Numerical methods in (non-hyperbolic) chaos

Caroline Wormell, Sorbonne Université/CNRS

Outline

1. Motivation, distinguishing chaos, unknowns
2. Monte Carlo estimation and error analysis, statistical tests for chaos
3. Transfer and Koopman operator discretisations I: Galerkin discretisation, EDMD
4. Transfer and Koopman operator discretisations II: convergence rates

Part 1: trajectories and known unknowns

Why numerics in dynamics

Calculations are a natural part of doing mathematics. We need them to:

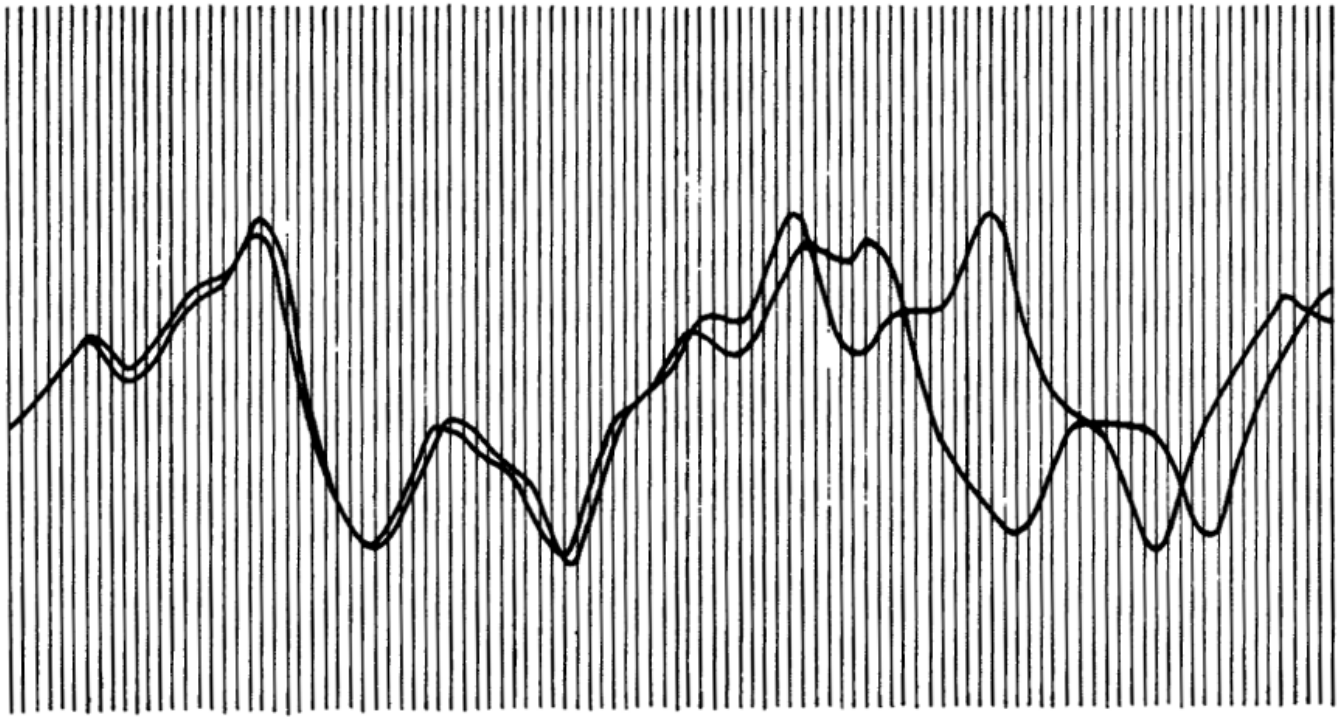
- Estimate quantities/qualitative behaviour in applications (90%*)
- Understand something for mathematical ends: e.g. what results should we try and prove? (9%)
- Prove a theorem (1%)

If they are extensive enough, they are very tedious, so we get a computer to do them for us.

NASA Susan Kare

Symbolic computations (e.g. Mathematica) generally don't help much with chaotic systems, because of their lack of algebraic structure and exponentially growing complexity.

This means we mostly have to *approximate* our systems. This is a lot of grunt work, so computers are crucial.



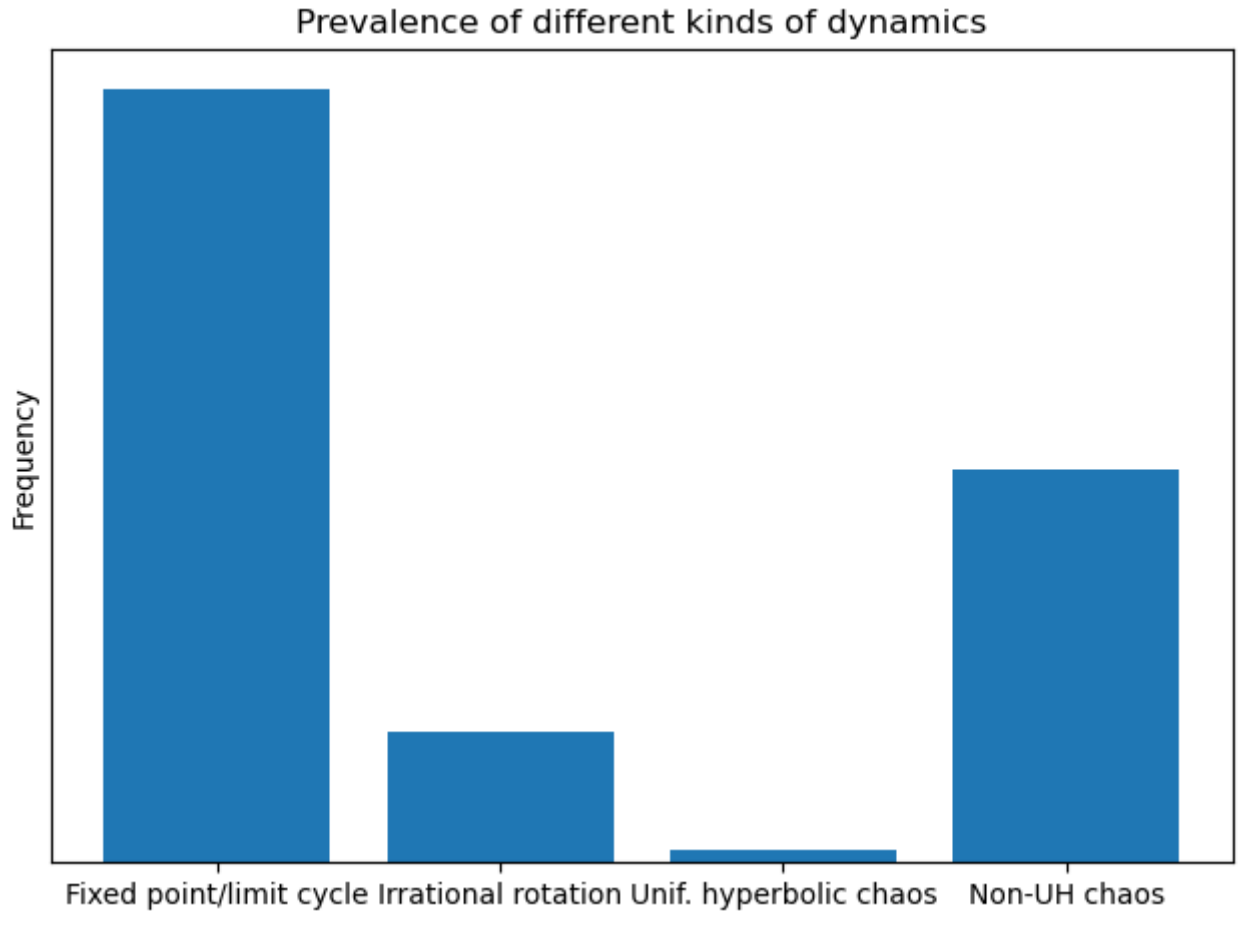
HOW TWO WEATHER PATTERNS DIVERGE. From nearly the same starting point, Edward Lorenz saw his computer weather produce patterns that grew farther and farther apart until all resemblance disappeared. (From Lorenz's 1961 printouts.)

James Gleick, *Chaos*, p. 17

(Non-hyperbolic) chaos

Solid proofs are usually only achievable for systems with nice hyperbolicity properties, which is why we all study them.

But there are good reasons to try to study non-uniformly hyperbolic systems:



with thanks to Ian Melbourne

Trajectories

Sometimes the easiest way to see the dynamical behaviour of the system is to simulate a trajectory (or a lot), and see what it does.

Let's try that for one of the (symbolically!) simplest chaotic systems:

Trajectories

Sometimes the easiest way to see the dynamical behaviour of the system is to simulate a trajectory (or a lot), and see what it does.

Let's try that for one of the (symbolically!) simplest chaotic systems:

In [196]:

```
logistic(x,a) = a*x*(1-x) # logistic map on [0,1] with parameter a ∈ [0,4]
```

Out[196]:

```
logistic (generic function with 1 method)
```

Trajectories

Sometimes the easiest way to see the dynamical behaviour of the system is to simulate a trajectory (or a lot), and see what it does.

Let's try that for one of the (symbolically!) simplest chaotic systems:

In [196]:

```
logistic(x,a) = a*x*(1-x) # logistic map on [0,1] with parameter a ∈ [0,4]
```

Out[196]:

```
logistic (generic function with 1 method)
```

In [197]:

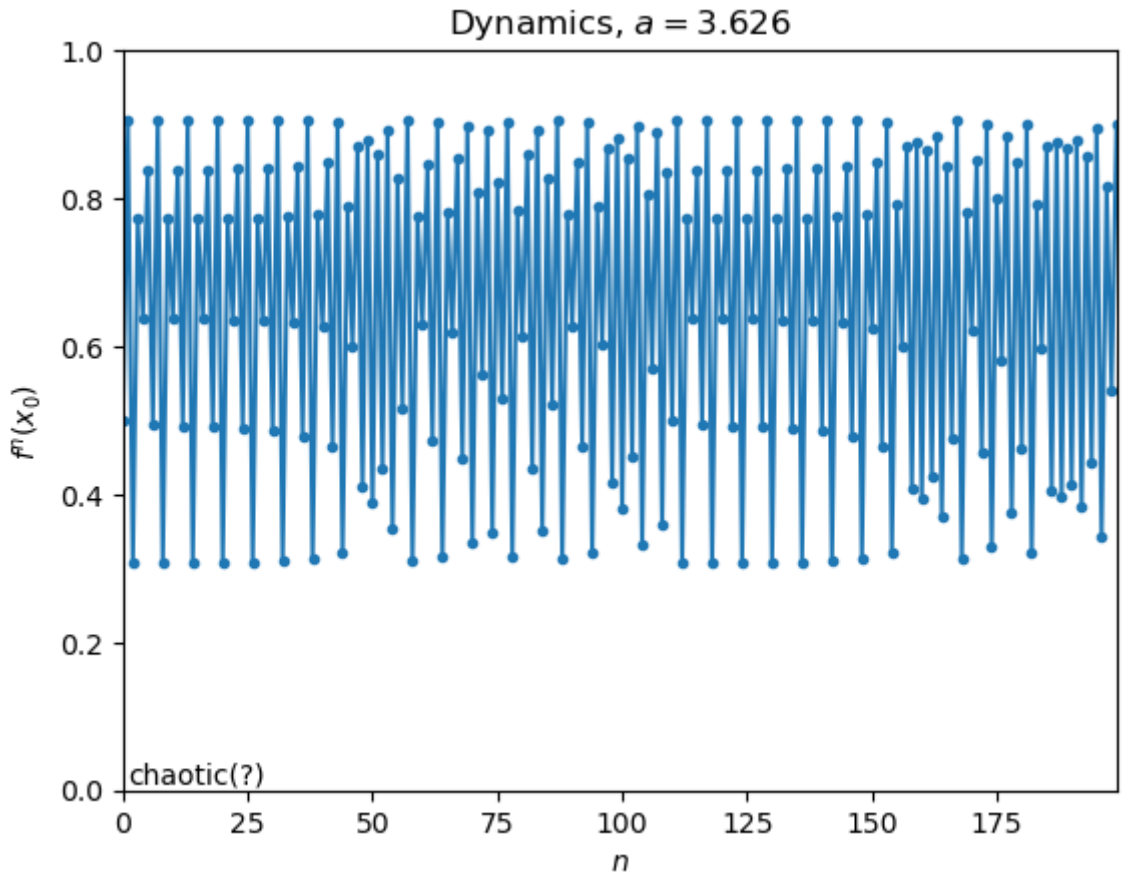
```
function simulate_logistic(a, # logistic parameter
                           N; # time series length
                           x0=rand()) # initial value
    xh = Array{Float64}(undef,N) # initialises a vector to put values in

    xh[1] = x0
    for n = 1:N-1
        xh[n+1] = logistic(xh[n],a)
    end
    xh
end
```

Out[197]:

```
simulate_logistic (generic function with 1 method)
```

In [228]:

```
println(simulate_logistic(3.8,100;x0=sqrt(0.5)))
```




```
[0.7071067811865476, 0.7870057685088805, 0.636985217601
9822, 0.8786931906024428, 0.4050475760870993 5, 0.915739
3413336463, 0.29321104025637046, 0.787505639286960 7, 0.
```

63589392803714 5, 0.879824793226013 9, 0.401785880513099 94, 0.913345149586969, 0.30075399178911 96, 0.799143907205 7132, 0.609949106570458 8, 0.90406253706456, 0.3295871 7133561416, 0.839645977741090 1, 0.511634317256336, 0.949 485642115519 9, 0.182258098621595 3, 0.566352319612057 3, 0.933270004791977 6, 0.236652991200726 75, 0.6864637412346 069, 0.817878838178215 6, 0.566021568106177 7, 0.9334363 796702451, 0.236105038157734 45, 0.685365906634223, 0.81 943003026499155 5, 0.562264941048845 4, 0.935267693041498 4, 0.230059734498461 4, 0.673102561631643 6, 0.8361349119949 394, 0.520650619564986 6, 0.948379497264012 3, 0.1860321 4 04464321, 0.575411896035934, 0.928389574557808 7, 0.2526 3301515868275, 0.717476384279959, 0.770275284665959 3, 0.672414827895191 7, 0.837037883063111 4, 0.518340768445 2783, 0.948721741608779 3, 0.184865434708818 78, 0.5726227 818832053, 0.929958539895868 7, 0.247515485088328 05, 0.7 07755596497330 72, 0.785982344268437 9, 0.639213575313568 8, 0.876354405703970 6, 0.411757972967402 13, 0.920410709 7276774, 0.27836837355582 83, 0.763341804206902 3, 0.6864 741577968033, 0.817864076201097, 0.566057230431128 1, 0. 933418480769521 5, 0.236163998004271 68, 0.685484143393 4581, 0.819263403688461 3, 0.562669340447982 7, 0.935075704 3176964, 0.230694699747349 63, 0.674403689972153 4, 0.834 4167411115686, 0.525028684404408 8, 0.947619546836541, 0.1886196169014286 4, 0.581560576681270 4, 0.924721914859 3907, 0.26452292114505 54, 0.739292072268980 2, 0.7324093 557670253, 0.744746387137433 9, 0.722376982736058 7, 0.76

20842146869853, 0.6889850847652436, 0.81428162339784, 0.5746628325387892, 0.9288167534616818, 0.25124152941221617, 0.7148530485811414, 0.774585036558480 9, 0.6634916192530753, 0.8484278636492283, 0.4886724905647722, 0.9495124126139804, 0.18216664544264263, 0.566131443177105 2, 0.933381202448590 6, 0.2362867867840 1224, 0.685730296466136, 0.818916176503 4846]

In [202]:

```
display(dynamics_graph);
```

WebIO not detected.

Please read the troubleshooting guide for more information on how to resolve this issue.

https://juliagizmos.github.io/WebIO.jl/latest/troubleshooting/not-detected/

In [202]:

```
display(dynamics_graph);
```

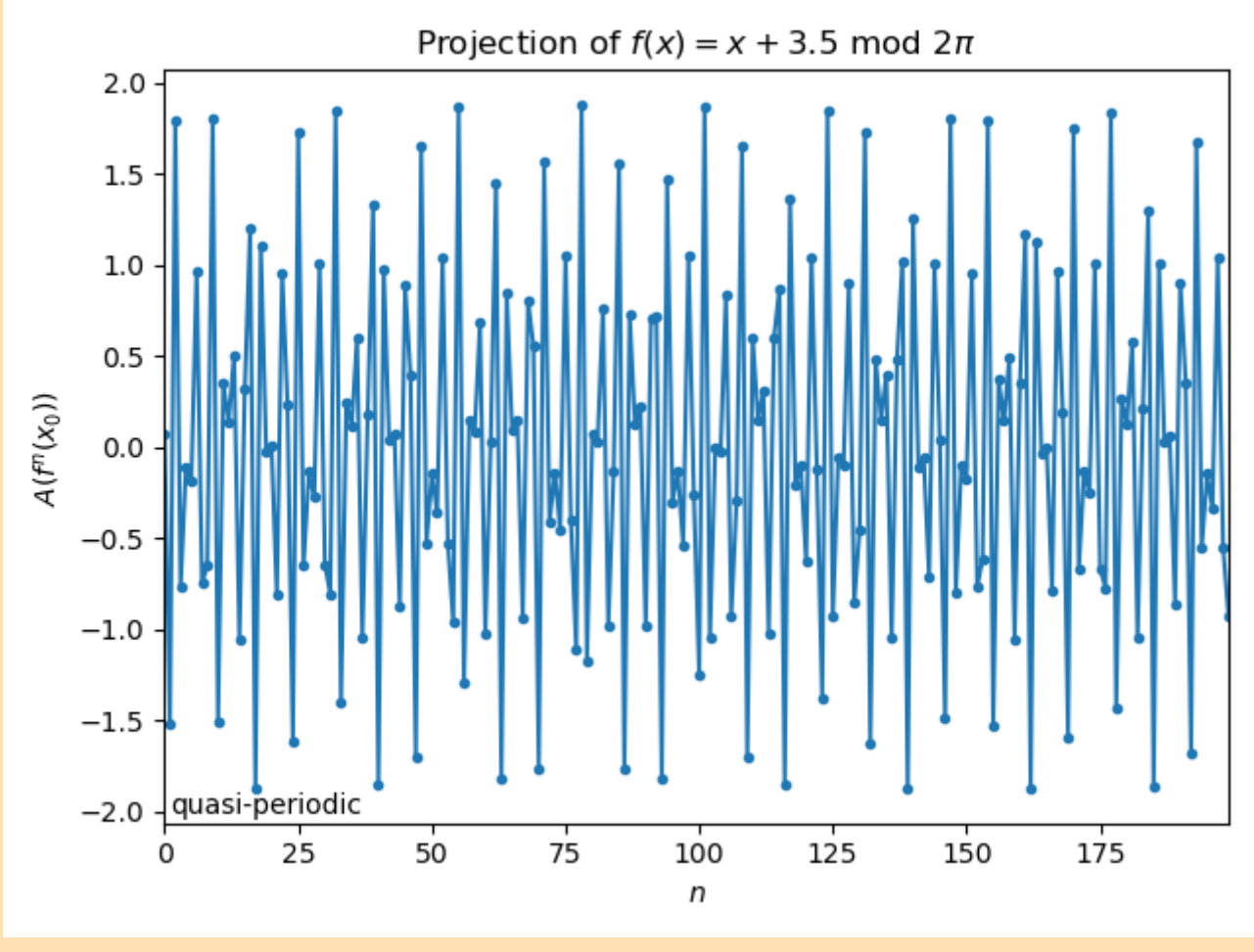
WebIO not detected.

Please read the troubleshooting guide for more information on how to resolve this issue.

https://juliagizmos.github.io/WebIO.jl/latest/troubleshooting/not-detected/

Thought exercise: What appear to be some features of chaotic time series?

Thought exercise 2: Can you distinguish a chaotic time series from other things (e.g. a signal from a irrational rotation):



What does a generic trajectory tell us?

Let's assume we have a discrete time system $f : M \to M$. Continuous time usually conceptually similar but some more issues.

Physical measures

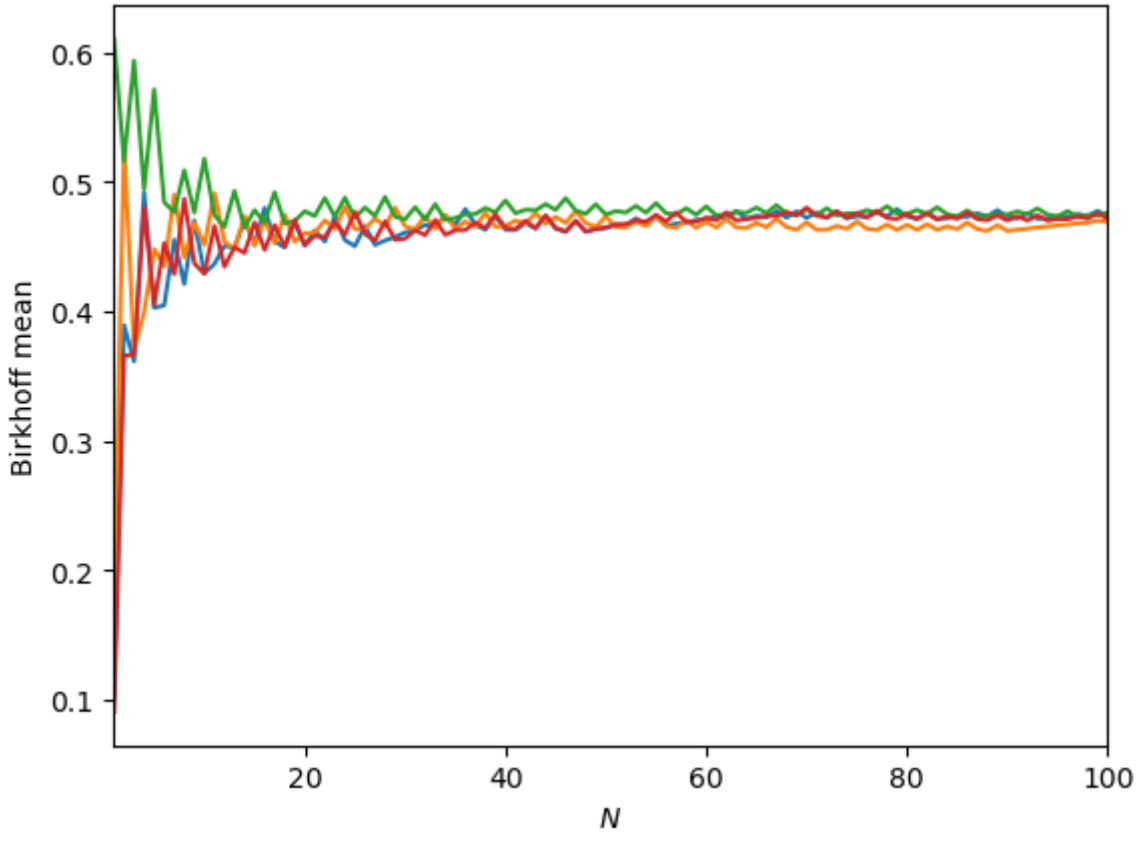
A *physical measure* is a measure that describes the statistical behaviour of a some set of initial conditions E with $\mathrm{Leb}(E) > 0$. (Often $\bar{E} = M$, or if not $\operatorname{int} \bar{E}$ is a basin of attraction for some attractor.)

That is, for all $x \in E$ and continuous $A : M \to \mathbb{R}$,

$$\lim_{N\to\infty} \frac{1}{N} \sum_{n=0}^{N-1} A(f^n(x)) = \int_M A(y)\,\mathrm{d}\rho(y).$$

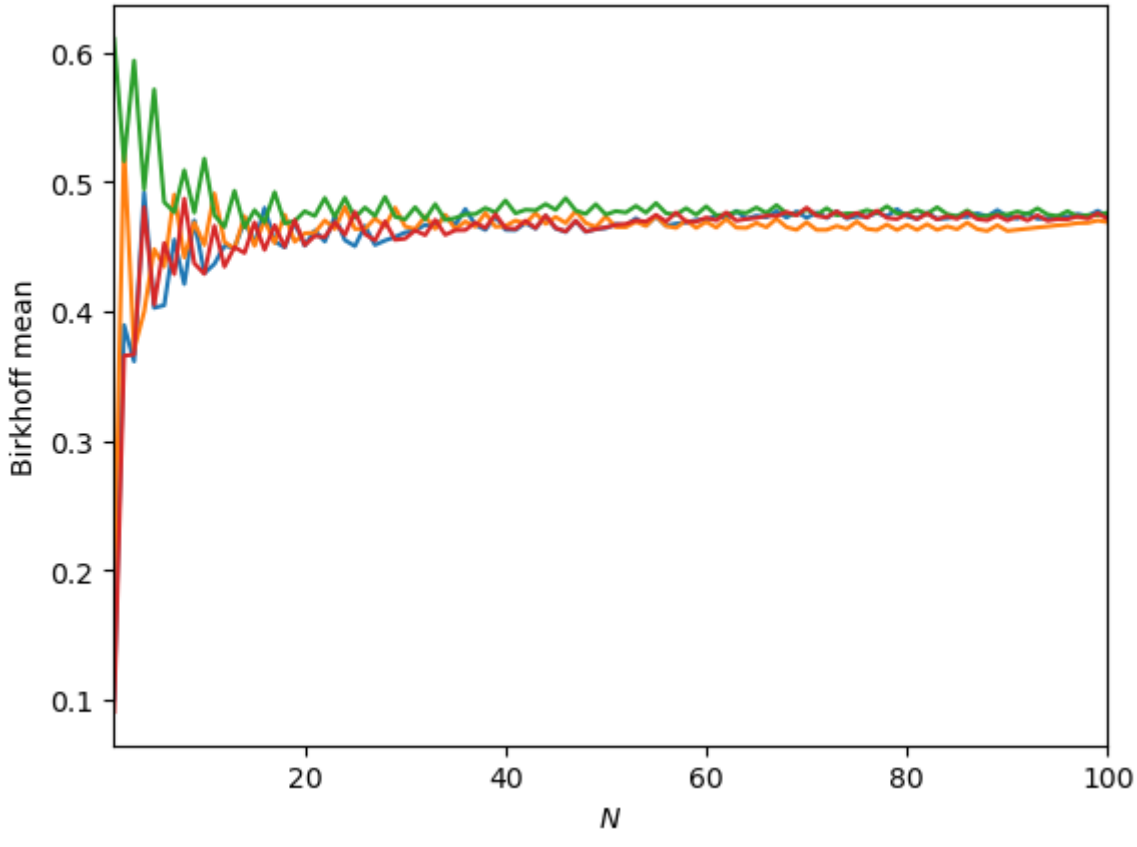
In [229]:

```
using Statistics
cummean(v) = cumsum(v) ./ eachindex(v)

A(x) = x^2
for i = 1:4
    Nmax = 100
    xh = Array{Float64}(undef,Nmax) # initialises a vector to put values in
    a = 3.8 # logistic parameter (chaotic we think)

    xh[1] = rand() # should select a Lebesgue-generic point
    for n = 1:Nmax-1
        xh[n+1] = logistic(xh[n],a)
    end

    plot(1:Nmax, cummean(A.(xh)))
    xlabel("\$N\$"); xlim(1,Nmax)
    ylabel("Birkhoff mean")
end
```


0.6
0.5
0.4
0.3
0.2
0.1
Birkhoff mean
20
40
60
80
100
N

In [229]:

```
using Statistics
cummean(v) = cumsum(v) ./ eachindex(v)

A(x) = x^2
for i = 1:4
    Nmax = 100
    xh = Array{Float64}(undef,Nmax) # initialises a vector to put values in
    a = 3.8 # logistic parameter (chaotic we think)

    xh[1] = rand() # should select a Lebesgue-generic point
    for n = 1:Nmax-1
        xh[n+1] = logistic(xh[n],a)
    end

    plot(1:Nmax, cummean(A.(xh)))
    xlabel("\$N\$"); xlim(1,Nmax)
    ylabel("Birkhoff mean")
end
```



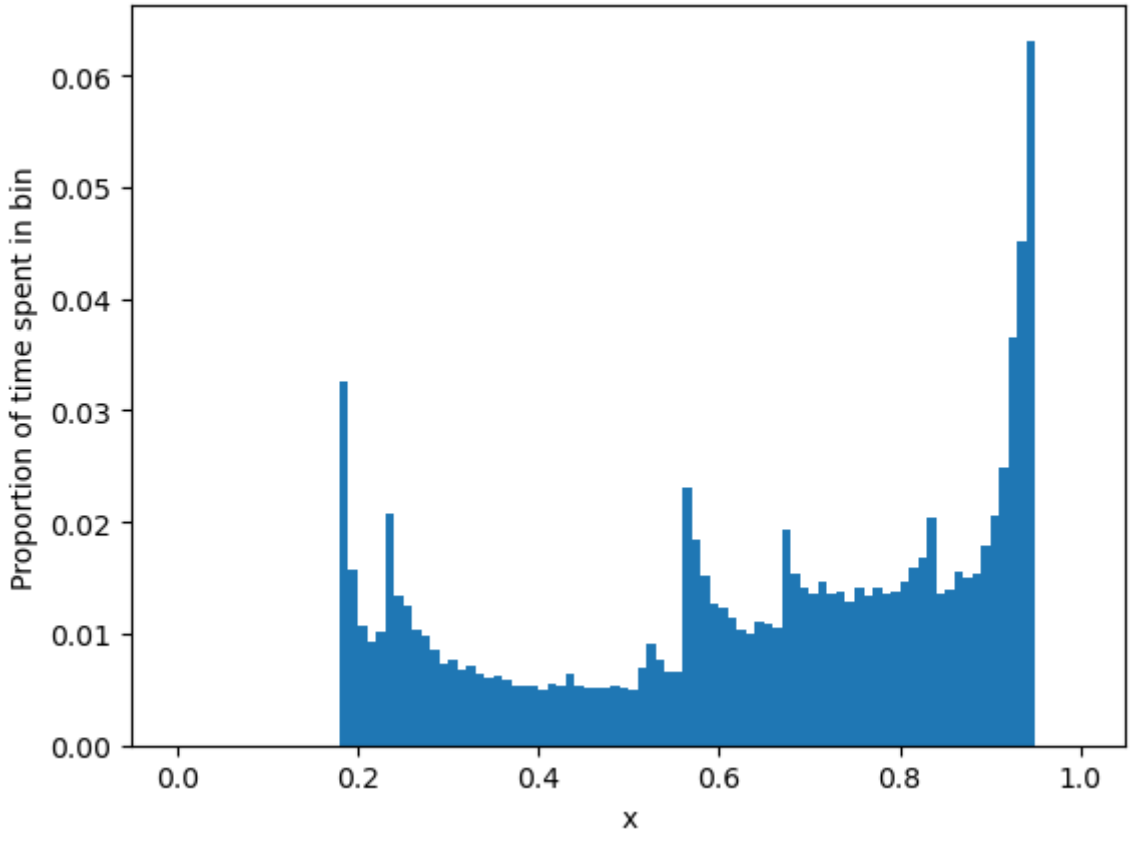
How do you sample from Lebesgue measure? Use random number generator: `rand()`

$$\frac{1}{N}\sum_{n=0}^{N-1} 1_E(f^n(x)) \to \int_M 1_E \, d\rho = \rho(E)$$

for each bin $E \subset [0,1]$.

In [232]:

```
N = 100000
xh = Array{Float64}(undef,N) # initialises a vector to put values in
a = 3.8 # logistic parameter

xh[1] = rand() # should select a Lebesgue-generic point
for n = 1:N-1
    xh[n+1] = logistic(xh[n],a)
end

using PyPlot
hist(xh,bins=0:0.01:1,weights=fill(1/N,N)); xlabel("x"); ylabel("Proportion of time spent in bin");
```


Proportion of time spent in bin
0.06
0.05
0.04
0.03
0.02
0.01
0.00
0.0
0.2
0.4
0.6
0.8
1.0
x

In [208]:

```
display(physmeas_graph)
```

WebIO not detected.

Please read the troubleshooting guide for more information on how to resolve this issue.

https://juliagizmos.github.io/WebIO.jl/latest/troubleshooting/not-detected/

How to test for chaotic dynamics?

There are a lot of ways to test for chaos using trajectory information. Perhaps the most obvious is to test for positive Lyapunov exponents:

$$L_{exp}(x_0) = \lim_{N\to\infty} \frac{\log \|Df^N(x_0)\|}{N}$$

This measures how fast nearby orbits move away from each other.

$$\|f^N(y_0) - f^N(x_0)\| = \|Df^N(x_0)(y_0 - x_0)\| + \mathcal{O}(\|y_0 - x_0\|^2)$$

Options:

- $L_{exp} > 0$: chaos*
- $L_{exp} < 0$: convergence to a fixed point, or in discrete time, a periodic orbit
- $L_{exp} = 0$: more information necessary

In one dimension, i.e. is a Birkhoff average.

This is how we've been testing for chaos:

In [209]:

```
# a slightly simplified version
function lyap_logistic(a,# logistic parameter
                       N=10^5, # length of time to run
                       x0=rand()) # starting point
    lyap_birkhoff_sum = 0.
    for i = 1:N
        lyap_birkhoff_sum += log(a*abs(1-2x0)) # log |f'| for the logistic map
        x0 = logistic(x0,a)
    end
    lyap_birkhoff_sum / N # to get birkhoff mean = Lyapunov exponent estimate
end;
```

In one dimension,

i.e. is a Birkhoff average.

This is how we've been testing for chaos:

In [209]:

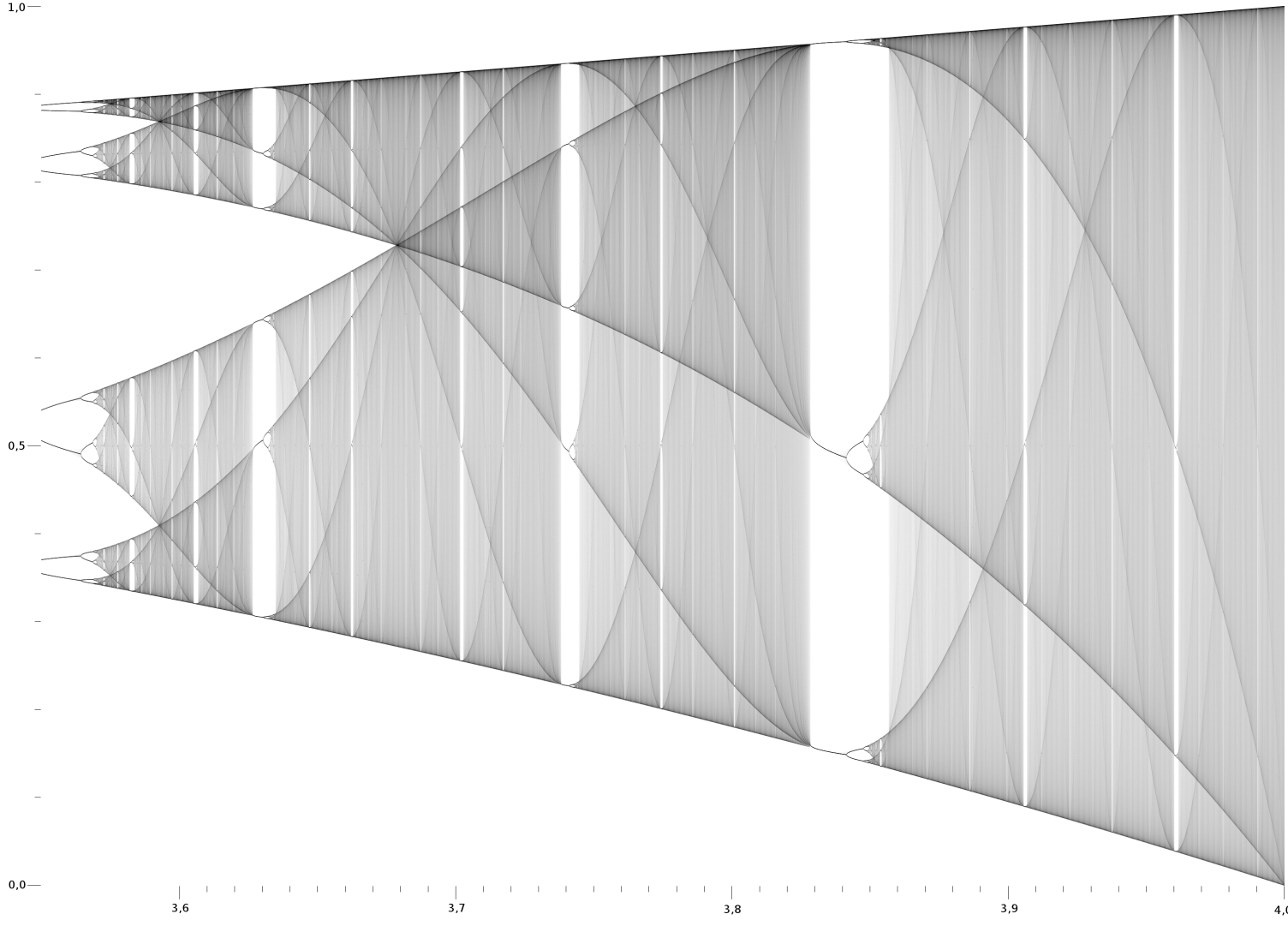
```
# a slightly simplified version
function lyap_logistic(a,# logistic parameter
                       N=10^5, # length of time to run
                       x0=rand()) # starting point
    lyap_birkhoff_sum = 0.
    for i = 1:N
        lyap_birkhoff_sum += log(a*abs(1-2x0)) # log |f'| for the logistic map
        x0 = logistic(x0,a)
    end
    lyap_birkhoff_sum / N # to get birkhoff mean = Lyapunov exponent estimate
end;
```

But we are assuming that a finite estimate is a good approximation of the limit as . Is that fair?

Regular vs chaotic

For the logistic map, you have more or less two options:

Theorem (Jakobsen '81, Lyubich '97, Lyubich '02): The logistic parameters [0,4] are a disjoint union $\mathcal{A}_C \cup \mathcal{A}_P \cup \mathcal{A}_{AP}$, where: * $\mathcal{A}_C$ consists of parameters with chaotic orbits (positive Lyapunov exponents) almost everywhere. It is nowhere dense in $[0, 4]$, and of positive Lebesgue measure. * $\mathcal{A}_P$ consists of parameters with globally stable periodic orbits. It is open and dense in $[0, 4]$, and of positive Lebesgue measure. * $\mathcal{A}_{AP}$ has zero Lebesgue measure in $[0, 4]$.



InXnI, Wikimedia Commons

The set of chaotic parameters is not nice to work with computationally: arbitrarily small perturbations give regular dynamics!

(The chaotic parameters are *structurally unstable*.)

Why is this?

The set of chaotic parameters is not nice to work with computationally: arbitrarily small perturbations give regular dynamics!

(The chaotic parameters are *structurally unstable*.)

Why is this?

There's no structural guarantee in the logistic map against stable periodic orbits ("sinks").

Thus, you can only have chaos when you don't have a sink. This happens a positive proportion of the time.

The set of chaotic parameters is not nice to work with computationally: arbitrarily small perturbations give regular dynamics!

(The chaotic parameters are *structurally unstable*.)

Why is this?

There's no structural guarantee in the logistic map against stable periodic orbits ("sinks").

Thus, you can only have chaos when you don't have a sink. This happens a positive proportion of the time.

Numerical exercise: try and estimate the measure of .

However, it is very easy to perturb a chaotic orbit to get a sink. Take the orbit of the critical point 0.5:

$$(f^N)'(0.5) = f'(0.5)(f^{N-1})'(f(0.5)) = 0$$

So we always have a sink when 0.5 is periodic.

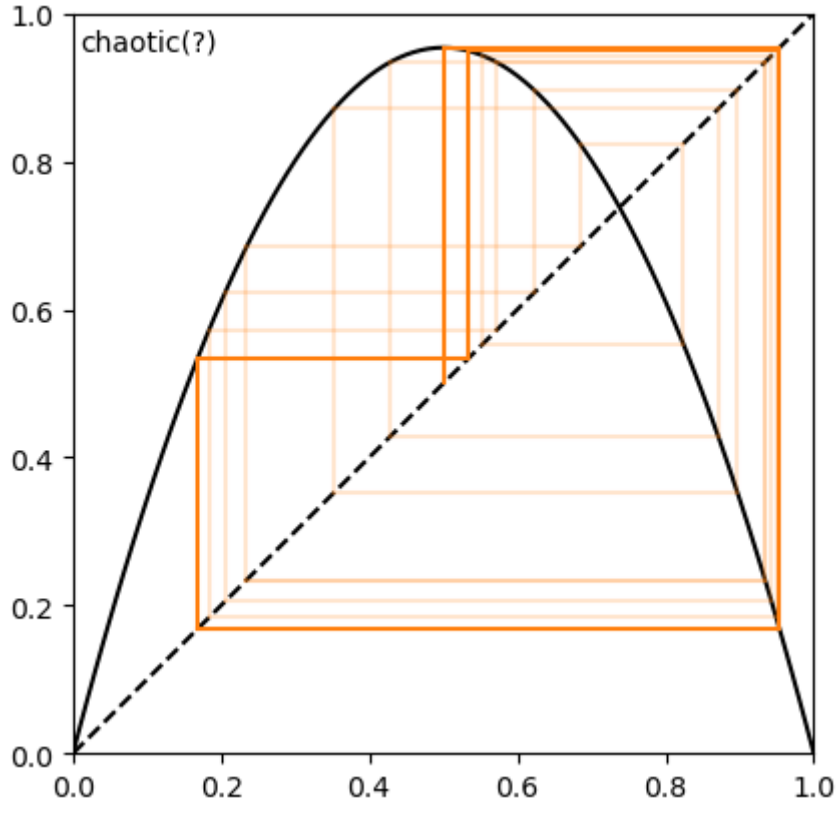
In [211]:

```
display(periodicorbitgraph(4,3.8:0.005:3.83));
```

WebIO not detected.

Please read the troubleshooting guide for more information on how to resolve this issue.

https://juliagizmos.github.io/WebIO.jl/latest/troubleshooting/not-detected/


chaotic(?)
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

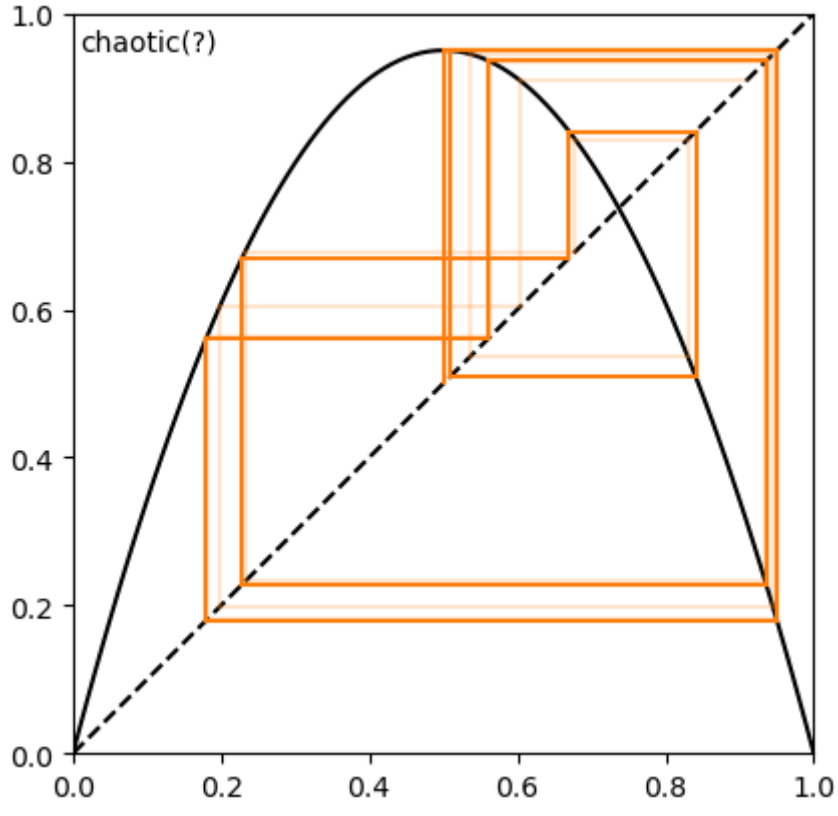
In [212]:

```
display(periodicorbitgraph(9,3.8:0.0002:3.801));
```

WebIO not detected.

Please read the troubleshooting guide for more information on how to resolve this issue.

https://juliagizmos.github.io/WebIO.jl/latest/troubleshooting/not-detected/


chaotic(?)
1.0
0.8
0.6
0.4
0.2
0.0
0.0
0.2
0.4
0.6
0.8
1.0

If there isn't a sink, then the forward orbit of f is very unstable (according to the Lyapunov exponent). So if we perturb the parameter a little then we can move the orbit of 0.5 so as to recur and create a sink:

$$f_a^N(0.5) - 0.5 = (f_{a_0}^N(0.5) - 0.5) + \frac{\partial f_{a_0}^N}{\partial a_0}(0.5)(a - a_0) + \text{h. o. t.}.$$

But

$$\frac{\partial f_{a_0}^N}{\partial a_0}(0.5) = \sum_{n=0}^{N-1} (f_{a_0}^{N-n})'(f^{n+1}(0.5)) \frac{\partial f_{a_0}}{\partial a_0}(f^n(0.5)) \sim L_{exp}^{N-1}.$$

So (for certain N's at least) for $a \sim L_{exp}^{-N} a_0$ we can make 0.5 periodic of order N, and therefore a sink, so $a \in \mathcal{A}_P$.

Of course, this is more complicated to prove.

Numerical exercise: find a logistic parameter with a stable orbit of length 51. (Hint: you will want to use a root-finding package)

So how do you know you if have chaos for a given (non-special) logistic map or not?

You can show that you *do* have chaos by saying something about (e.g. fixing pointwise Lyapunov exponent). But you need to know its behaviour for all time (usually impossible).

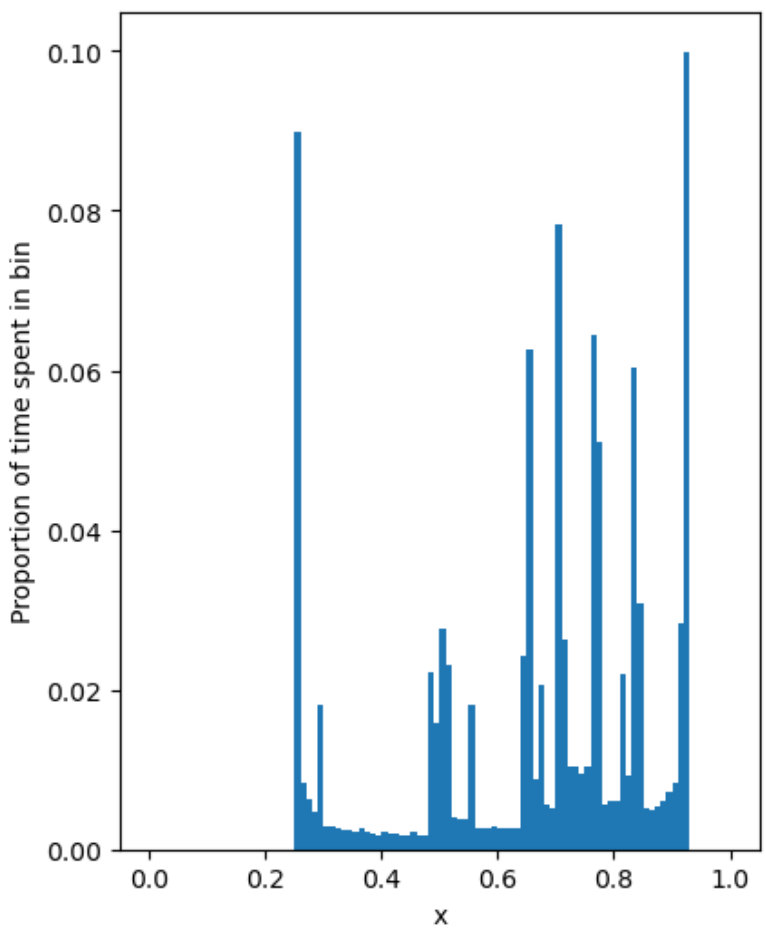
So how do you know you if have chaos for a given (non-special) logistic map or not?

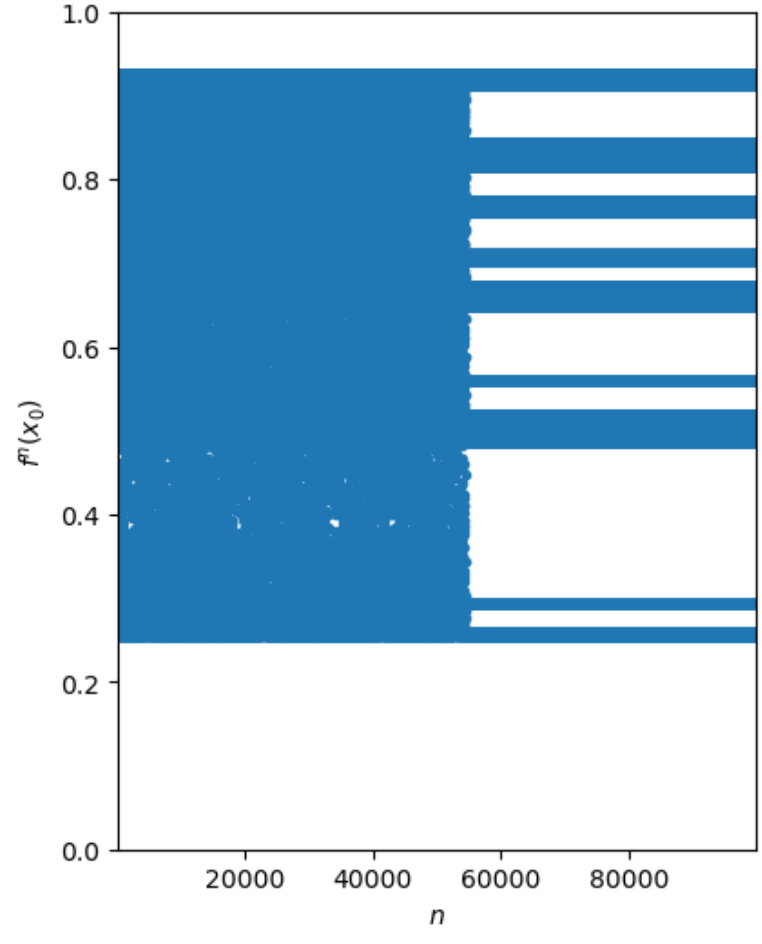
You can show that you *do* have chaos by saying something about (e.g. fixing pointwise Lyapunov exponent). But you need to know its behaviour for all time (usually impossible).

You can show that you *don't* have chaos by finding a stable periodic orbit. But this could take an arbitrarily long time, and you have no way to know when to give up.

In [241]:

```
N = 100000
xh = Array{Float64}(undef,N) # initialises a vector to put values in
a = 3.7030314384 # logistic parameter

xh[1] = rand() # should select a Lebesgue-generic point
for n = 1:N-1
    xh[n+1] = logistic(xh[n],a)
end

figure(1,figsize=(10,6))
subplot(121)
hist(xh,bins=0:0.01:1,weights=fill(1/N,N)); xlabel("x"); ylabel("Proportion of time spent in bin");
subplot(122)
xlim(1,N-1); xlabel("\$n\$"); ylim(0,1);ylabel("\$f^n(x_0)\$");
plot(1:N,xh[1:end],".");

println("estimated Lyapunov exponent = ",lyap_logistic(a,N))
```






```
estimated Lyapunov exponent = 0.0893804304351754
```

In [214]:

```
logistic_inv(x,y,a) = 0.5 + flipsign(sqrt(max(0.25-x/a,0)),y-0.5)
        # the pre-image of the logistic map that is closest to y (or if no pre-image exists, the critical point)
```

Out[214]:

```
logistic_inv (generic function with 2 methods)
```

In [214]:

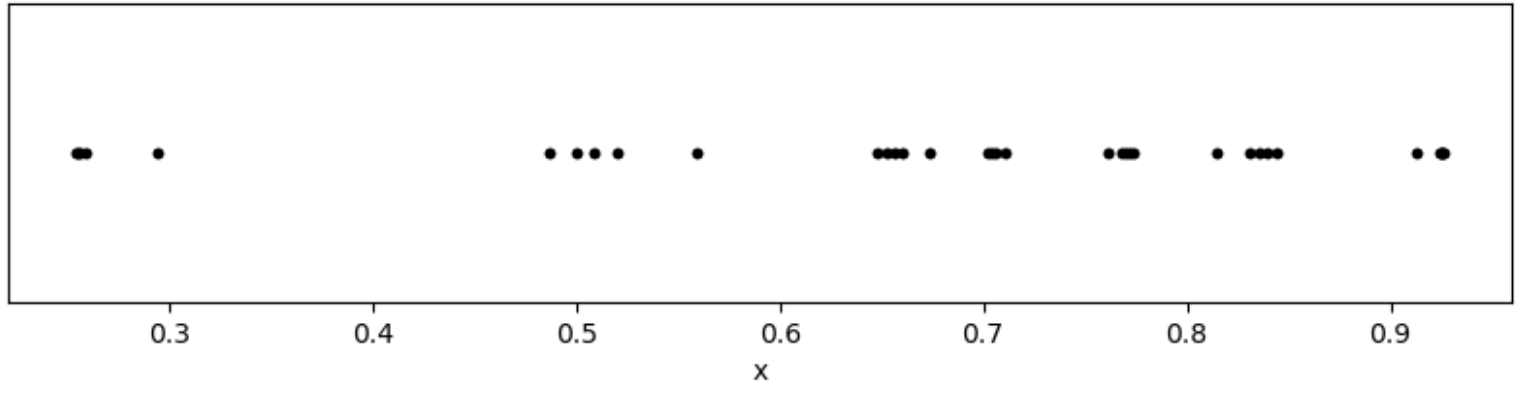
```
logistic_inv(x,y,a) = 0.5 + flipsign(sqrt(max(0.25-x/a,0)),y-0.5)
        # the pre-image of the logistic map that is closest to y (or if no pre-image exists, the critical point)
```

Out[214]:

```
logistic_inv (generic function with 2 methods)
```

In [215]:

```
x = xh[end]+1e-9 # a small perturbation of our stable period 34 orbit
for y = repeat(xh[end-1:-1:end-34],100001)
    x = logistic_inv(x,y,a)
end # iterate backwards close to the unstable periodic orbit, to try and find a stable periodic orbit
for y = repeat(xh[end-1:-1:end-34],1)
    x = logistic_inv(x,y,a)
    abs(x-0.5) < 1e-3 && break
end
xw = Array{Float64}(undef,34,2)
xw[1,:] = [x,0.5]
for i = 1:33
    xw[i+1,:] = logistic.(xw[i,:],a)
end
figure(figsize=(10,2))
plot(xw',zeros(size(xw')),"k.-");
xlabel("x"); yticks([])
```



Out[215]:

```
(Any[], Any[])
```

The globally attracting periodic dynamics is qualitatively visible on a set of Lebesgue measure

In [216]:

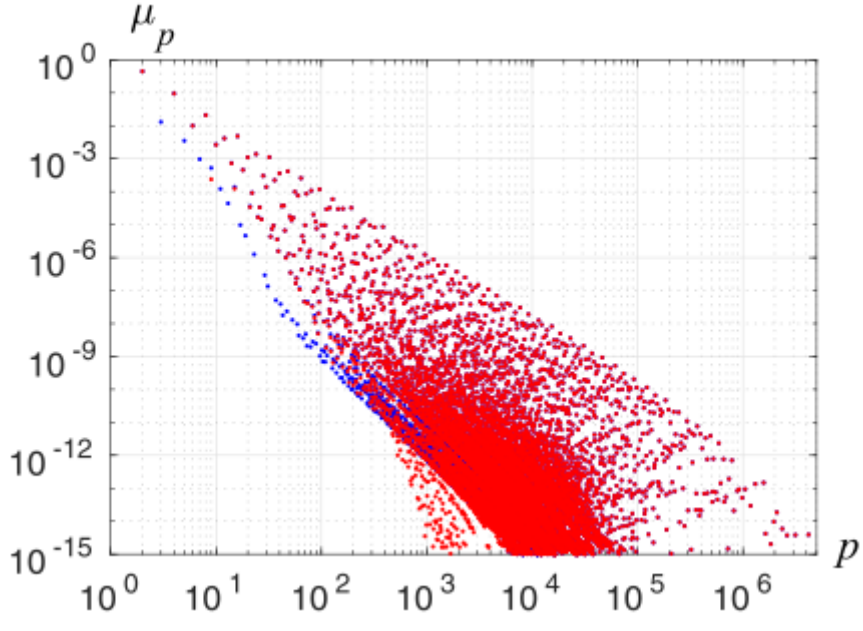
```
sum(abs.(xw[1:end,1] - xw[1:end,2]))
```

Out[216]:

```
2.6353601171869023e-5
```

However, a randomly selected periodic parameter is likely to be of low order (numerical study by Galias '17):

Galias '17



Stable periodic orbits of low order are relatively easy to rule out (e.g. by running a long integration), one can make a "probabilistic" guess of chaos.

Some notes on iteration

Most computer operations take place in floating point arithmetic. This introduces errors.

Most computer operations take place in floating point arithmetic. This introduces errors.

In [246]:

```
Float64(π)*Float64(π)
```

Out[246]:

```
9.869604401089358
```

Most computer operations take place in floating point arithmetic. This introduces errors.

In [246]:
```
Float64(π)*Float64(π)
```

Out[246]:
```
9.869604401089358
```

In [247]:
```
BigFloat(π) * BigFloat(π)
```

Out[247]:
```
9.869604401089358618834490999876151135313699407240790626413349376220044822419144
```

Most computer operations take place in floating point arithmetic. This introduces errors.

In [246]:
```
Float64(π)*Float64(π)
```

Out[246]:
```
9.869604401089358
```

In [247]:
```
BigFloat(π) * BigFloat(π)
```

Out[247]:
```
9.869604401089358618834490999876151135313699407240790626413349376220044822419144
```

In [219]:
```
(500/27 + 500)*27 - 500*27 # should equal 500
```

Out[219]:
```
499.9999999999982
```

This is usually not a problem for sampling the physical measure accurately.

The usual justification is:

```
The discretisation error behaves like a noise, and physical measures are usually stable to the
introduction of a small dynamical noise ("stochastically stable").
```

(See also recent work of Guihéneuf and collaborators.)

This is why we talk about the physical measure: other invariant measures will not be sampled efficiently.

0In practice, the discretisation of the dynamical system has a finite state space (usually of size $P \approx (10^{16})^{\text{dimension}}$) of which only maybe $\sqrt{P}$ points will be sampled in the limit (Lanford '98).

So don't try and iterate too long (in double floating point).

Beware of this sort of thing too:

In [253]:

```
doubling(x) = mod(2x,1) # doubling map f(x) = 2x mod 1
x = rand()
for i = 1:100
    print(x,", ")
    x = doubling(x)
end
```

```
0.2889500823917614 5, 0.5779001647835229, 0.15580032956704581, 0.3116006591340916 3, 0.6232013182681833, 0.2464026365363663 6651, 0.4928052730727330 3, 0.985610546145466 1, 0.9712210922909321, 0.9424421845818642, 0.8848436916372 85, 0.769768738327457, 0.5395374766549139, 0.0790749533 0982783, 0.1581499066196556 7, 0.3162998132393113 3, 0.6 32599626478622 7, 0.2651992529572453 3, 0.5303985059144907, 0.0607970118289813 4, 0.1215940236579626 8, 0.2431880473159253 6, 0.4863760946318507, 0.9727521892637014, 0.9455043785274029, 0.8910087570548058, 0.7820175141096115, 0.56403502821922 3, 0.1280700564384460 4, 0.2561401128768921, 0.5122802257537842, 0.0245604515075683 6, 0.0491209 0301513672, 0.0982418060302734 4, 0.1964836120605468 8, 0.3929672241210937 5, 0.7859344482421875, 0.5718688964843 75, 0.1437377929687 5, 0.2874755859375, 0.574951171875, 0.1499023437 5, 0.2998046875, 0.599609375, 0.19921875, 0.3984375, 0.796875, 0.59375, 0.1875, 0.375, 0.75, 0.5, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
```

```
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0, 0.0,
0.0, 0.0, 0.0, 0.0,
```

Easy to fix (using stochastic stability of physical measure):

In [221]:

```
doubling(x) = mod(2x,1) # doubling map f(x) = 2x mod 1
x = rand()
for i = 1:100
    print(x,", ")
    x = doubling(x) + 10eps()*randn()
end
```

```
0.764245094529198, 0.528490189058400, 0.056980378116800
88, 0.113960756233603, 0.227921512467207, 0.455843
02493441725, 0.911686049868833, 0.823372099737668, 0.6
4674744199475337, 0.293488398950673, 0.586976797901347,
0.173953595802696, 0.347907191605395, 0.695814383210
7883, 0.391628766421576, 0.783257532843153, 0.566515
0656863049, 0.133030131372611, 0.266060262745226, 0.5
32120525490449, 0.064241050980898, 0.128482101961800
54, 0.256964203923600, 0.513928407847201, 0.02785681
5694397647, 0.055713631388798, 0.111427262777595,
0.222854525555192, 0.445709051110385, 0.89141810222
0772, 0.782836204441545, 0.565672408883093, 0.13134481
776618487, 0.262689635323709, 0.525379271064743, 0.05
07585421294860, 0.101517084258969, 0.20303416851793
57, 0.406068337035876, 0.812136674071754, 0.624273348
1435064, 0.248546696287017, 0.497093392574036, 0.99
4186786785148, 0.988373570296143, 0.976747140592284,
0.953494281184567, 0.906988562369135, 0.81397712473826
```

58, 0.627954249476530 8, 0.2559084989530658 3, 0.5118169979061321, 0.023633995812261402, 0.04726799162452099, 0.094535983249040 8, 0.18907196649807992, 0.3781439329961593, 0.756287865992320 5, 0.512575731984642 2, 0.025151463969283114, 0.0503029279385667 5, 0.10060585587713 34, 0.201211711754267 52, 0.40242342350853 88, 0.8048468470170 79, 0.609693694034158 3, 0.21938738806831692, 0.43877477613663296, 0.87754955227326 36, 0.75509910454652 94, 0.5101982090930568, 0.020396418186115983, 0.040792836372231 77, 0.08158567274446284, 0.16317134548892437, 0.3263426909778515, 0.65268538195569 85, 0.30537076391139634, 0.6107415278227926, 0.22148305564558 26, 0.44296611291161 37, 0.88593222258232 03, 0.77186444516464 23, 0.5437288903292848, 0.08745778065856 79, 0.17491556131713 97, 0.3498311226342802, 0.69966224526855 94, 0.39932449053712 07, 0.7986489810742387, 0.59729796214847 35, 0.1945959242969477, 0.389191848593897 7, 0.77838369718779 59, 0.5567673943755912, 0.11353478875118042, 0.22706957750236037, 0.4541391550047178 5, 0.90827831000943 65, 0.8165566200188765,

Like our calculations, computer representations are finite.

We have to make finite approximations of:

- Space/position
- Time (if applicable)
- Numbers! (e.g. in $\mathbb{R}, \mathbb{C}$)

We are constantly introducing error into our calculations (and the effect of these errors may be hard/impossible to know rigorously).

How rigorous to be?

- As with paper calculations, there are different levels of rigour.
- They are all useful!
    - We regularly make mathematical hypotheses based on inductive (scientist-style) reasoning.

Suppose we use algorithm to compute proposition . We could have:

1. is definitely, mathematically true (i.e. constitutes a proof).

Example: The Lorenz flow is a Geometric Lorenz flow (Tucker 1999)

Suppose we use algorithm to compute proposition . We could have:

1. is definitely, mathematically true (i.e. constitutes a proof).

Example: The Lorenz flow is a Geometric Lorenz flow (Tucker 1999)

1. That converges is a theorem, (1) would be true if we computed the (small) approximation errors explicitly.

Example: Running some proven-to-work approximation algorithm but not keeping track of the errors.

Suppose we use algorithm to compute proposition . We could have:

1. is definitely, mathematically true (i.e. constitutes a proof).

Example: The Lorenz flow is a Geometric Lorenz flow (Tucker 1999)

1. That converges is a theorem, (1) would be true if we computed the (small) approximation errors explicitly.

Example: Running some proven-to-work approximation algorithm but not keeping track of the errors.

1. We have a good idea of how to prove converges, (2) would be true if we did that.

Example: Minor extensions of existing algorithms. "What if we used a Lipschitz observable instead of like in the theorem"

1. would converge if clearly true condition holds, (2) or (3) would be true if we could prove .

Example: Assuming a dynamical system that appears to be chaotic, exponentially mixing, etc, is actually those things

1. would converge if clearly true condition holds, (2) or (3) would be true if we could prove .

Example: Assuming a dynamical system that appears to be chaotic, exponentially mixing, etc, is actually those things

1. (2) or (3) is true in an analogous setting, and would be true if we could extend it to our setting.

Example: Applying an algorithm proven for Anosov maps to a non-uniformly hyperbolic map

1. would converge if clearly true condition holds, (2) or (3) would be true if we could prove .

Example: Assuming a dynamical system that appears to be chaotic, exponentially mixing, etc, is actually those things

1. (2) or (3) is true in an analogous setting, and would be true if we could extend it to our setting.

Example: Applying an algorithm proven for Anosov maps to a non-uniformly hyperbolic map

1. We have some formal calculation/intuition that should compute (usually plus some evidence in practice).

Example: Dynamic mode decomposition, etc

All of these are useful for both mathematicians and scientists!

Non-uniformly hyperbolic systems will almost always fall into cases 4–6.

All of these are useful for both mathematicians and scientists!

Non-uniformly hyperbolic systems will almost always fall into cases 4–6.

General exercises: find or recall examples of numerics that you have seen corresponding to cases 1–6.

In [ ]:

In [ ]: