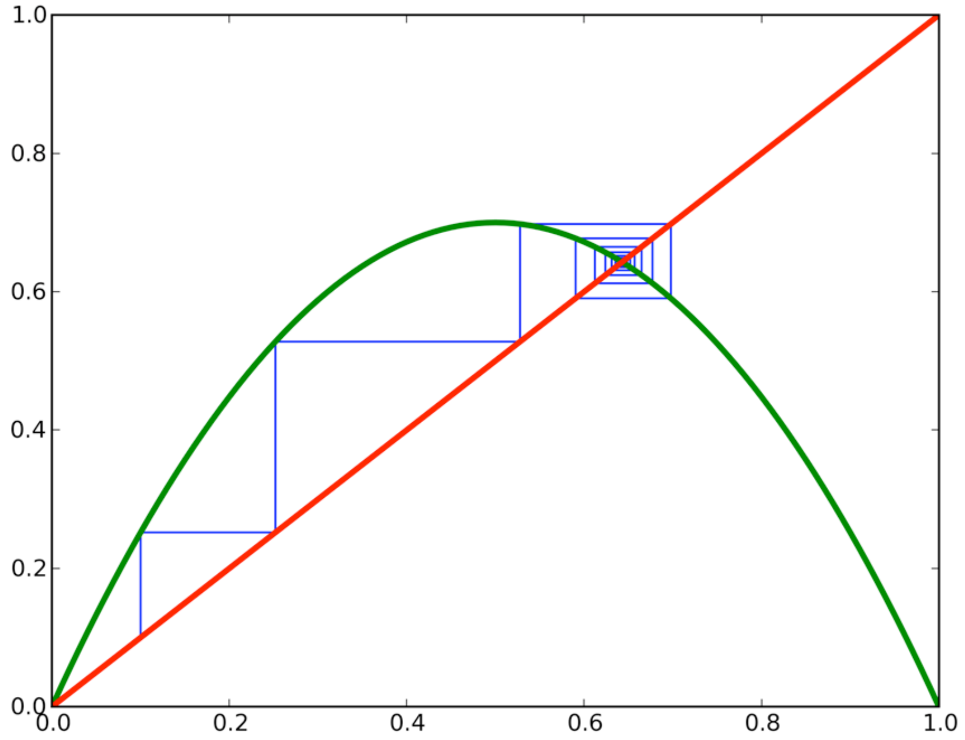
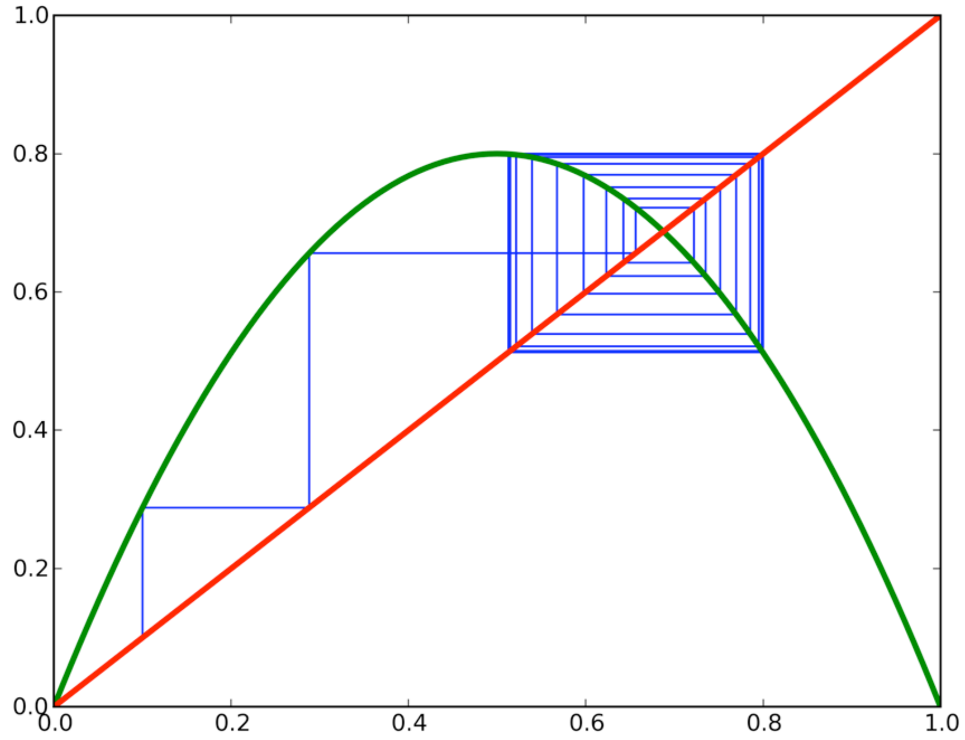
Bifurcations & Chaos in Iterated Maps I: Chaos & Lyapunov Exponents / Invariant Measure

Phys 7882 / CIS 6229: Computational Methods for Nonlinear Systems

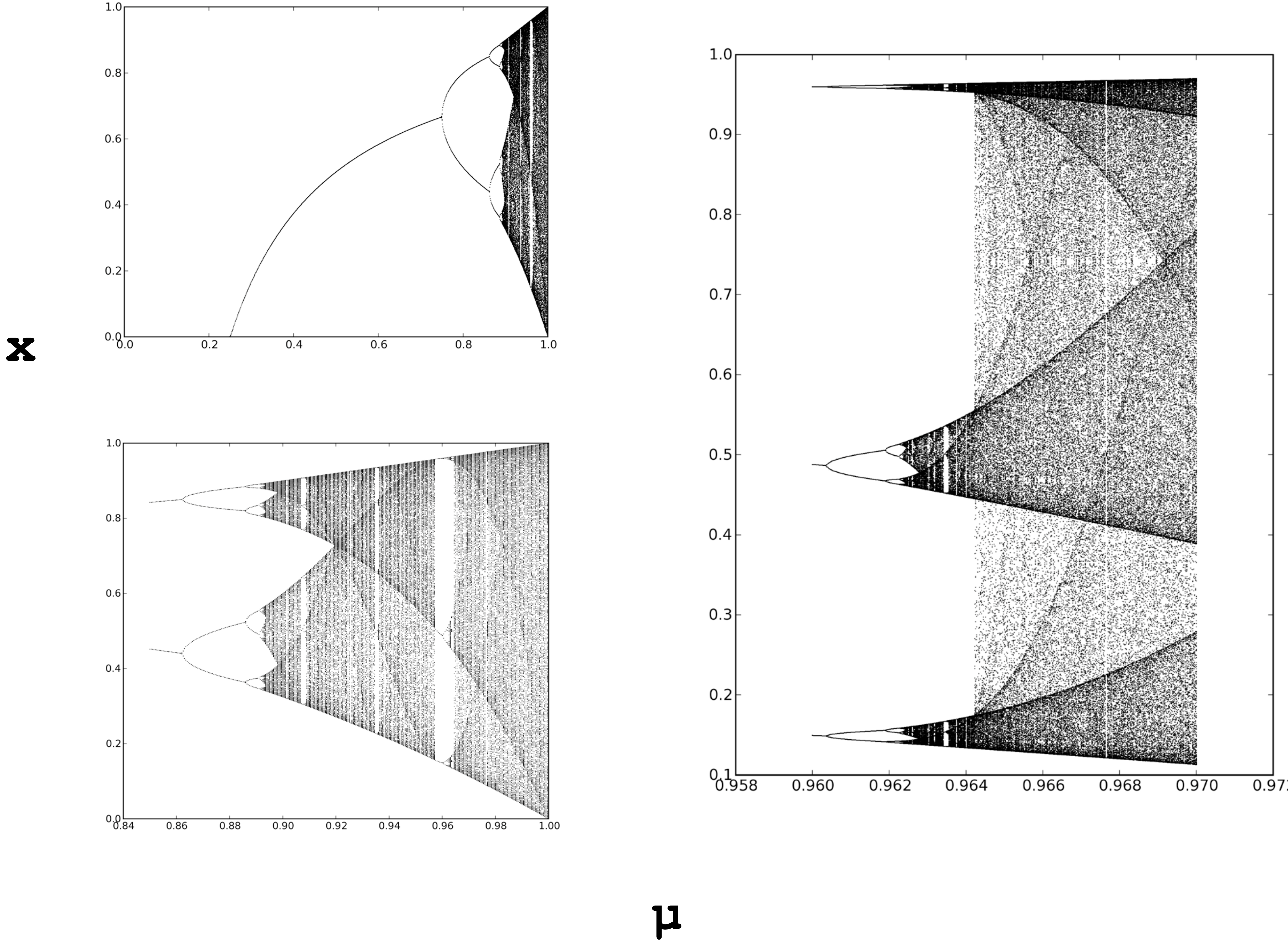




Logistic map:

$$x_{n+1} = 4\mu x_n(1-x_n)$$

Bifurcation Diagram

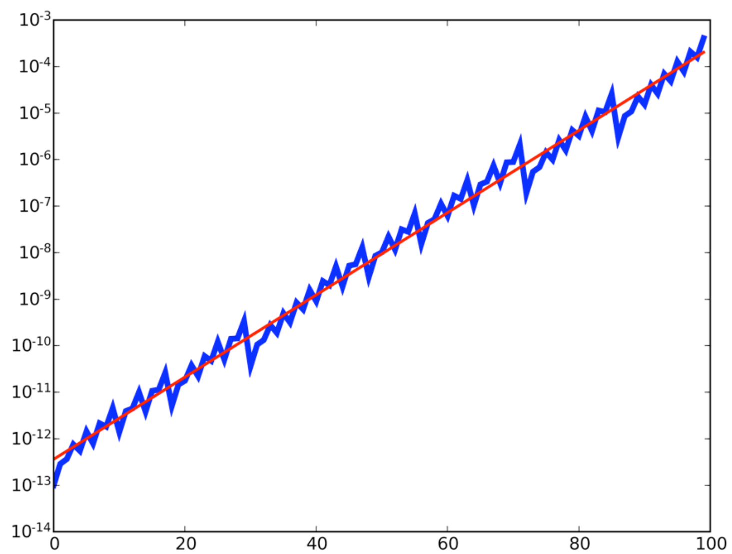


Lyapunov Exponent

rate of divergence (or convergence) of nearby trajectories

$\Delta x_{n+1} \sim \exp(\lambda t)$ → sensitive dependence on initial conditions

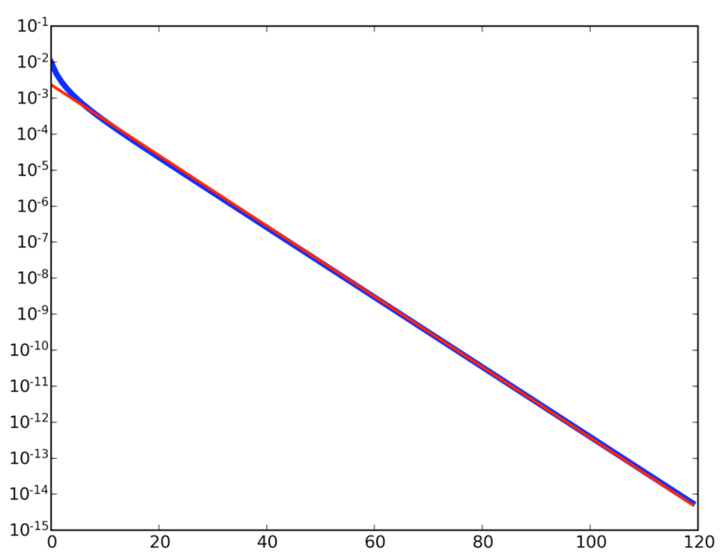
Δx



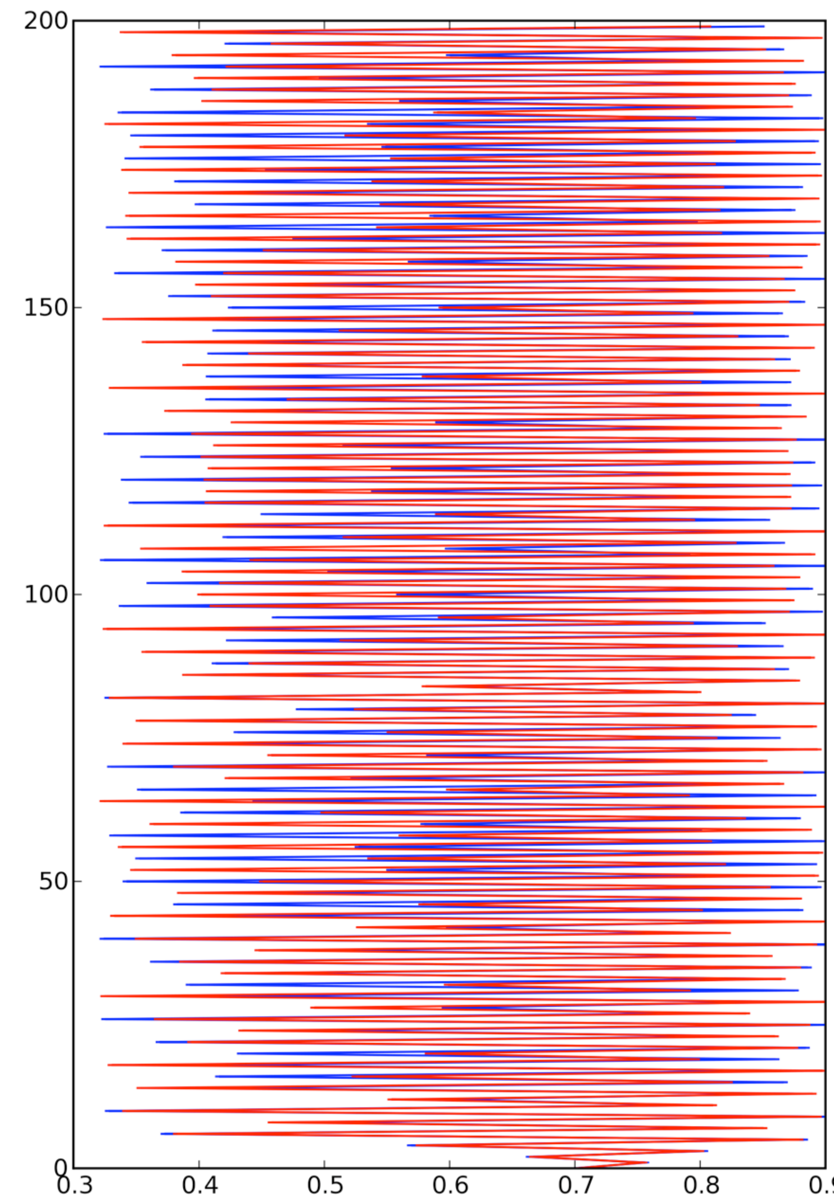
μ=0.9

t

Δx

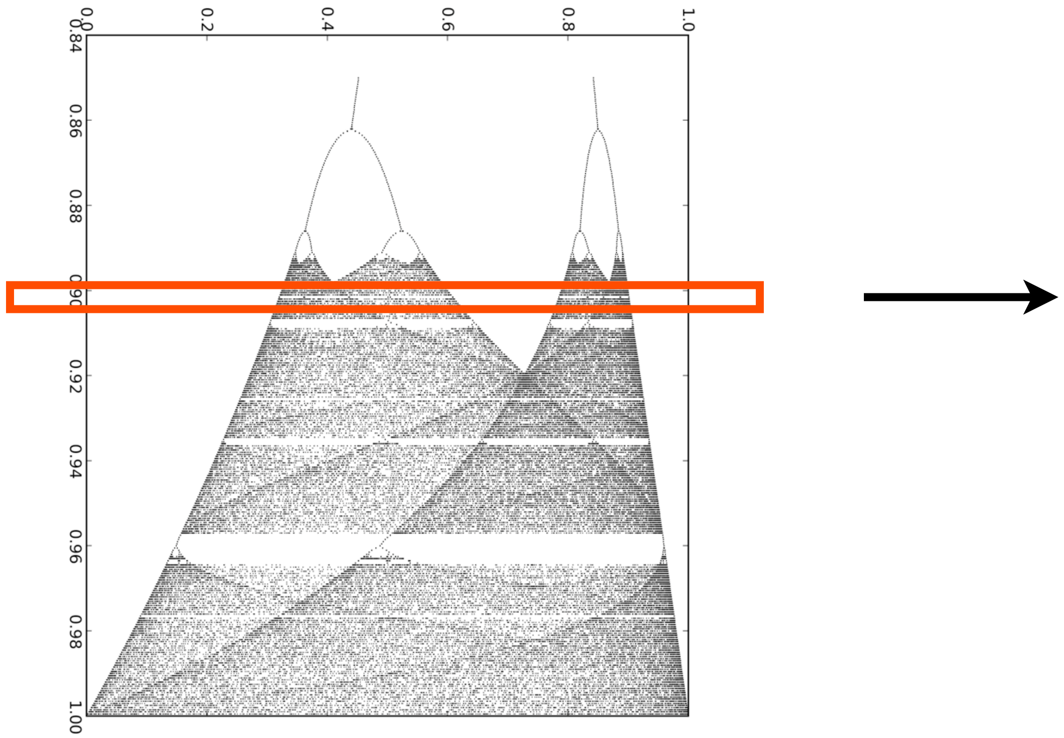


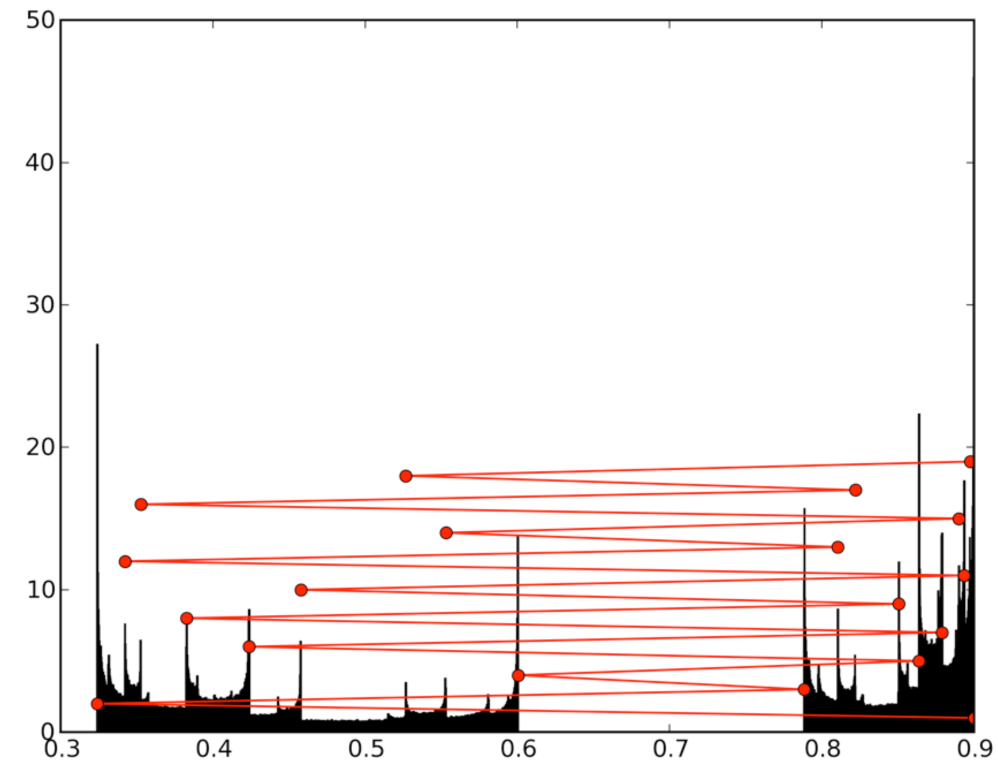
μ=0.3

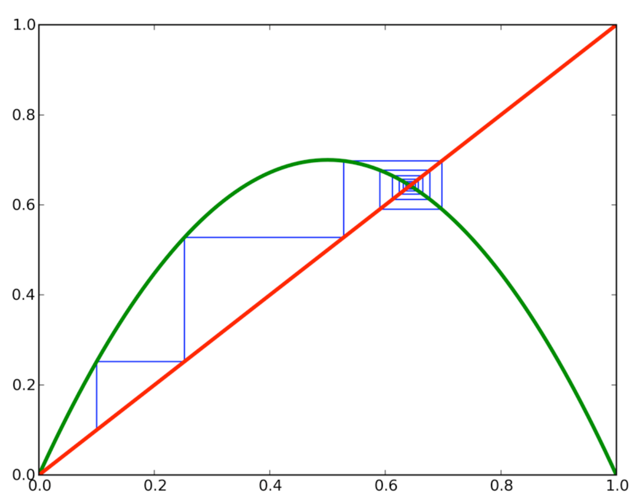


Invariant Measure

stationary probability density in chaotic regime







`Δy = (local slope) * Δx`

- trajectories get expanded or compressed depending on value of local slope
- at the critical point of the map (at x=0.5), slope → 0 ⇒ compression → ∞

- singularities in invariant measure

Persistence in Python

- Persistence: preserving data
  - also serialization: converting arbitrarily structured data to flat (serial) datastream (e.g., file)
  - in Python, "pickling"

```
import pickle  # or, alternatively, import cPickle

output = open('data.pkl', 'wb')

# Pickle data1 using protocol 0.
pickle.dump(data1, output)

# Pickle data2 using the highest protocol available.
pickle.dump(data2, output, -1)
output.close()

pkl_file = open('data.pkl', 'rb')
data1 = pickle.load(pkl_file)
data2 = pickle.load(pkl_file)
pkl_file.close()
```

Timing in Python

```
import time

t1 = time.time()  # number of seconds since the Epoch (1/1/70)
run_some_big_function()
t2 = time.time()  # some slightly bigger integer
diff = t2-t1

print “%s seconds to execute run_some_big_function()” % diff

# or, in ipython, some convenience functions

%time run_some_big_function()

CPU times: user 20.29 s, sys: 1.44 s, total: 21.73 s
Wall time: 22.07 s

%timeit run_some_not_quite_so_big_function()

1000 loops, best of 3: 772 µs per loop
```

Profiling in Python

```
import cProfile, SmallWorldNetworks

cProfile.run('SmallWorldNetworks.FindAverageAveragePathLength(100,4,0.1,1000)')
         22883822 function calls in 63.027 CPU seconds

   Ordered by: standard name

   ncalls  tottime  percall  cumtime  percall filename:lineno(function)
        1    0.000    0.000   63.027   63.027 <string>:1(<module>)
   100000   41.544    0.000   52.533    0.001 Networks.py:134(FindPathLengthsFromNode)
     1000    6.602    0.007   59.691    0.060 Networks.py:167(FindAveragePathLength)
     1000    0.001    0.000    0.001    0.000 Networks.py:29(__init__)
   440000    0.301    0.000    0.301    0.000 Networks.py:39(HasNode)
   440000    0.677    0.000    0.979    0.000 Networks.py:49(AddNode)
   220000    1.249    0.000    2.418    0.000 Networks.py:54(AddEdge)
     2000    0.003    0.000    0.009    0.000 Networks.py:76(GetNodes)
 10000000    6.576    0.000    6.576    0.000 Networks.py:80(GetNeighbors)
     1000    0.536    0.001    2.832    0.003 SmallWorldNetworks.py:26(MakeRingGraph)
     1000    0.080    0.000    0.473    0.000 SmallWorldNetworks.py:39(AddRandomEdges)
     1000    0.007    0.000    3.314    0.003 SmallWorldNetworks.py:48(MakeSmallWorldNetwork)
        1    0.022    0.022   63.027   63.027 SmallWorldNetworks.py:95(FindAverageAveragePathLength)
        1    0.000    0.000    0.000    0.000 fromnumeric.py:440(any)
        2    0.000    0.000    0.000    0.000 numeric.py:126(asarray)
    40000    0.126    0.000    0.160    0.000 random.py:246(choice)
        1    0.000    0.000    0.000    0.000 scimath.py:28(_fix_real_lt_zero)
        1    0.000    0.000    0.000    0.000 scimath.py:46(sqrt)
        1    0.000    0.000    0.000    0.000 type_check.py:58(imag)
        1    0.000    0.000    0.000    0.000 type_check.py:77(isreal)
  1053723    0.427    0.000    0.427    0.000 {len}
        1    0.000    0.000    0.000    0.000 {method 'any' of 'numpy.generic' objects}
 10338087    4.193    0.000    4.193    0.000 {method 'append' of 'list' objects}
        1    0.000    0.000    0.000    0.000 {method 'disable' of '_lsprof.Profiler' objects}
   100000    0.552    0.000    0.552    0.000 {method 'items' of 'dict' objects}
     2000    0.006    0.000    0.006    0.000 {method 'keys' of 'dict' objects}
    40000    0.018    0.000    0.018    0.000 {method 'random' of '_random.Random' objects}
   102001    0.106    0.000    0.106    0.000 {range}
     1000    0.001    0.000    0.001    0.000 {round}
```