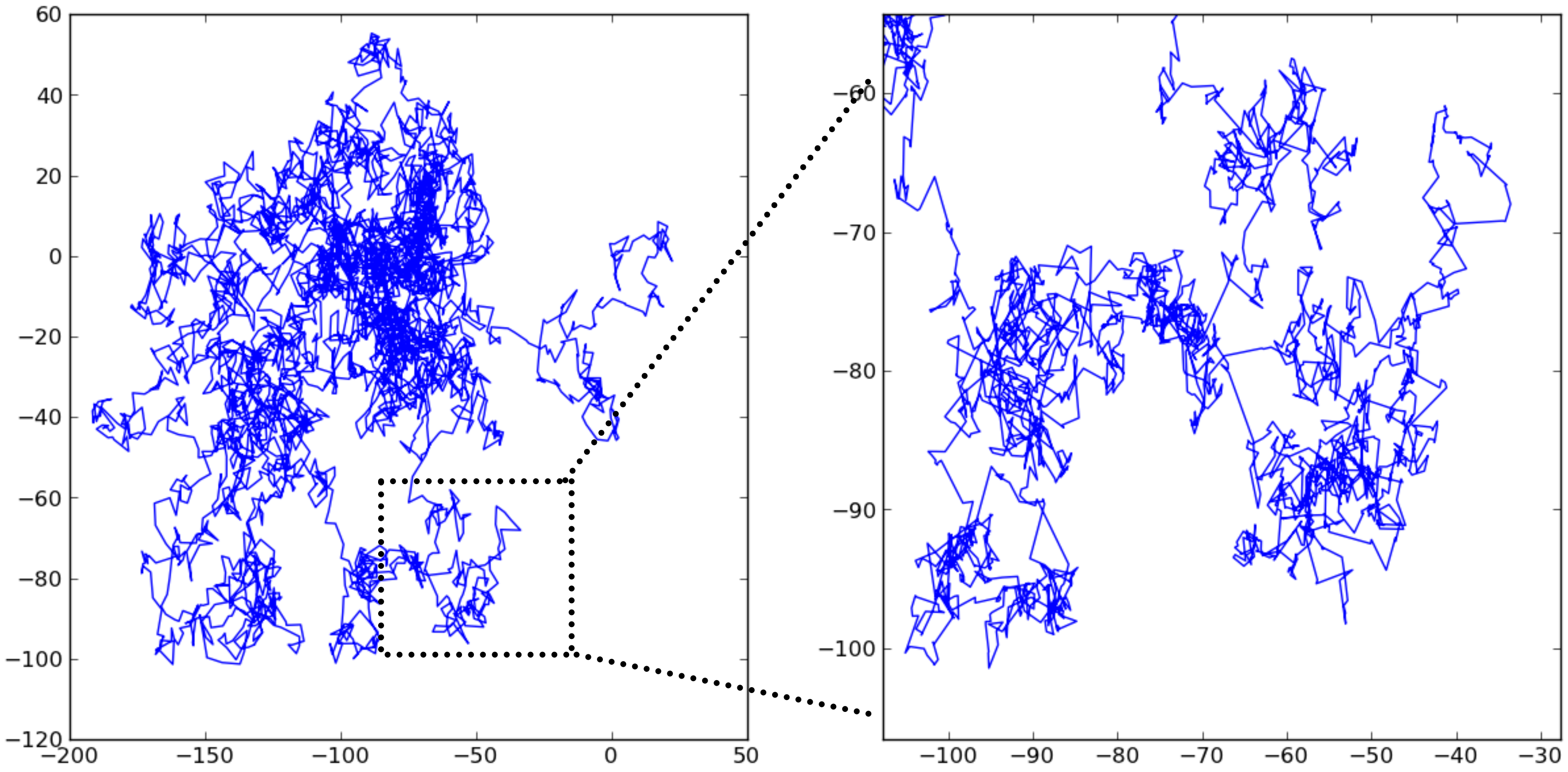
Random Fractals

Random Fractals

- Many natural processes have fractal aspects

- For example Random Walks
  - Self similarity / scale invariance in the statistics as much as the trajectories


60
40
20
0
−20
−40
−60
−80
−100
−120
−200
−150
−100
−50
0
50


60
40
20
0
−20
−40
−60
−80
−100
−120
−200
−150
−100
−50
0
50
−60
−70
−80
−90
−100
−100
−90
−80
−70
−60
−50
−40
−30

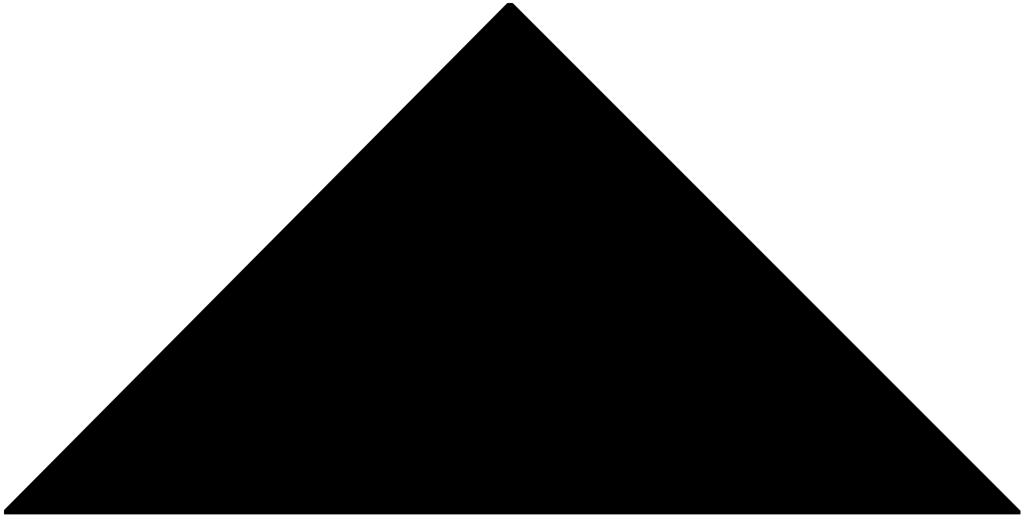
Iterated Function System Fractals

Iterated Function System

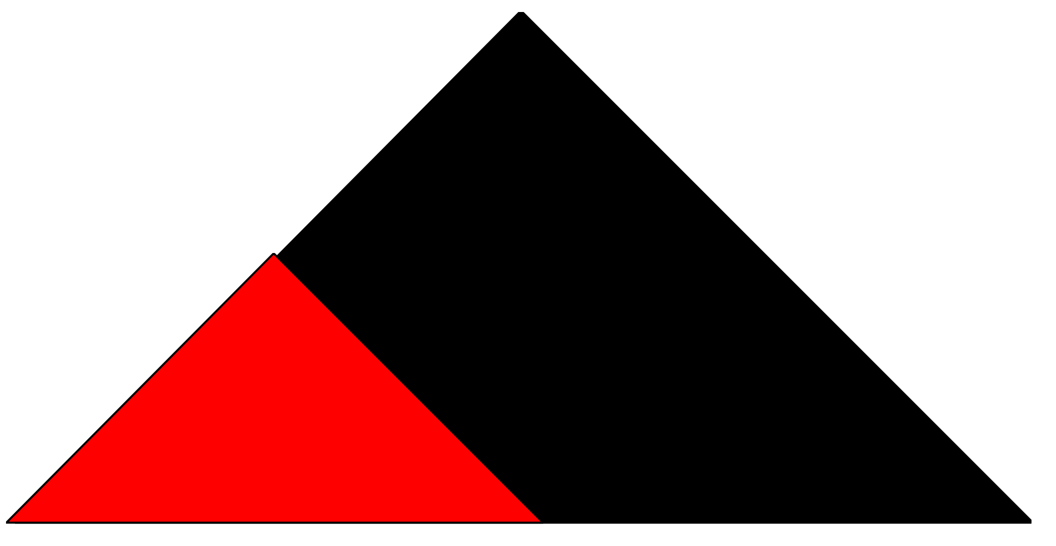
- Take a shape
- Make multiple transformed copies
  - Transform by a function
- Iterate

IFS : Sierpinski Triangle

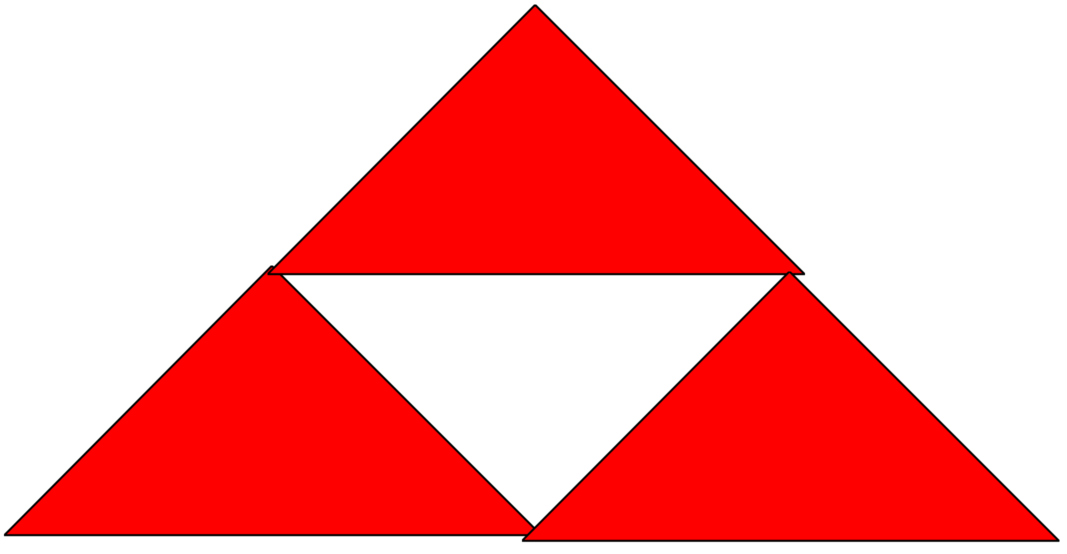
Shrink



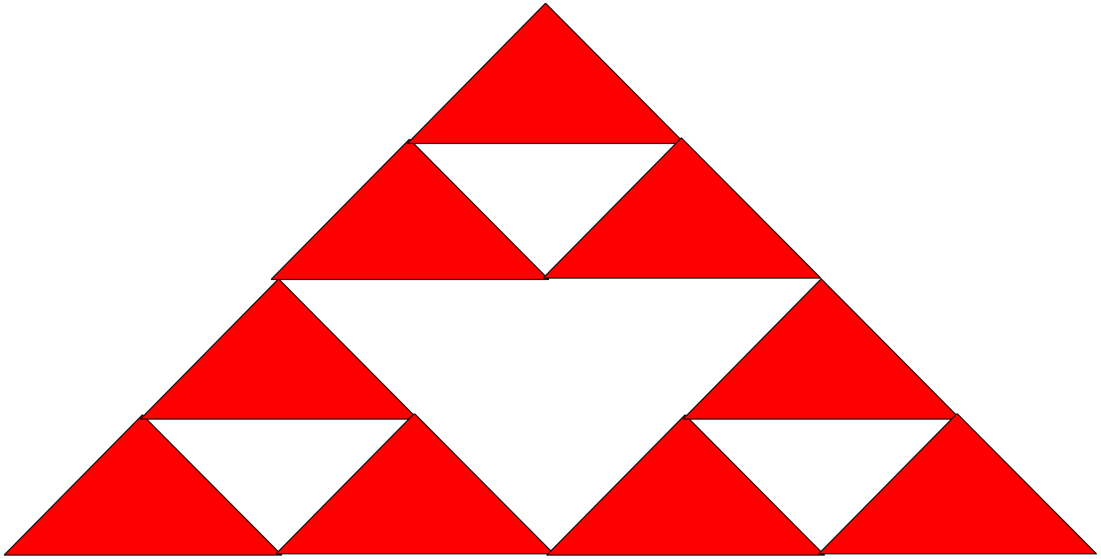
Transform: Shrink



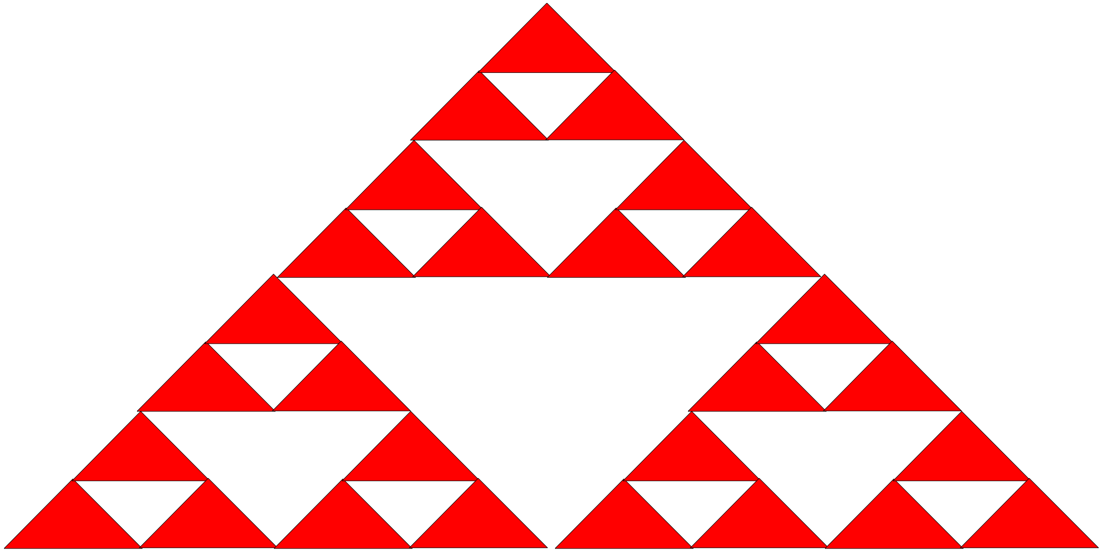
Transform: Triplicate



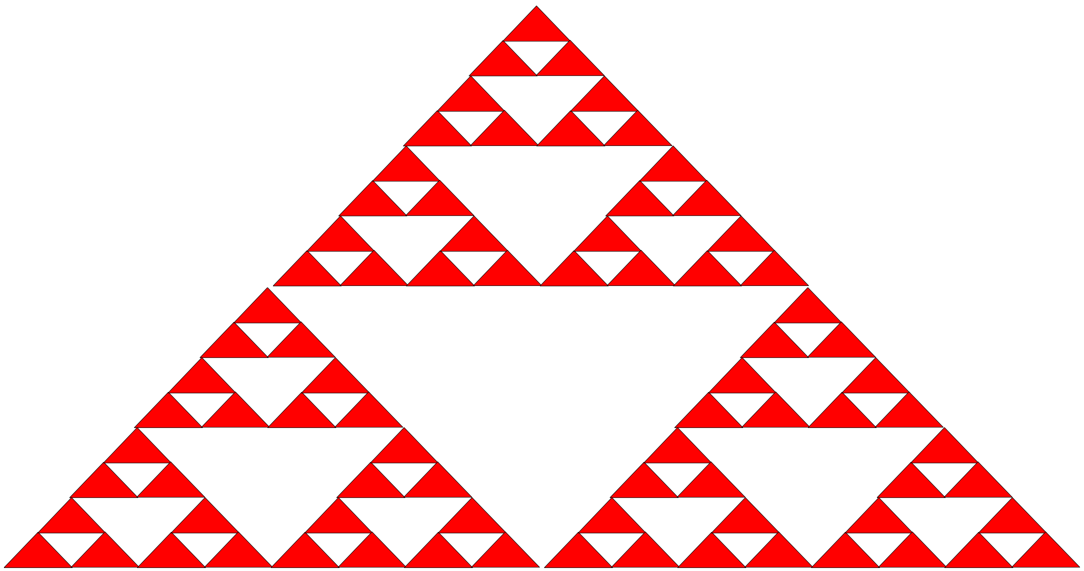
Iterate



Iterate



Iterate



Fractals and Chaos

- Fractals and Chaos are different but connected
- Many ways of visually presenting chaotic systems produce fractals
- For example, the behaviour of many recursive functions is highly sensitive to the initial value
  - Visualising this produces fractals

Recursive Function Fractals

AKA Escape Time Fractals

Recursive Function Fractals

- Given:
  - Initial value Z_0
  - A position (x, y)
  - An function $Z_{n+1} = f(Z_n, x, y)$

- Apply the function recursively many times

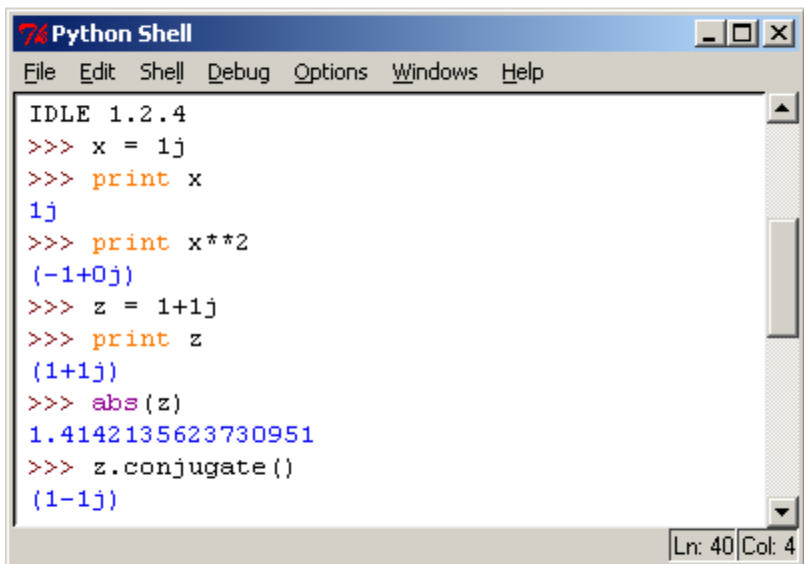
- Does |Z| tend to 0 or infinity?
  - Define a set of all points where $|Z| \rightarrow 0$

Complex Numbers

In Python

Complex Numbers in Python

- Python supports imaginary and complex numbers out of the box
- Append a ‘j’ (Jay, not ‘i’) to a number to make it imaginary
- DOES NOT NEED A NUMPY ARRAY



Example 1

- $Z_{n+1} = Z_n^2 + |(x,y)|^2$

File Edit Format Run Options Windows Help

```
from __future__ import division
import numpy
import matplotlib.pyplot as pyplot
import matplotlib.cm

def f(x, y):
    z0 = 0
    c = (x**2 + y**2)
    z = z0
    MAX_ITER = 10
    for i in range(MAX_ITER):
        z = z**2 + c
        if abs(z) > 1:
            break

    return i == MAX_ITER-1
```

Ln: 15 Col: 8

Example 1

- $Z_{n+1} = Z_n^2 + |(x,y)|^2$

```
File  Edit  Format  Run  Options  Windows  Help
from __future__ import division
import numpy
import matplotlib.pyplot as pyplot
import matplotlib.cm

def f(x, y):
    z0 = 0
    c = (x**2 + y**2)
    z = z0
    MAX_ITER = 10
    for i in range(MAX_ITER):
        z = z**2 + c
        if abs(z) > 1:
            break

    return i == MAX_ITER-1
Ln: 15 Col: 8
```

To speed things up we exit the ‘for’ loop early if we detect that our value is tending to infinity

We exit the loop early with the **`break`** statement.

Example 1

- $Z_{n+1} = Z_n^2 + |(x,y)|^2$

```python
from __future__ import division
import numpy
import matplotlib.pyplot as pyplot
import matplotlib.cm

def f(x, y):
    z0 = 0
    c = (x**2 + y**2)
    z = z0
    MAX_ITER = 10
    for i in range(MAX_ITER):
        z = z**2 + c
        if abs(z) > 1:
            break

    return i == MAX_ITER-1
```

If this condition is true it means that the ‘for’ loop is exhausted – it didn’t break early.

We assume that this means the value is not tending to infinity but to zero

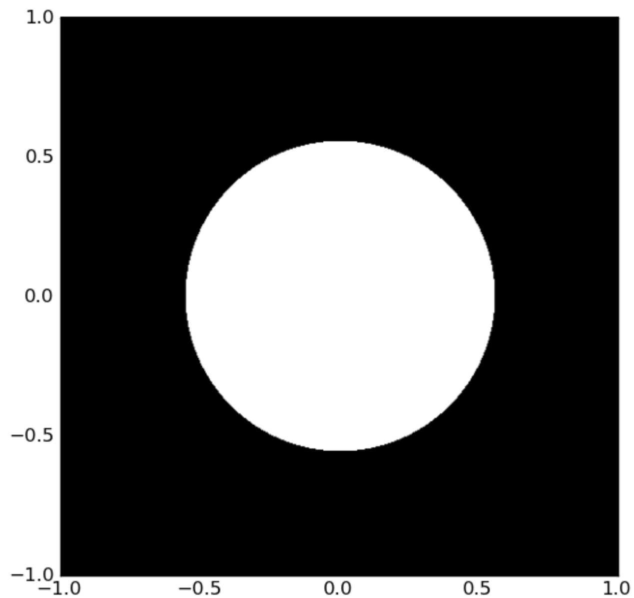
Example 1

- $Z_{n+1} = Z_n^2 + |(x,y)|^2$

File Edit Format Run Options Windows Help

```
from __future__ import division
import numpy
import matplotlib.pyplot as pyplot
import matplotlib.cm

def f(x, y):
    z0 = 0
    c = (x**2 + y**2)
    z = z0
    MAX_ITER = 10
    for i in range(MAX_ITER):
        z = z**2 + c
        if abs(z) > 1:
            break

    return i == MAX_ITER-1
```

Ln: 15 Col: 8

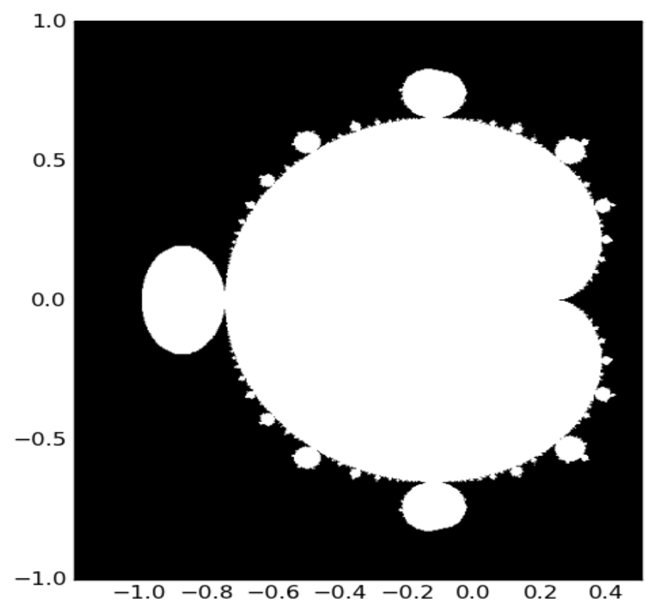


Example 2

- $Z_{n+1} = Z_n{}^2 + x + iy$

Complex plane

```
def f(x, y):
    z0 = 0
    c = x + y * 1j
    z = z0
    MAX_ITER = 200
    for i in range(MAX_ITER):
        z = z**2 + c
        if abs(z) > 1:
            break

    return i == MAX_ITER-1
```



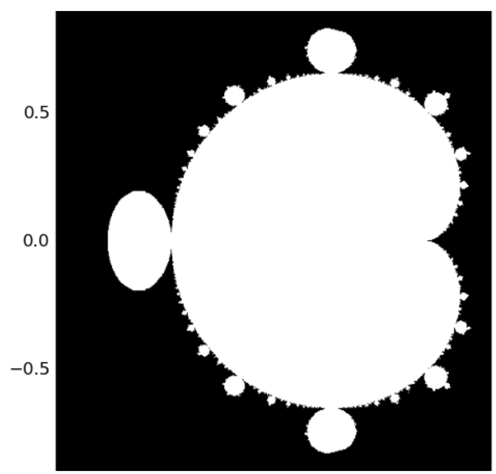
Mandelbrot Set

- A set (collection)
  - of all points (x, y)
  - Where the formula $Z_{n+1} = Z_n^2 + (x+iy)$ remains *bounded* – i.e. magnitude does not tend to infinity
- The Mandelbrot set results from the recursion being chaotic
  - A tiny change in initial position has a large effect (inside/outside the set)
- The Mandelbrot set displays
  - an incredible level of endless detail when zoomed
  - Self similarity at all levels

Making it pretty

escape1.py - G:\teaching\2010-2011\CompPhy...

```python
def f(x, y):
    z0 = 0
    c = x + y * 1j
    z = z0
    MAX_ITER = 200
    for i in range(MAX_ITER):
        z = z**2 + c
        if abs(z) > 1:
            break

    return i == MAX_ITER-1
```



escape1.py - G:/teaching/2010-2011/CompPh...

```python
def f(x, y):
    z0 = 0
    c = x + y * 1j
    z = z0
    MAX_ITER = 200
    for i in range(MAX_ITER):
        z = z**2 + c
        if abs(z) > 1:
            break

    return i
```

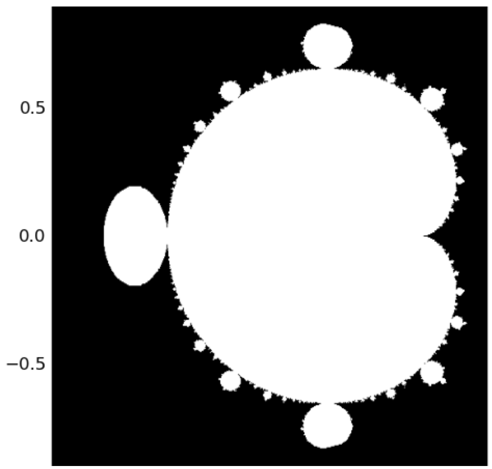
Making it pretty

escape1.py - G:\teaching\2010-2011\CompPhy...

File Edit Format Run Options Windows Help

```
def f(x, y):
    z0 = 0
    c = x + y * 1j
    z = z0
    MAX_ITER = 200
    for i in range(MAX_ITER):
        z = z**2 + c
        if abs(z) > 1:
            break

    return i == MAX_ITER-1
```

Ln: 1 Col: 0



Colour points based on:

- Inside the set (white)
- Outside the set (black)

Informative but dull

Making it pretty

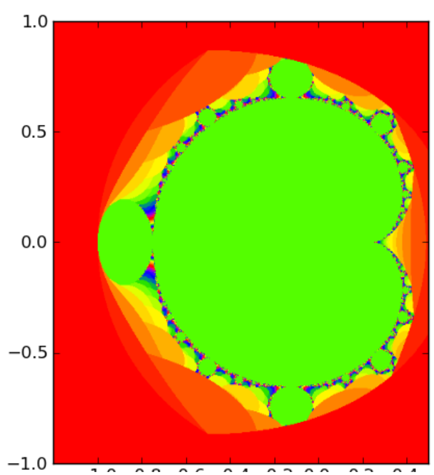
- Colour points based on how quickly the recursion ‘escapes’ from the set – i.e. how fast it is tending to infinity
- Not as interesting mathematically
- But it was the key step to getting fractals into the wider public eye!

escape1.py - G:/teaching/2010-2011/CompPh...

File Edit Format Run Options Windows Help

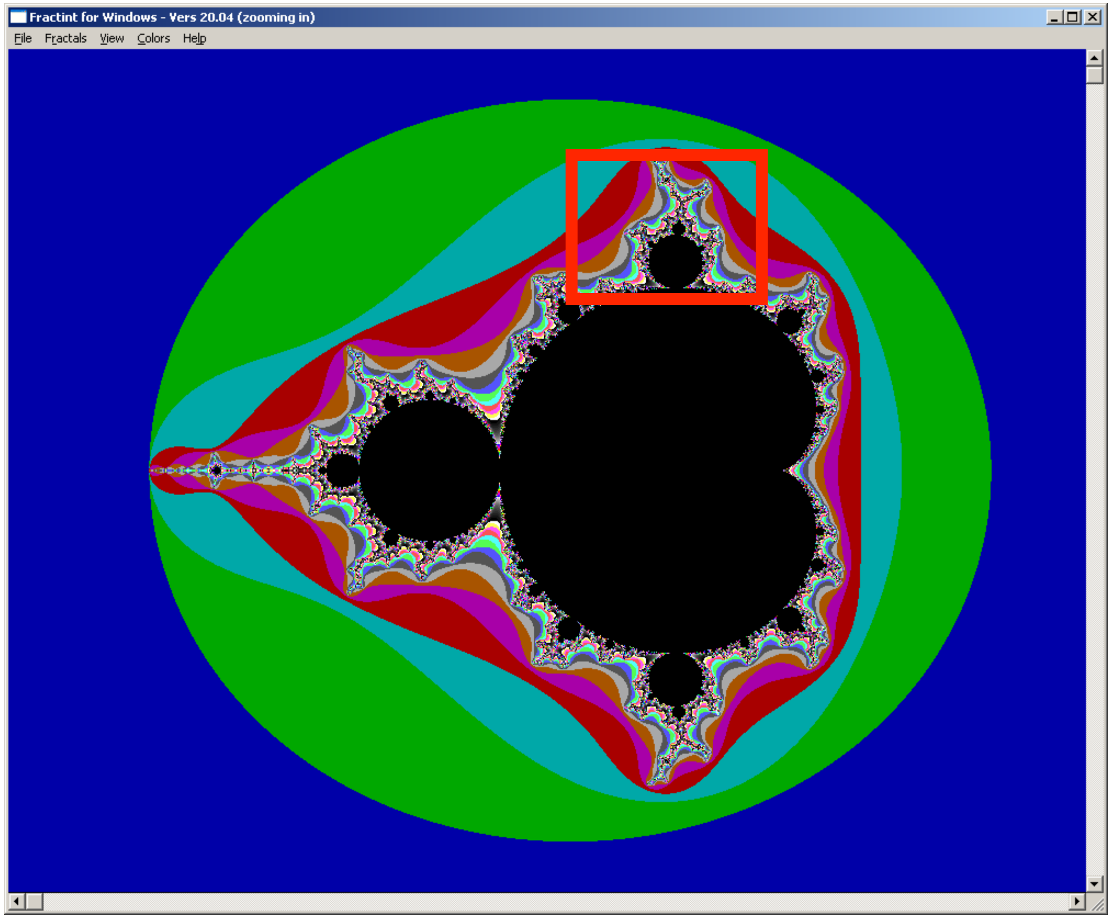
```python
def f(x, y):
    z0 = 0
    c = x + y * 1j
    z = z0
    MAX_ITER = 200
    for i in range(MAX_ITER):
        z = z**2 + c
        if abs(z) > 1:
            break
    return i
```

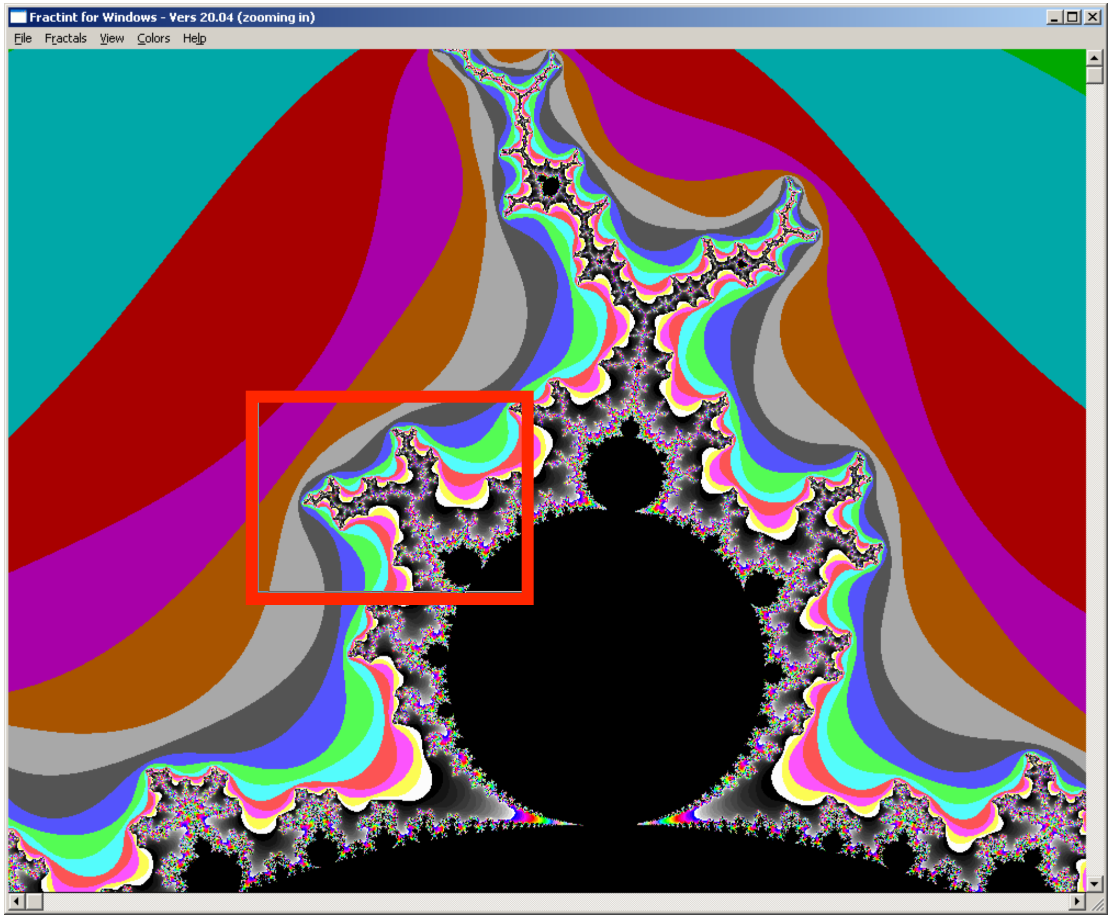
Ln: 16 Col: 13

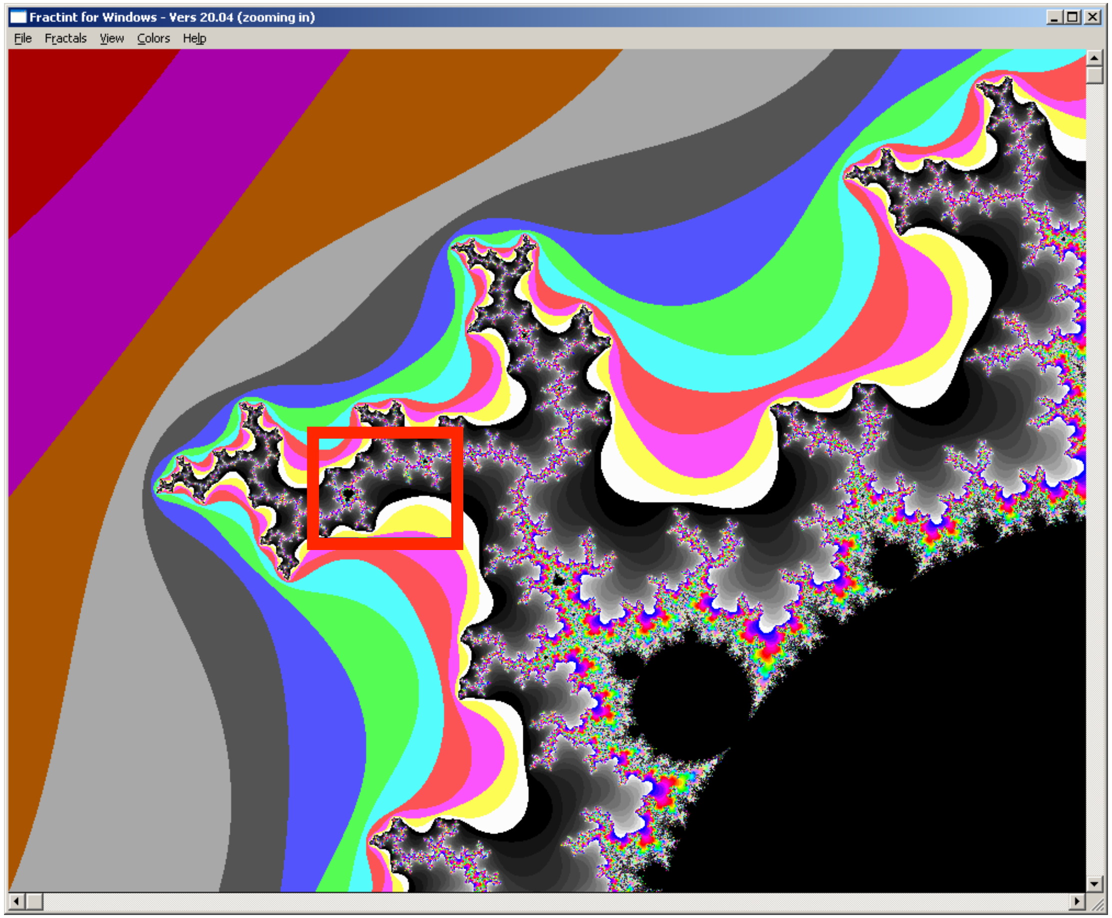


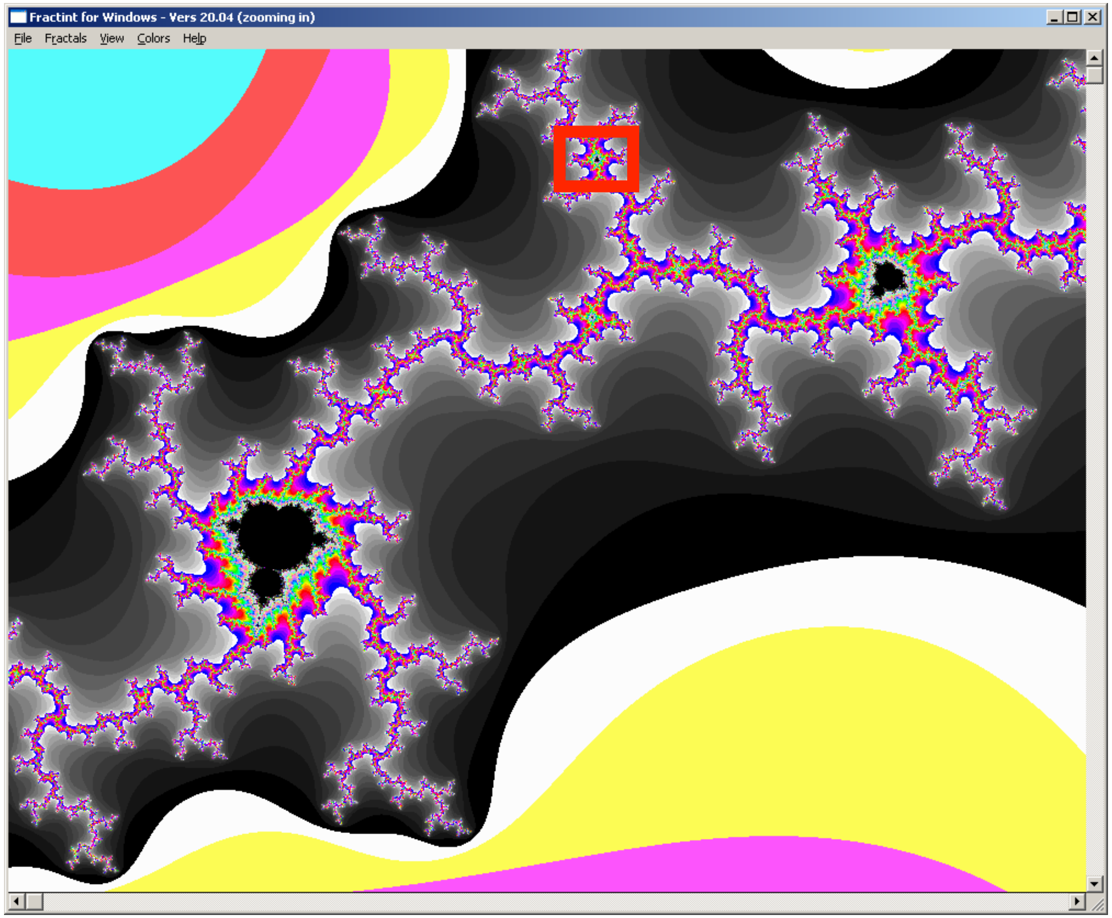
Exploring the Mandelbrot Set

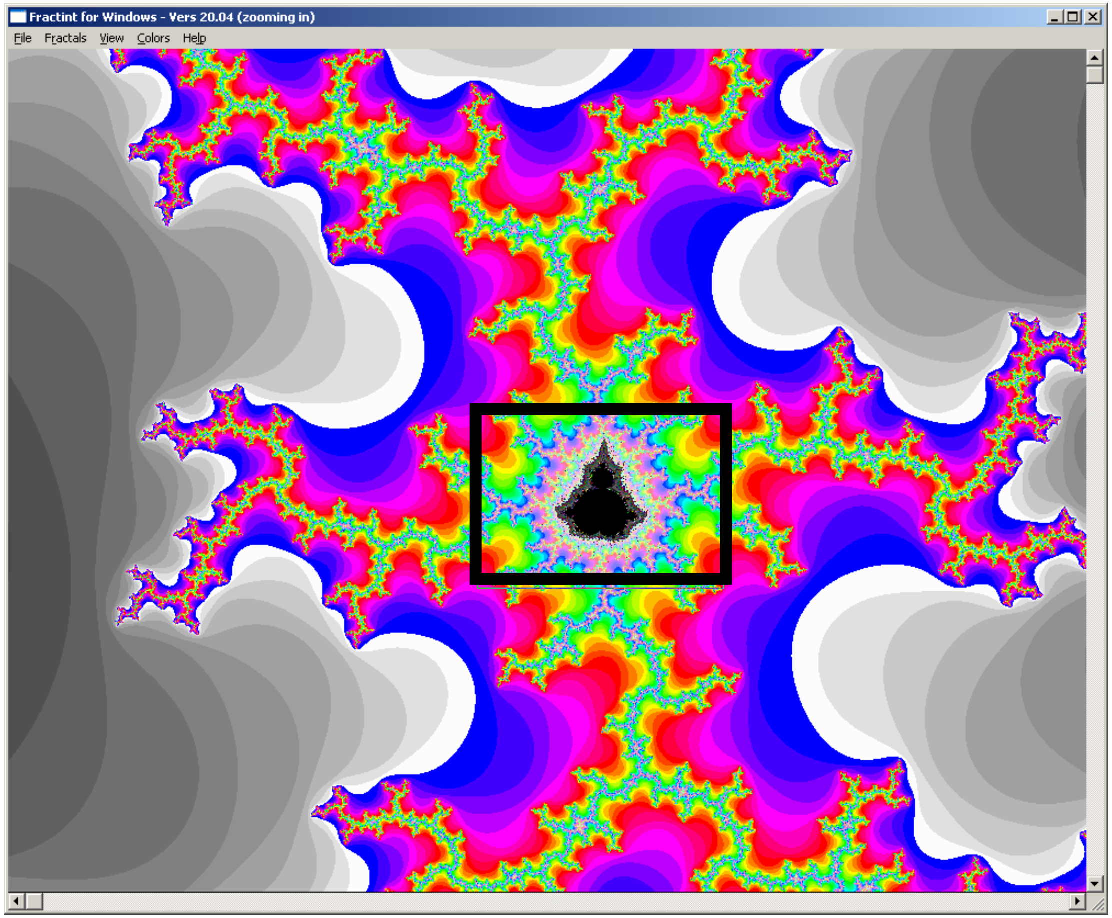
- Fractint / Winfract
- I was using this 20+ years ago, and it’s got staying power!
- http://www.fractint.org/

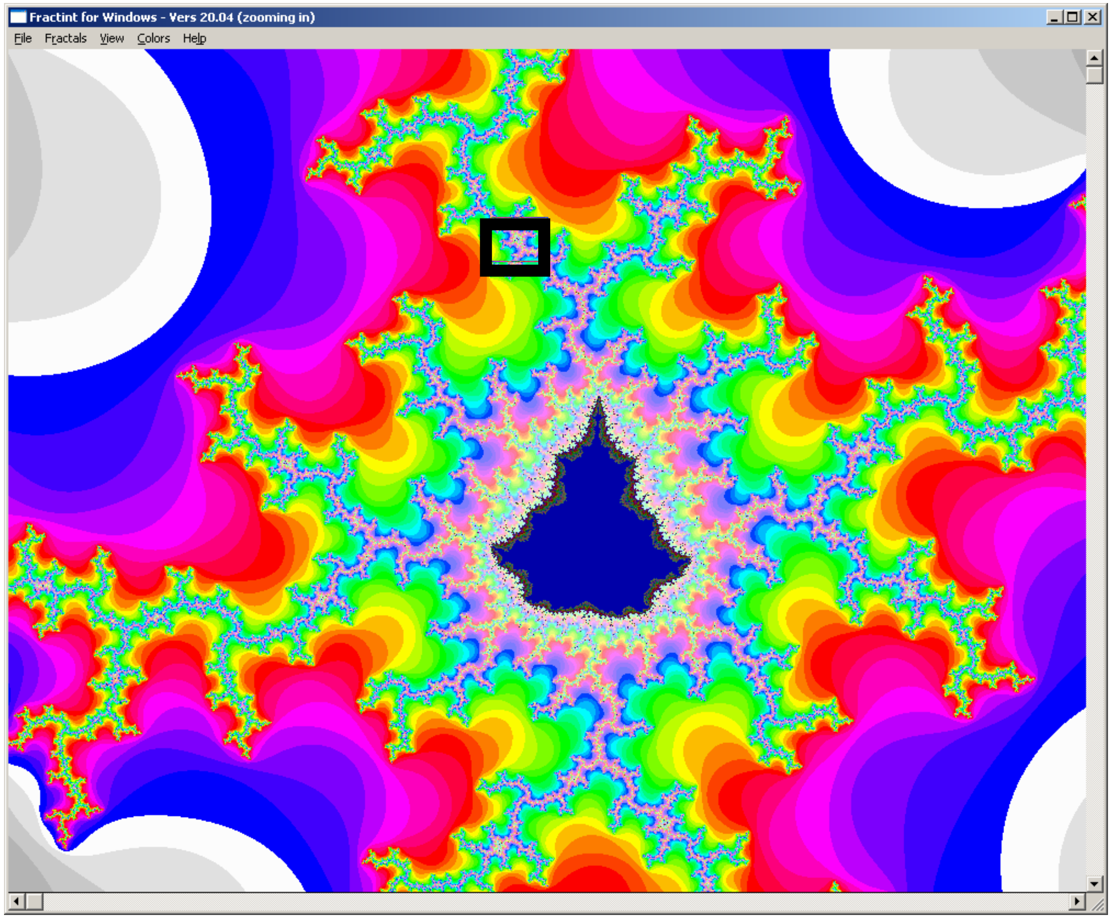

Fractint for Windows - Vers 20.04 (zooming in)
File Fractals View Colors Help

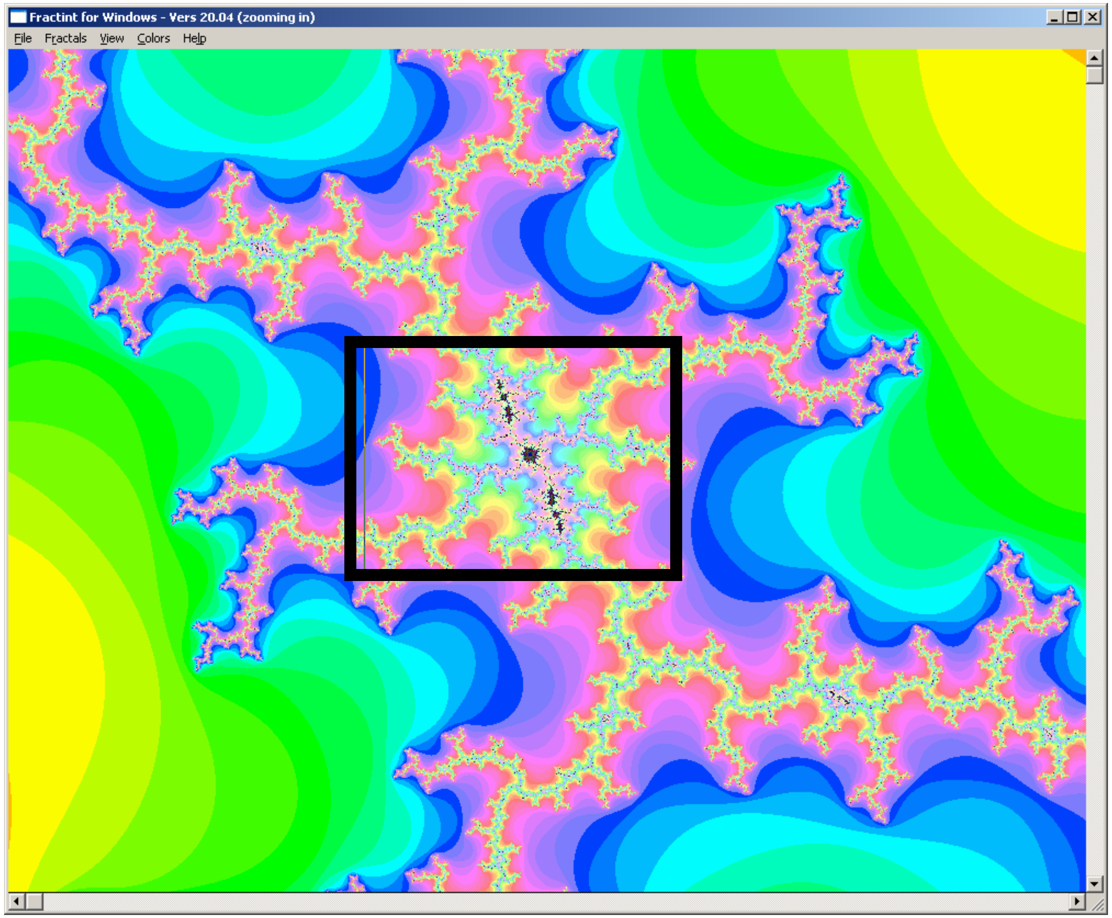

Fractint for Windows - Vers 20.04 (zooming in)
File Fractals View Colors Help

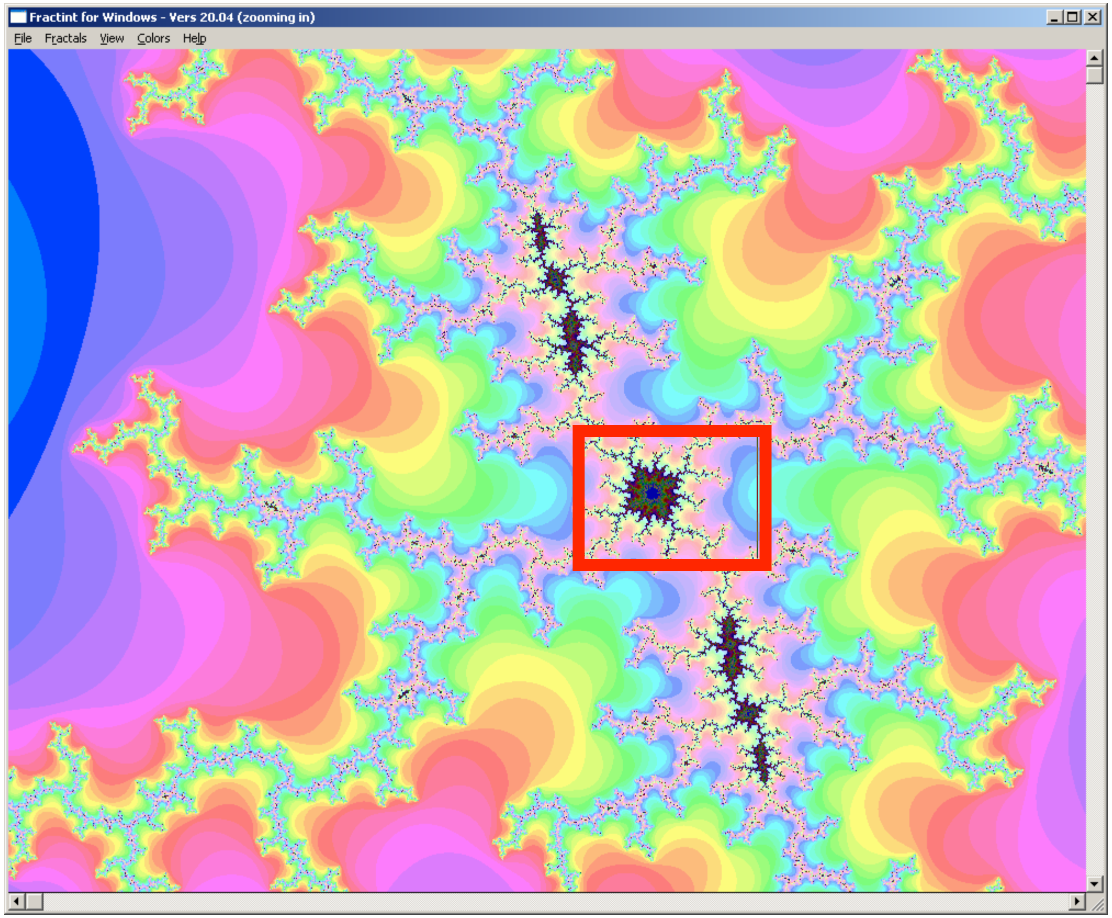

Fractint for Windows - Vers 20.04 (zooming in)
File
Fractals
View
Colors
Help

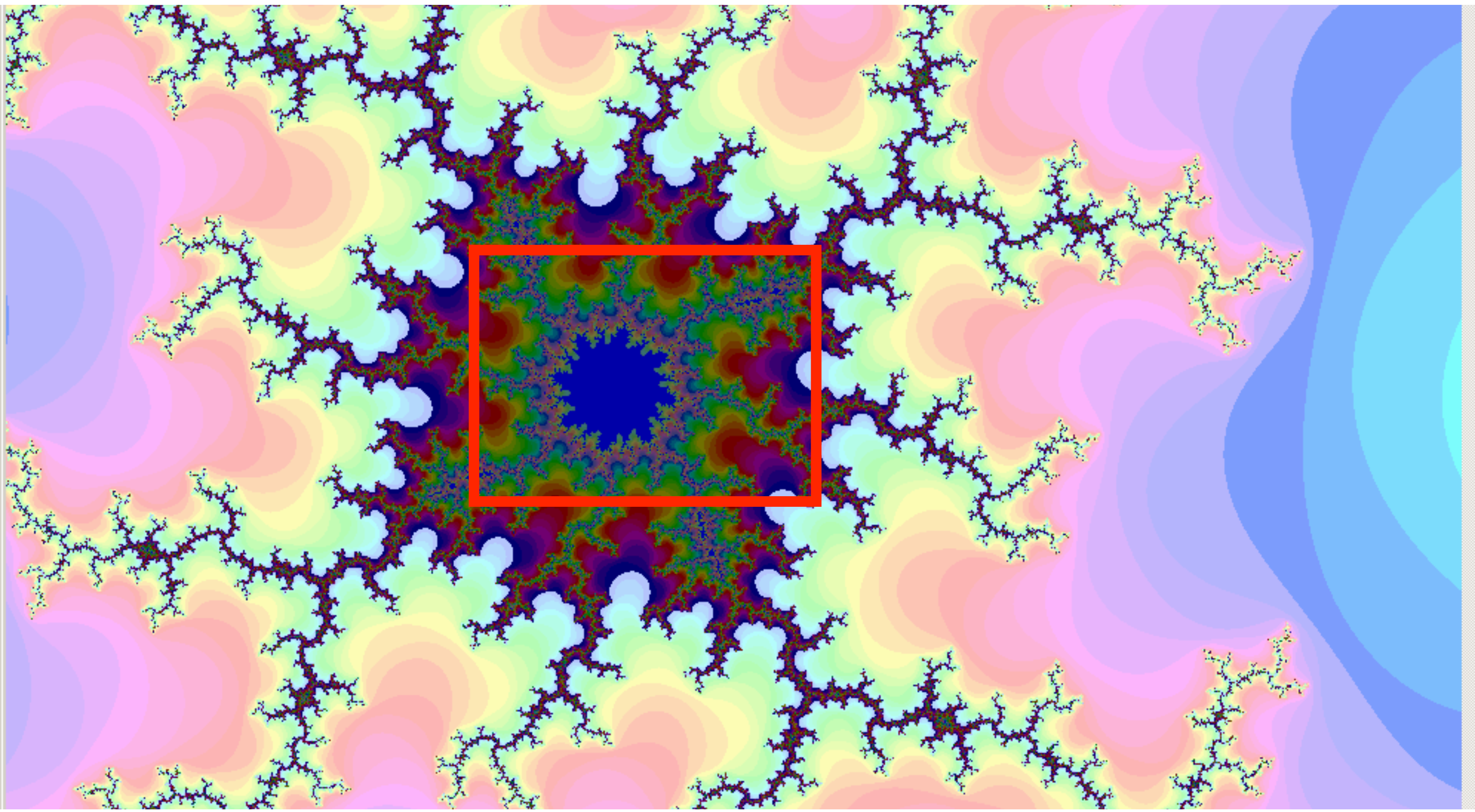

Fractint for Windows - Vers 20.04 (zooming in)
File Fractals View Colors Help

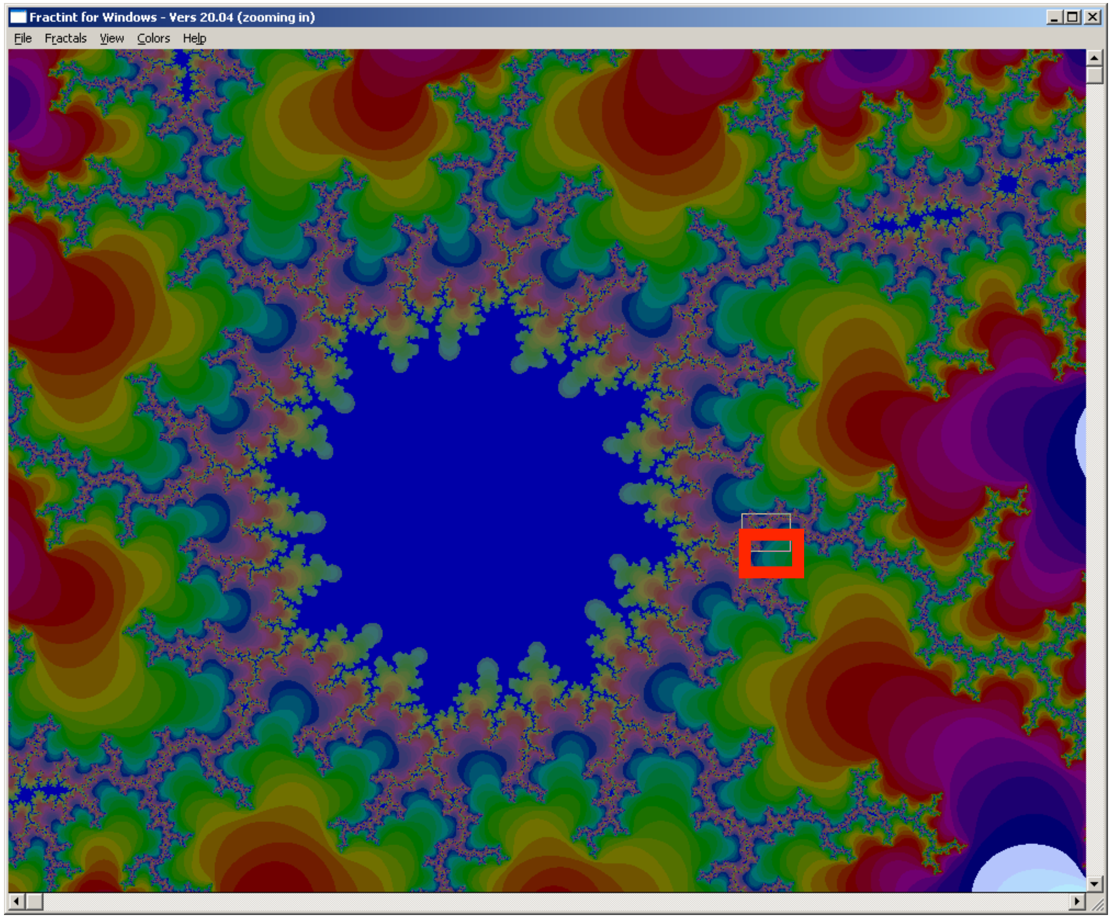

Fractint for Windows - Vers 20.04 (zooming in)
File Fractals View Colors Help

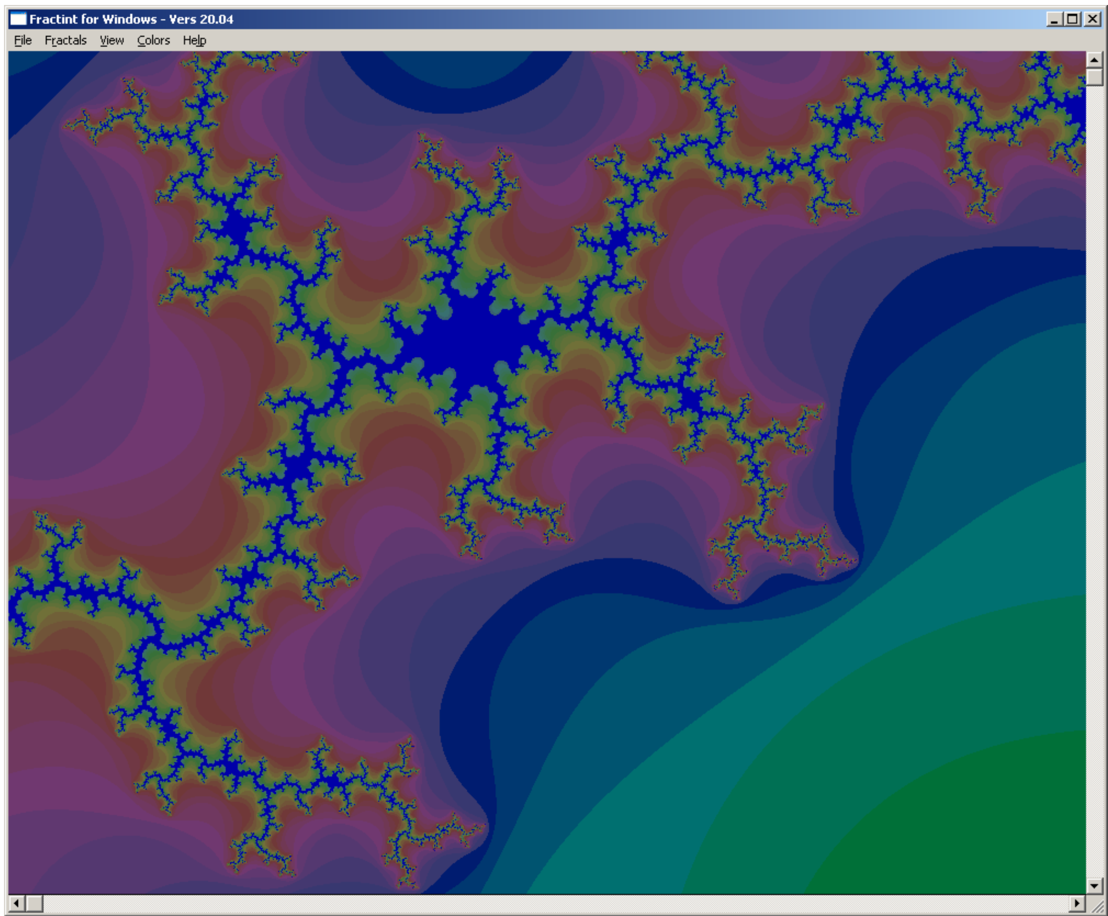

Fractint for Windows - Vers 20.04 (zooming in)
File Fractals View Colors Help

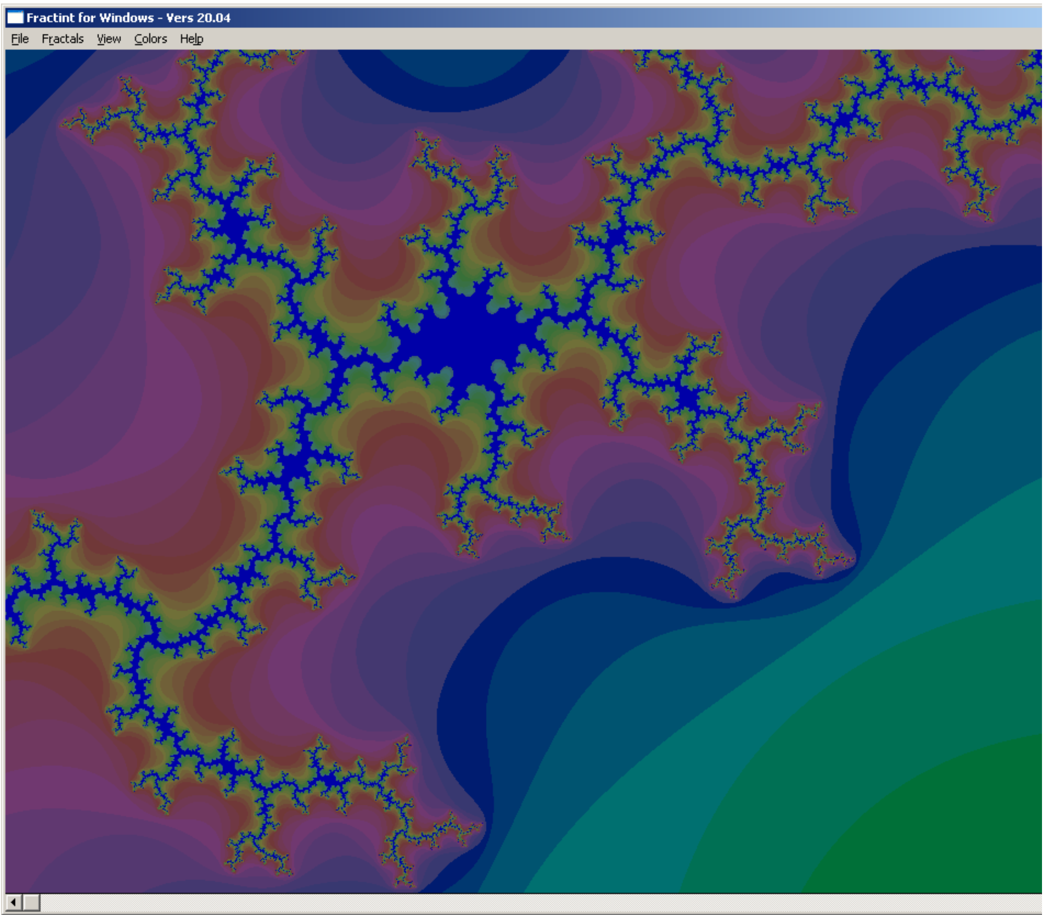

Fractint for Windows - Vers 20.04 (zooming in)
File Fractals View Colors Help


Fractint for Windows - Vers 20.04 (zooming in)
File Fractals View Colors Help




Fractint for Windows - Vers 20.04 (zooming in)
File Fractals View Colors Help


Fractint for Windows - Vers 20.04
File Fractals View Colors Help



We have now zoomed in ~100,000,000x from the first image

That's similar to the ratio between a person and a virus

The complexity in the Mandelbrot set appears endless

Exploring it certainly challenges computers

Precision!

Root Finding

A change of tack?

Root Finding

- Root finding algorithms look for the root(s) of a function

- I.e.:
  - Given $f(x)$
  - Find a value of x where $f(x) = 0$

- Numerical methods
  - Bisection
  - Newton-Raphson
  - Secant
  - Many more

Newton Raphson

- Iterative method
  - Starts from a seed point, X_0
  - (normally) converges on the nearest root
- See whiteboard for use
- Found root can be chaotic in the presence of multiple roots
- You are going to investigate and plot this for your last weekly assessment