This packages does a little better than quaduple precision, so it is not arbitrary precision, just more precision.

```
> install.packages("Rmpfr")
> library(Rmpfr)
> exp(1)
[1] 2.718282
> options(digits=20)
> exp(1)
[1] 2.7182818284590450908
> pi
[1] 3.1415926535897932
> mypi <- mpfr(pi,120)
> mypi
1 'mpfr' number of precision  120   bits
[1] 3.1415926535897931159979634685441851616
```

For large factorials, R uses approximations, but the package allows higher precision. The last numbers can't be right for 23! and 24! because numbers like 24! have at least 5, 10, 15, and 20 as factors so must have several 0s in the 1s, 10s, 100s, digits etc.

```
> n <- 1;24
> factorial(n)
 [1]                          1                          2                          6
 [4]                         24                        120                        720
 [7]                       5040                      40320                     362880
[10]                    3628800                    39916800                  479001600
[13]                 6227020800                 87178291200               1307674368000
[16]             20922789888000             355687428096000            6402373705728000
[19]         121645100408832000         2432902008176640000         51090942171709440000
[22]    1124000727777607680000  25852016738884978212864 620448401733239409999872

> pn <- mpfr(1:24, 120)
> factorial(pn)
24 'mpfr' numbers of precision  120   bits
 [1]                          1                          2                          6
 [4]                         24                        120                        720
 [7]                       5040                      40320                     362880
[10]                    3628800                    39916800                  479001600
[13]                 6227020800                 87178291200               1307674368000
[16]             20922789888000             355687428096000            6402373705728000
[19]         121645100408832000         2432902008176640000         51090942171709440000
[22]    1124000727777607680000  25852016738884976640000 620448401733239439360000
```

```
pchaos <- function(x) {
  value <- mpfr(1-abs(1-2*x),120)
  return(value)
}
pchaos(.3) # true answer is 0.6
1 'mpfr' number of precision  120   bits
[1] 0.59999999999999997779553950749686919153

pchaos2 <- function(x,maxterms) {
  value <- rep(-1,maxterms)
  temp <- pchaos(x)
  for(i in 1:maxterms) {
    print(as.numeric(temp))
    value[i] <- as.numeric(temp)
    temp <- pchaos(temp)
  }
}
```

```
> pchaos2(.314,60)
[1] 0.62800000000000000266
[2] 0.74399999999999999467
[3] 0.51200000000000001066
[4] 0.97599999999999997868
[5] 0.04800000000000000042633
[6] 0.09600000000000000085265
[7] 0.19200000000000017053
[8] 0.38400000000000034106
[9] 0.76800000000000068212
[10] 0.46399999999999863576
[11] 0.92799999999999727152
[12] 0.14400000000000545697
[13] 0.28800000000001091394
[14] 0.57600000000002182787
[15] 0.84799999999995634425
[16] 0.30400000000008731149
[17] 0.60800000000017462298
[18] 0.78399999999965075403
[19] 0.43200000000069849193
[20] 0.86400000000139698386
[21] 0.27199999999720603228
[22] 0.54399999999441206455
[23] 0.91200000000111758709
[24] 0.17599999997764825821
[25] 0.35199999995529651642
[26] 0.70399999991059303284
[27] 0.59200000017881393433
[28] 0.81599999964237213135
[29] 0.36800000071525573730
[30] 0.73600000143051147461
[31] 0.52799999713897705078
[32] 0.94400000572204589844
[33] 0.11199998855590820312
[34] 0.22399997711181640625
```

```
[41] 0.6719970703125
[42] 0.656005859375
[43] 0.68798828125
[44] 0.6240234375
[45] 0.751953125
[46] 0.49609375
[47] 0.9921875
[48] 0.015625
[49] 0.03125
[50] 0.0625
[51] 0.125
[52] 0.25
[53] 0.5
[54] 1
[55] 0
[1] 0
[1] 0
[1] 0
[1] 0
[1] 0
```

```
 [1] 0.628000000000000113687 0.743999999999999772626 0.512000000000000454747
 [4] 0.975999999999999090505 0.048000000000001818989 0.096000000000003637979
 [7] 0.192000000000007275958 0.384000000000014551915 0.768000000000029103830
[10] 0.463999999999941792339 0.927999999999883584678 0.144000000000232830644
[13] 0.288000000000465661287 0.576000000000931322575 0.847999999998137354851
[16] 0.304000000003725290298 0.608000000007450580597 0.783999999985098838806
[19] 0.432000000029802322388 0.864000000059604644775 0.271999999880790710449
[22] 0.543999999761581420898 0.912000000476837158203 0.175999999046325683594
[25] 0.351999998092651367188 0.703999996185302734375 0.592000007629394531250
[28] 0.815999984741210937500 0.368000030517578125000 0.736000061035156250000
[31] 0.527999877929687500000 0.944000244140625000000 0.111999511718750000000
[34] 0.223999023437500000000 0.447998046875000000000 0.895996093750000000000
[37] 0.208007812500000000000 0.416015625000000000000 0.832031250000000000000
[40] 0.335937500000000000000 0.671875000000000000000 0.656250000000000000000
[43] 0.687500000000000000000 0.625000000000000000000 0.750000000000000000000
[46] 0.500000000000000000000 1.000000000000000000000 0.000000000000000000000
[49] 0.000000000000000000000 0.000000000000000000000
```

extra precision

Increased precision made the sequence fall to 0 later than the default precision in R, but not much later. The sequence hits 0 at iteration 53 instead of iteration 48. That's a lot of work with not too much benefit!

The logistic liar

The following are (admittedly confusing) variations on the chaotic liar:

> It is false that this statement is as true as it is estimated to be false.

> It is very false that this statement is as true as it is estimated to be false.

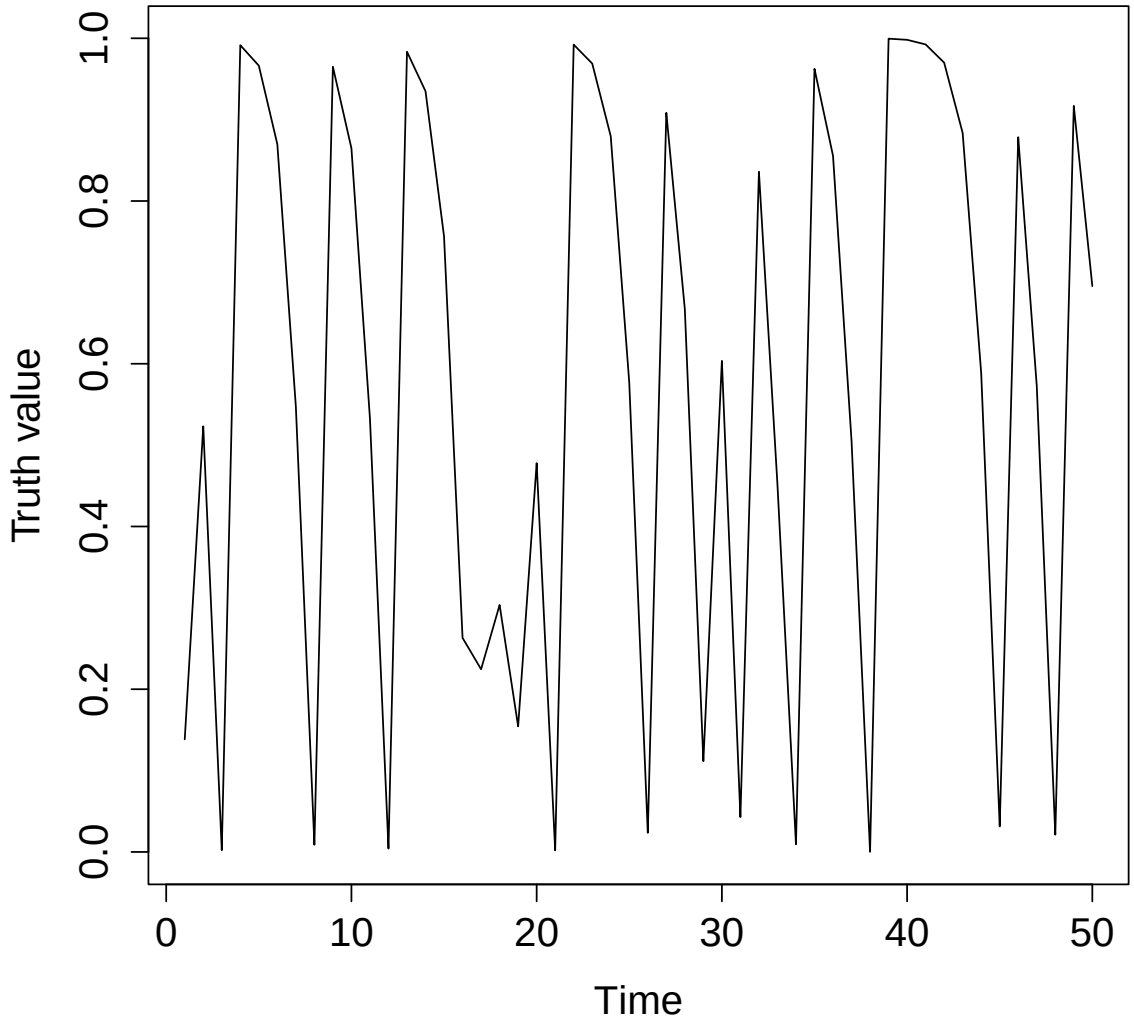
The second function can be written as

$$x_{n+1} = (1 - (1 - \text{Abs}(1 - 2x_n)))^2$$

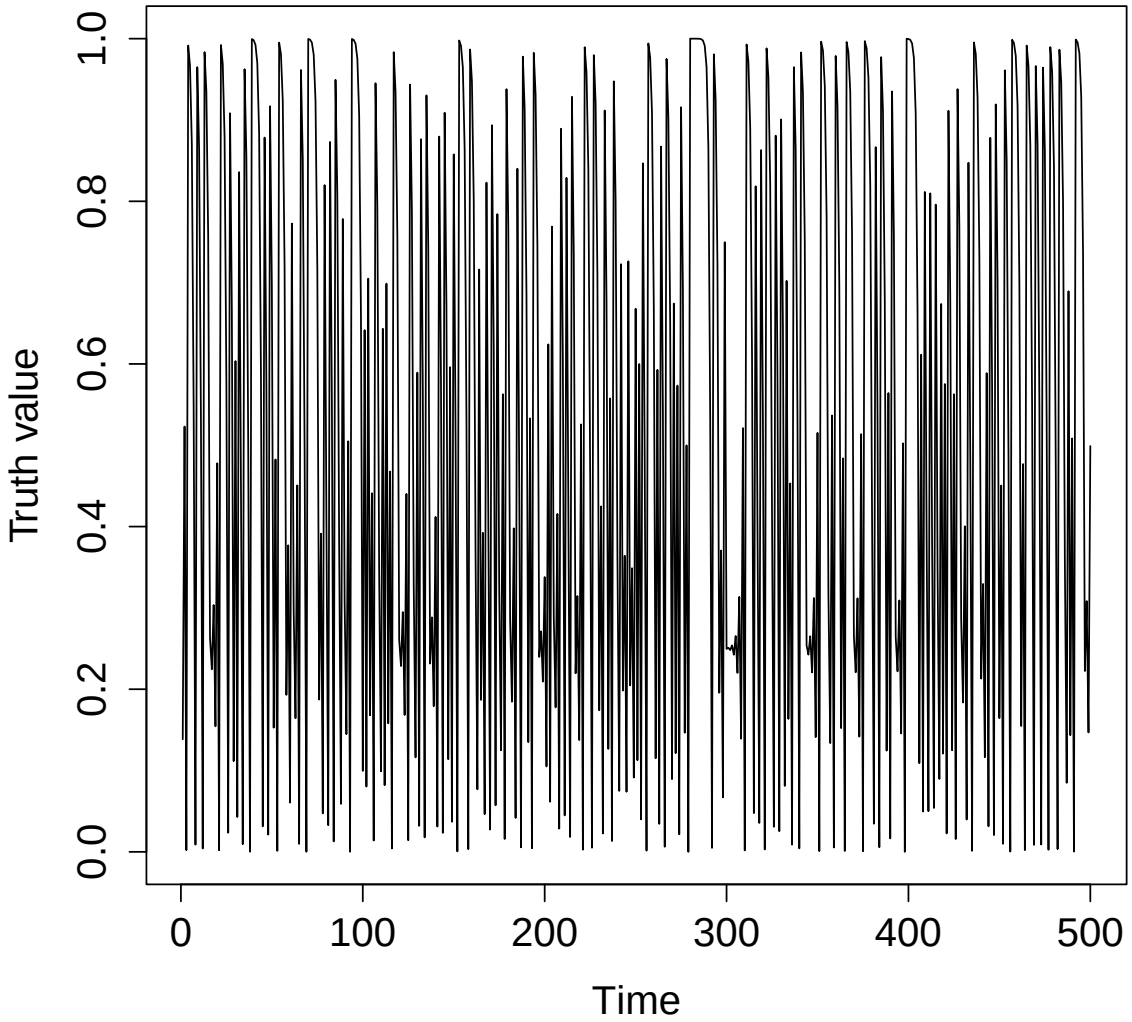
The logistic liar in R

```
logistic <- function(x) {
  value <- (1-(1-abs(1-2*x)))^2
  return(value)
}
logistic2 <- function(x,maxterms) {
  value <- rep(-1,maxterms)
  temp <- logistic(x)
  for(i in 1:maxterms) {
    print(temp)
    value[i] <- temp
    temp <- logistic(temp)
  }
  return(value)
}
```

Time plot of the logistic liar, 50 time points



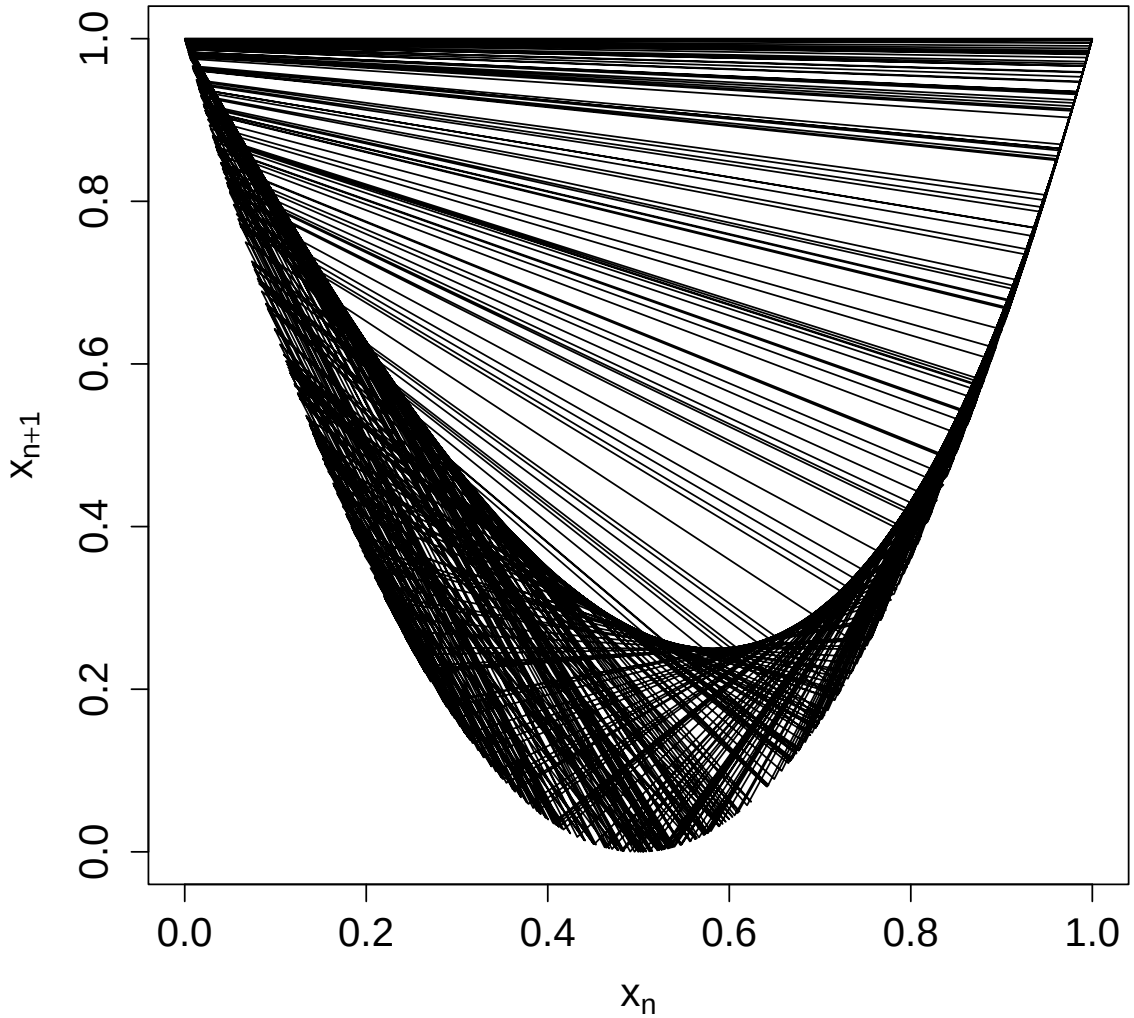
Time plot of the logistic liar, 500 time points



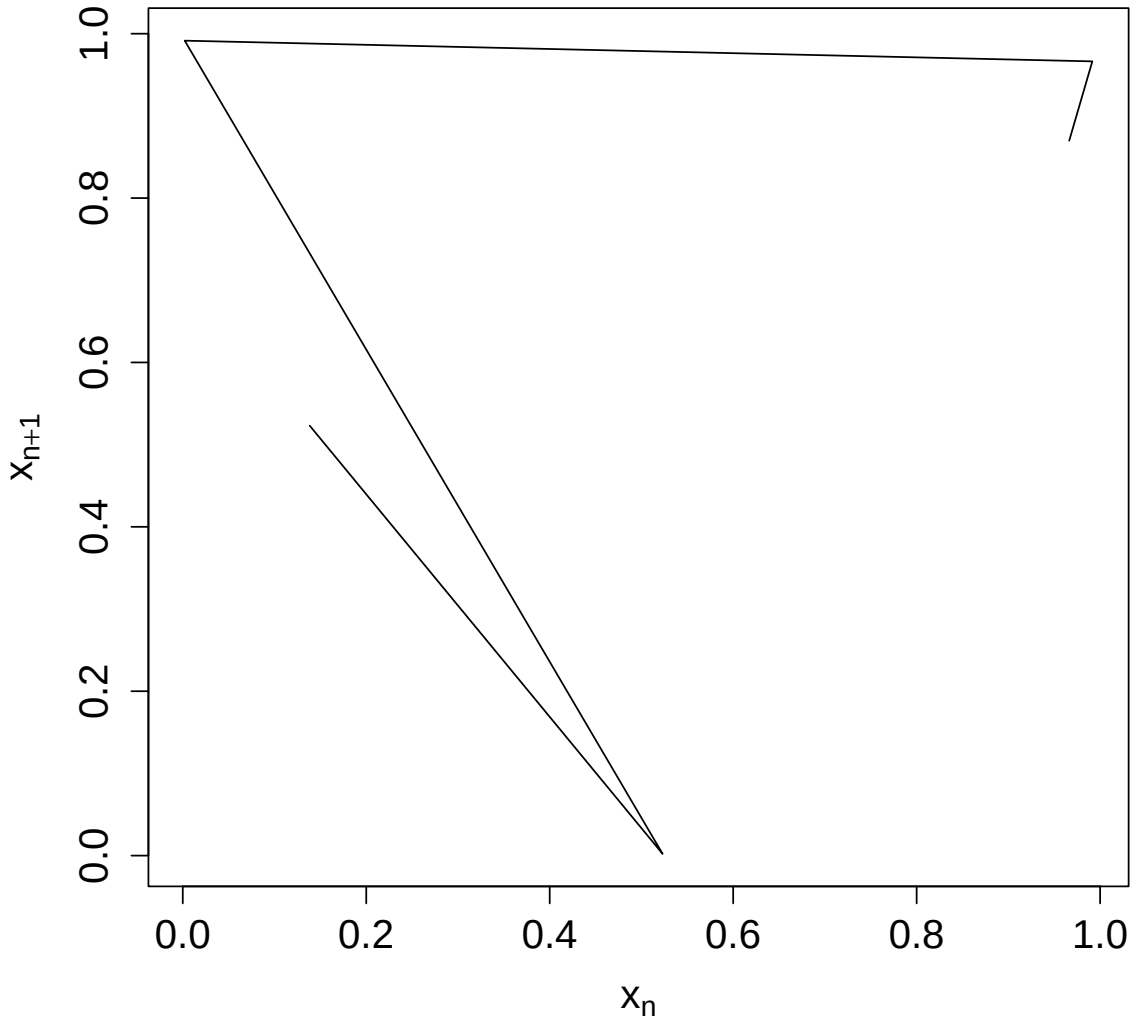
The logistic liar in R

```
> y <- logistic2(.314,500)
> y1 <- y[1:499]
> y2 <- y[2:500]
> plot(y1,y2,type="l",ylab=expression(x[n+1]),
xlab=expression(x[n]),cex.lab=1.4,cex.axis=1.4)
```

Time plot of the logistic liar, 500 time points



Time plot of the logistic liar, 5 time points



The logistic liar can be written in a simpler form. For $x_n < 1/2$

$$\begin{aligned} x_{n+1} &= (1 - (1 - \text{Abs}(1 - 2x_n)))^2 \\ &= (1 - (1 - (1 - 2x_n)))^2 \\ &= (1 - (1 - 1 + 2x_n))^2 \\ &= (1 - 2x_n)^2 \\ &= 1 - 4x_n(1 - x_n) \end{aligned}$$

For $x \geq 1/2$:

$$\begin{aligned} x_{n+1} &= (1 - (1 - \text{Abs}(1 - 2x_n)))^2 \\ &= (1 - (1 - (2x_n - 1)))^2 \\ &= (1 - (2 - 2x_n))^2 \\ &= (-1 + 2x_n)^2 \\ &= (1 - 2x_n)^2 \\ &= 1 - 4x_n(1 - x_n) \end{aligned}$$

Because of the square, even though the absolute value is different depending on whether $x_n < 1/2$ or not, we get the same formula. Thinking about the equation

$$x_n = 1 - 4x_n(1 - x_n)$$

we can imagine changing the coefficient 4 to other values, even if we cant necessarily thinking of a good translation into self-referential statements about truth. If the coefficient is lower than 3, you end up with the sequence converging to a limit. What do you think it converges to?

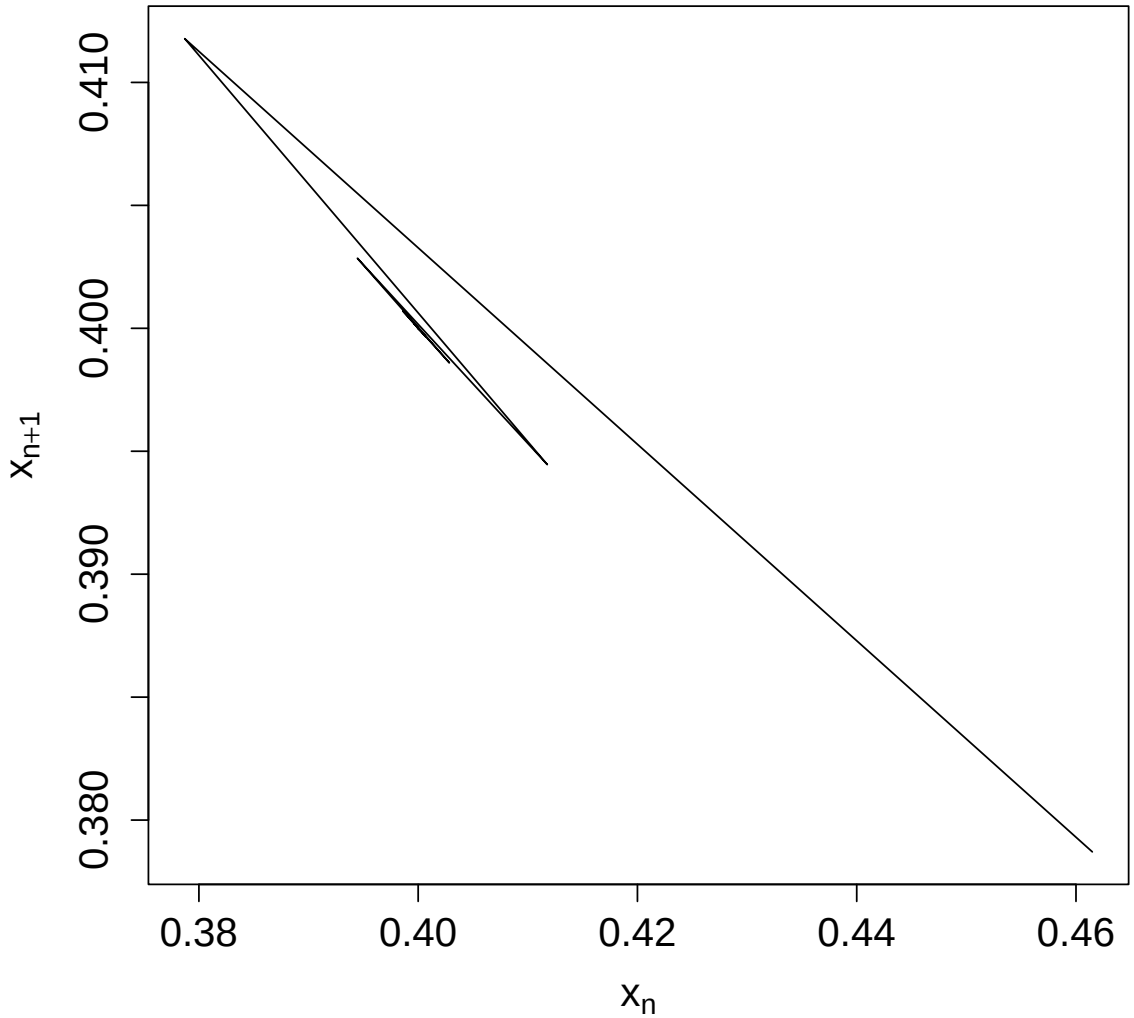
The logistic liar in R

```
#k is the coefficient
klogistic <- function(x,k) {
  value <- 1-k*x*(1-x)
  return(value)
}
klogistic2 <- function(x,k,maxterms) {
  value <- rep(-1,maxterms)
  temp <- klogistic(x,k)
  for(i in 1:maxterms) {
    print(temp)
    value[i] <- temp
    temp <- klogistic(temp,k)
  }
  return(value)
}
```

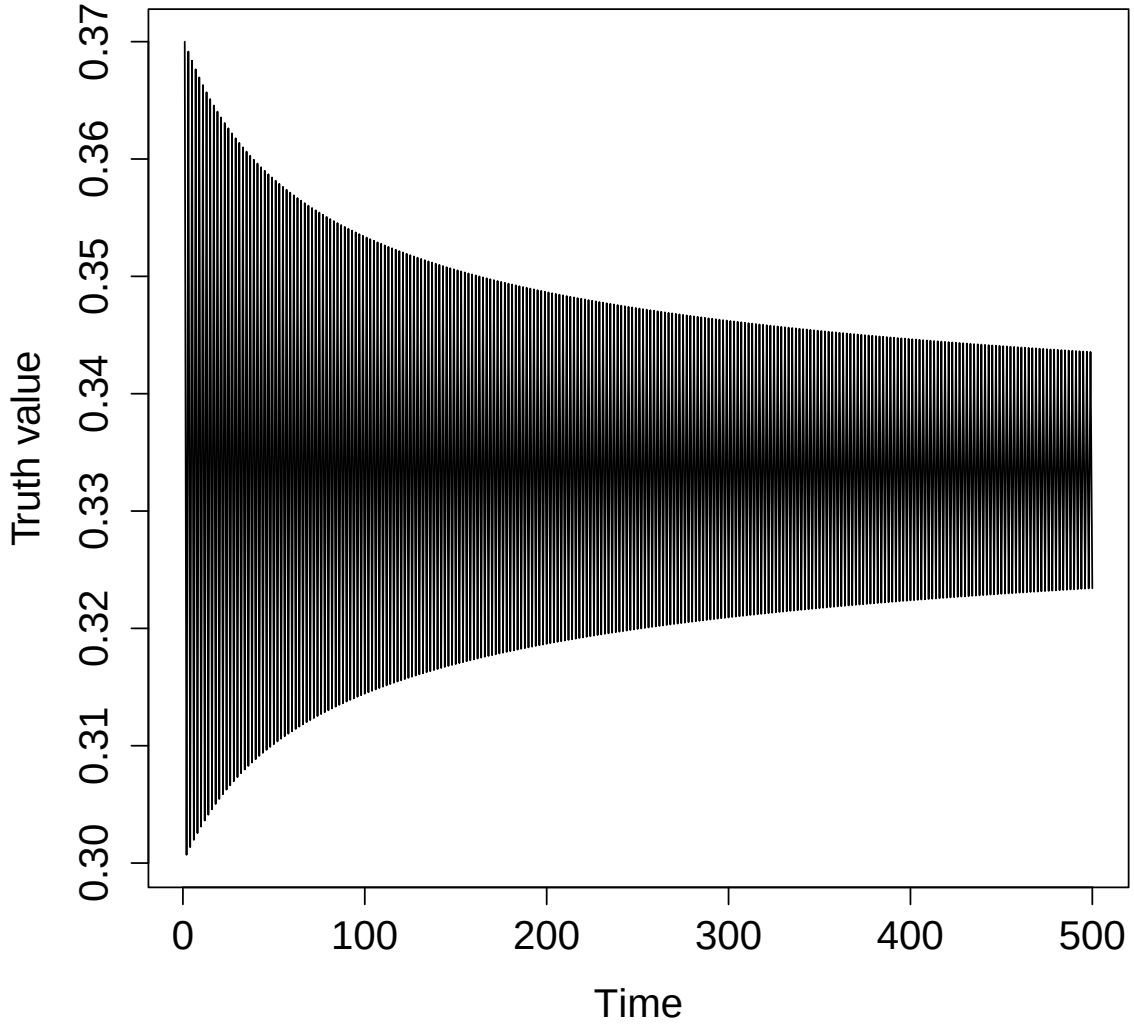
The logistic liar in R, $k = 2.5$

```
> klogistic2(.314,2.5,50)
 [1] 0.46149000000000006683 0.37870755025000002458 0.41177964591589066856
 [4] 0.39445707718681410903 0.40284827138987533424 0.39859614592983827475
 [7] 0.40070685405070660678 0.39964782208126914398 0.40017639903258150902
[10] 0.39991187827525609411 0.40004408027596793929 0.39997796471969282450
[13] 0.40001101885403755887 0.39999449087651905987 0.40000275463761658656
[16] 0.39999862270016173227 0.40000068865466154033 0.39999965567385475929
[19] 0.40000017216336902770 0.39999991391838951582 0.40000043040823705100
[22] 0.39999997847959278818 0.40000001076020474944 0.39999999461989799165
[25] 0.40000000269005109299 0.39999999865497448681 0.40000000067251273439
[28] 0.39999999966374366611 0.40000000016812820025 0.39999999991593593318
[31] 0.40000000004203195569 0.39999999997898394444 0.40000000001050806109
[34] 0.39999999999474600276 0.40000000000262703193 0.39999999999868640632
[37] 0.40000000000065683015 0.39999999999967161823 0.40000000000016411317
[40] 0.39999999999991786570 0.40000000000004110046 0.39999999999997937206
[43] 0.40000000000001023626 0.39999999999999491518 0.40000000000000246470
[46] 0.39999999999999868994 0.40000000000000068834 0.39999999999999957812
[49] 0.40000000000000024425 0.39999999999999980016
```

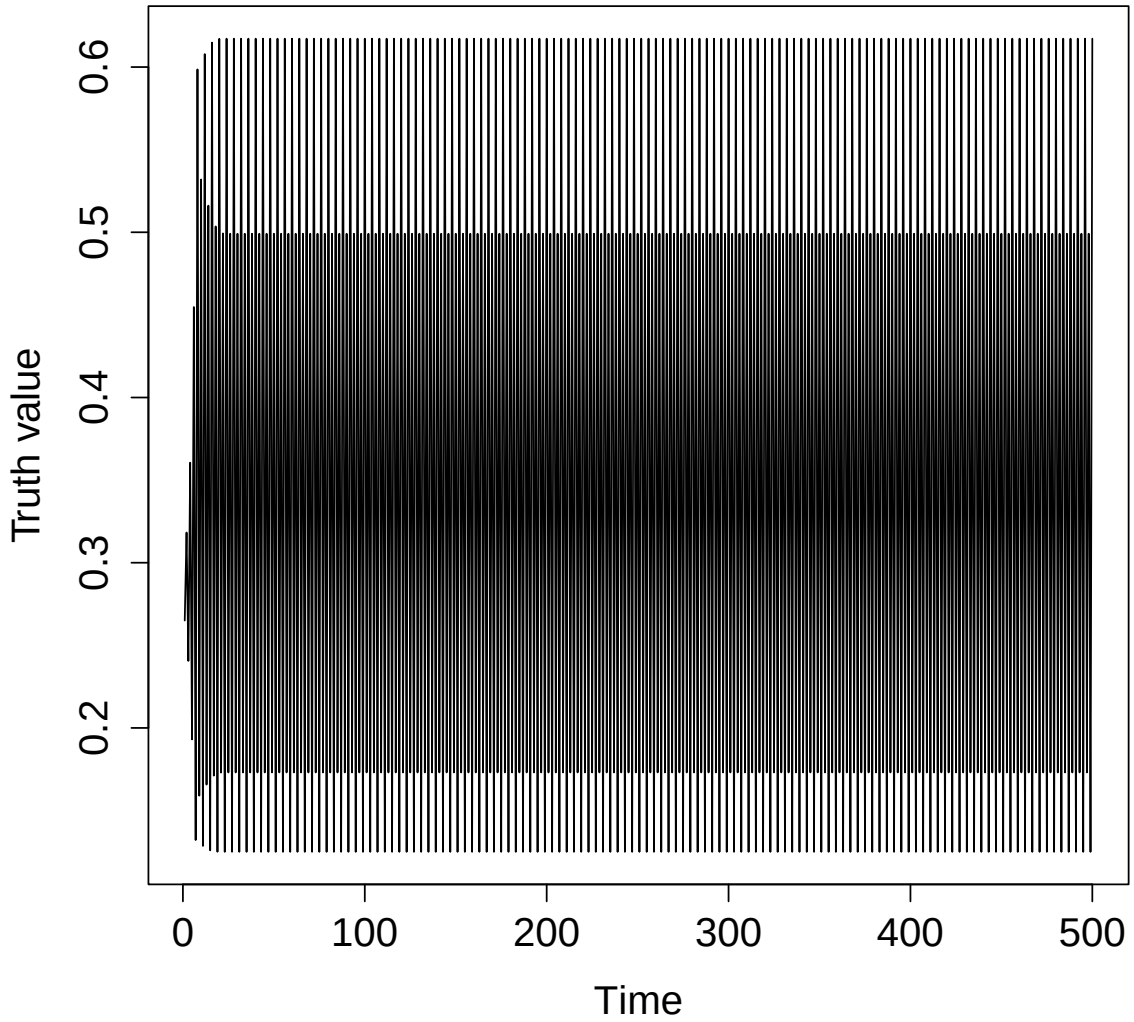
Time plot of the logistic liar, 5 time points



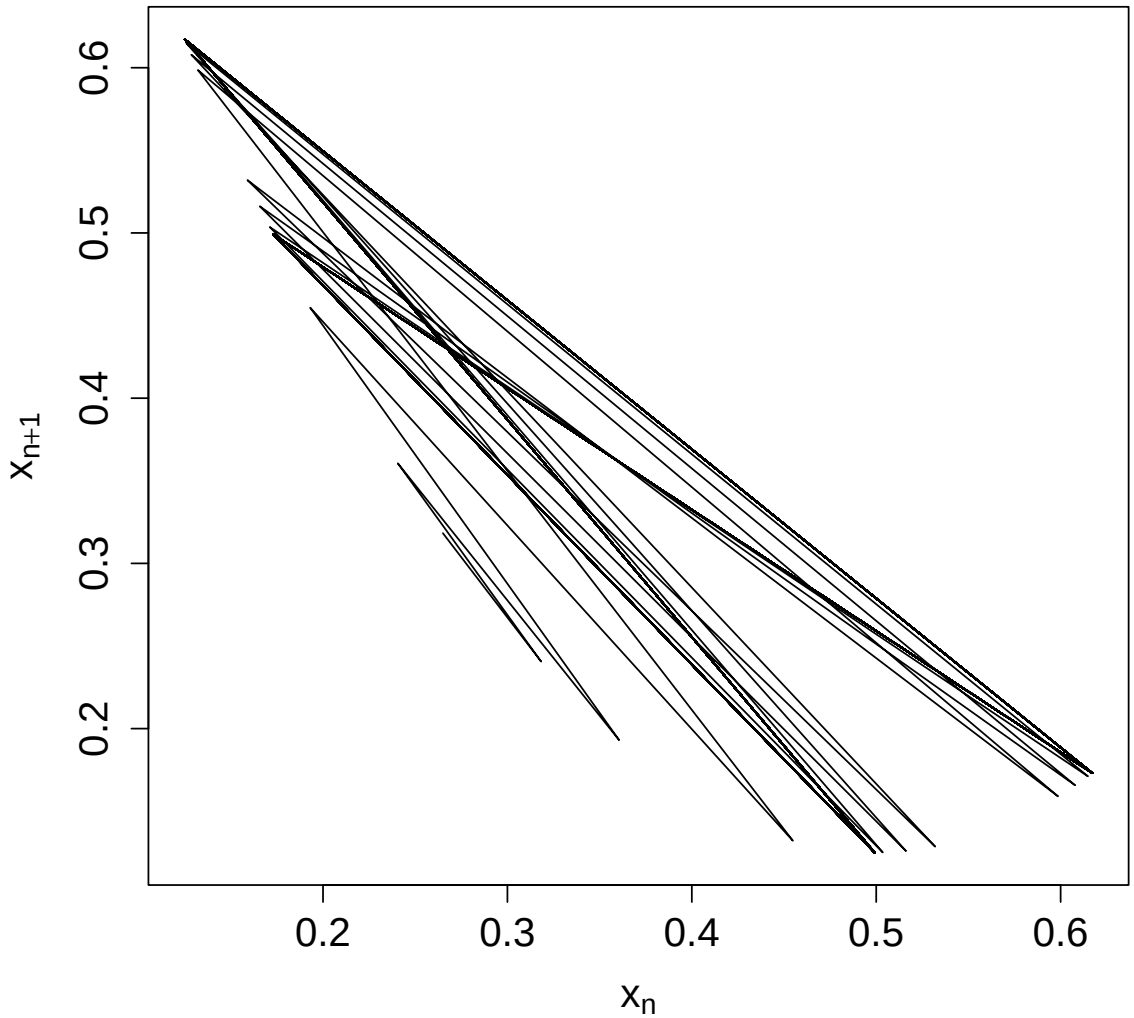
Time plot of the logistic liar, 500 time points, $k = 3$, $x_0 = 0.3$.



Time plot of the logistic liar, 500 time points, $k = 3.5$, $x_0 = 0.3$.



Plot of (x_n, x_{n+1}) for the logistic liar, 500 time points, $k = 3.5$, $x_0 = 0.3$.



Chaotic dualists

We can also model things when we have coupled sentences referring to each other. This was briefly mentioned early on. Here is another version

X: Y is true.

Y: X is false.

Here, if X is true, then Y is true, which means X is false, but this means that Y is false, which means X is true.

Here's another set designed for infinte-valued logic.

X: X is as true as Y.

Y: Y is as true as X is false

To interpret this using recursion, X claims to have the same truth value as Y. If this is true, then $\text{Abs}(y_n - x_n)$ is close to zero, and $\text{Abs}(y_n - x_n)$ is a measure of how false the statement is. (If it is 0, then the statement has 0 as the measure of falseness). The measure of the truth of the statement is therefore 1 minus the error, so

$$x_{n+1} = 1 - \text{Abs}(y_n - x_n)$$

Y claims to be true to the extent that X is false. So if Y is correct than $\text{Abs}((1 - x_n) - y_n) = 0$. Therefore

$$y_{n+1} = 1 - \text{Abs}((1 - x_n) - y_n)$$

Chaotic dualist

The above reasoning revised the truth values of X and Y simultaneously and gave

$$x_{n+1} = 1 - \text{Abs}(y_n - x_n)$$
$$y_{n+1} = 1 - \text{Abs}((1 - x_n) - y_n)$$

A variation on this is to update y_{n+1} based on the already updated value of x_{n+1}:

$$x_{n+1} = 1 - \text{Abs}(y_n - x_n)$$
$$y_{n+1} = 1 - \text{Abs}((1 - x_{n+1}) - y_n)$$

This second variation leads to more chaotic behavior.

For these to work, you need to give initial estimates for both sentences, (x_0, y_0). Here is some code. Here I actually did X: X is as true as Y, Y:Y is as true as X.

```
dualist <- function(x,y) {
  x_value <- 1-abs(x-y)
  y_value <- 1-abs(x_value-y)
  return(c(x_value,y_value))
}

dualist2 <- function(x,y,maxterms) {
  x_value <- rep(-1,maxterms)
  y_value <- rep(-1,maxterms)
  x_temp <- x
  y_temp <- y
  for(i in 1:maxterms) {
    x_value[i] <- dualist(x_temp,y_temp)[1]
    y_value[i] <- dualist(x_temp,y_temp)[2]
    x_temp <- x_value[i]
    y_temp <- y_value[i]
  }
  value <- cbind(x_value,y_value)
  return(value)
```

Chaotic dualist

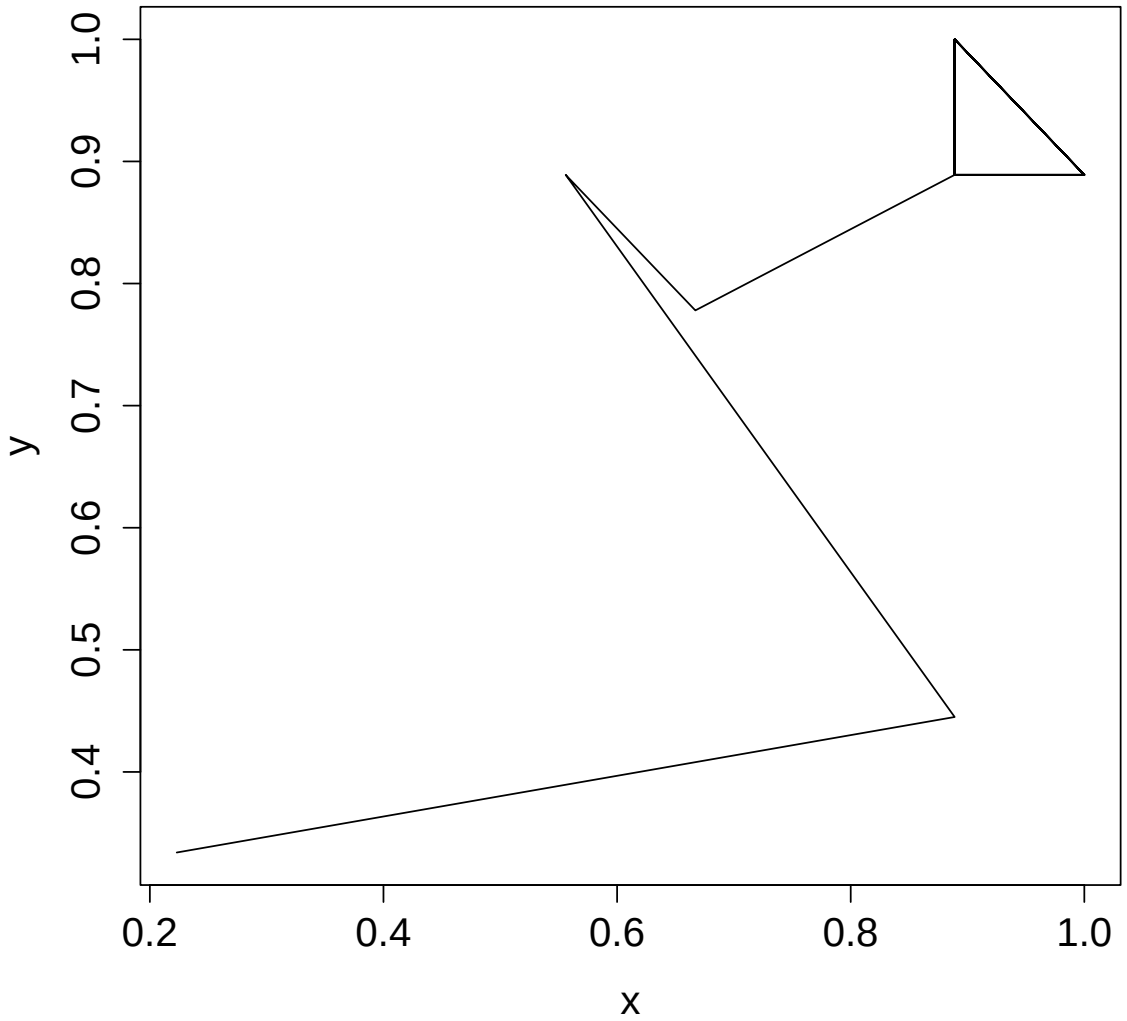
```
 > dualist(.41,.72)
[1] 0.69 0.97
> dualist(.69,.97)
[1] 0.72 0.75
> dualist(.72,.75)
[1] 0.97 0.78
> dualist(.97,.78)
[1] 0.81 0.97
> dualist(.81,.97)
[1] 0.84 0.87
```

Chaotic dualist

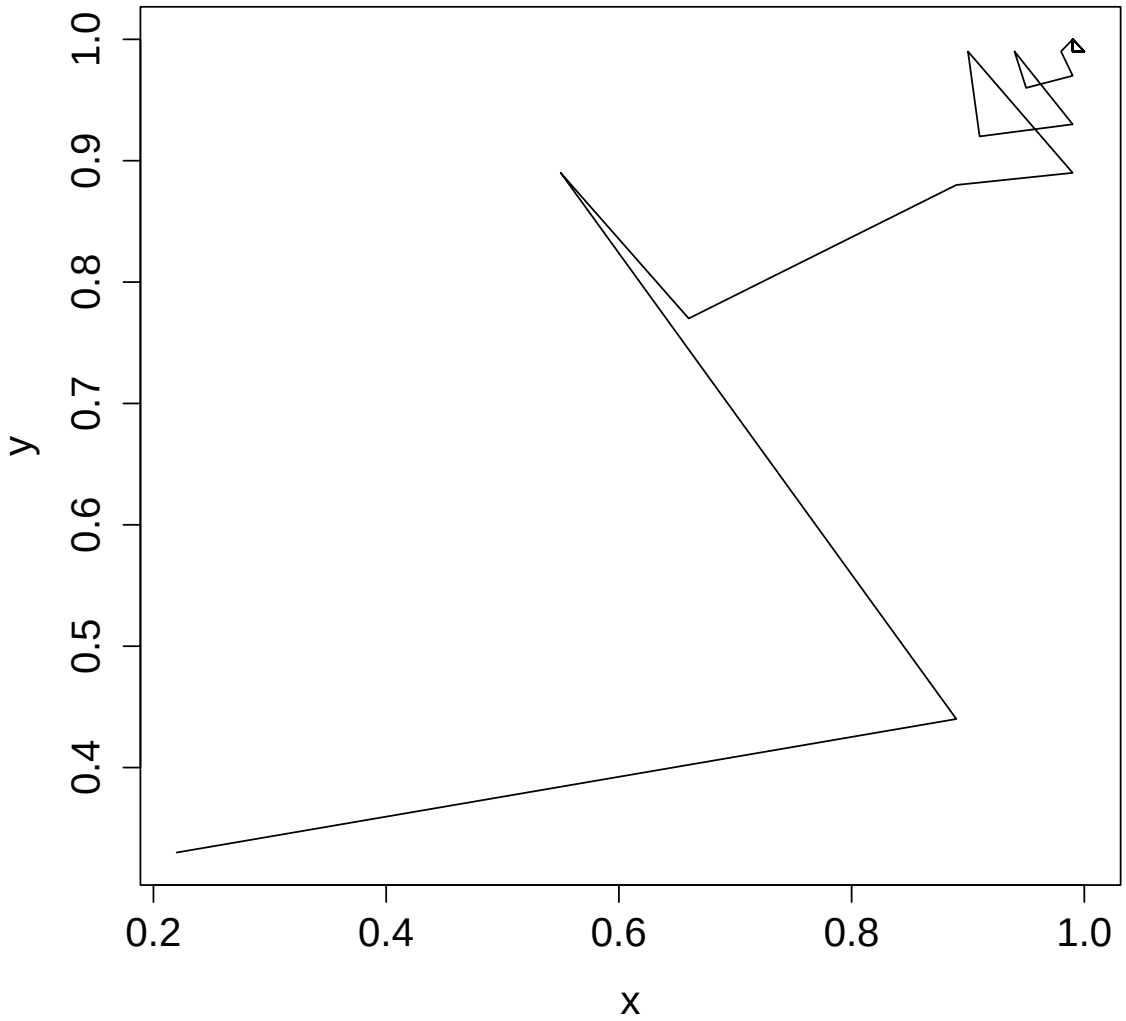
Here $(x_0, y_0) = (0.112, .0.889)$. It trapped in a cycle of length 3.

```
> values
      x_value y_value
 [1,]   0.223   0.334
 [2,]   0.889   0.445
 [3,]   0.556   0.889
 [4,]   0.667   0.778
 [5,]   0.889   0.889
 [6,]   1.000   0.889
 [7,]   0.889   1.000
 [8,]   0.889   0.889
 [9,]   1.000   0.889
[10,]   0.889   1.000
[11,]   0.889   0.889
```

Plot with $(x_0, y_0) = (0.112, 0.889)$ for the dualist.



Plot with $(x_0, y_0) = (0.11, 0.89)$ for the dualist.

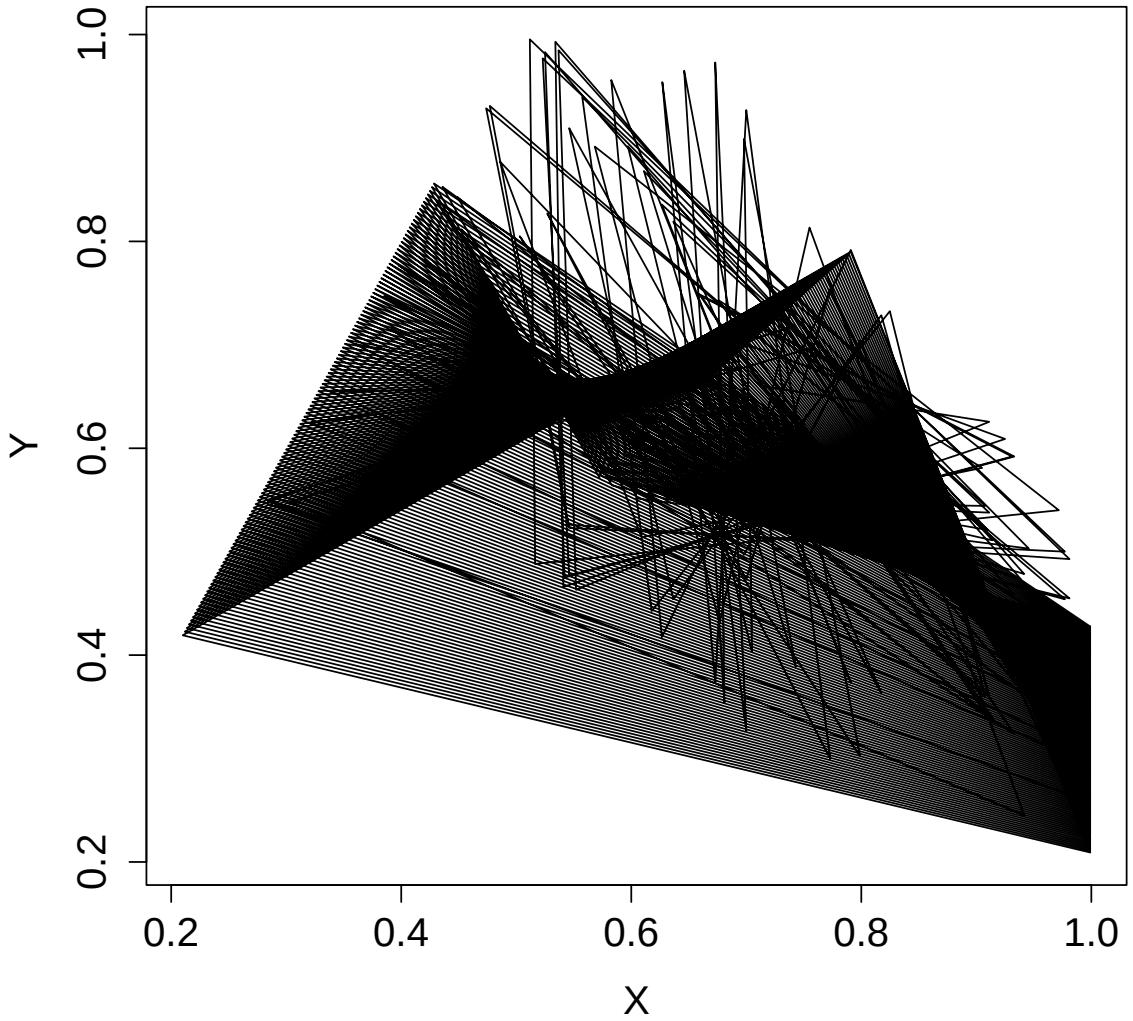


For these to work, you need to give initial estimates for both sentences, (x_0, y_0). Here is some code. Here I did X: X is as true as Y, Y:Y is as true as X is false.

```
dualistb <- function(x,y) {
  x_value <- 1-abs(x-y)
  y_value <- 1-abs(1-x_value-y)
  return(c(x_value,y_value))
}

dualistb2 <- function(x,y,maxterms) {
  x_value <- rep(-1,maxterms)
  y_value <- rep(-1,maxterms)
  x_temp <- x
  y_temp <- y
  for(i in 1:maxterms) {
    x_value[i] <- dualistb(x_temp,y_temp)[1]
    y_value[i] <- dualistb(x_temp,y_temp)[2]
    x_temp <- x_value[i]
    y_temp <- y_value[i]
  }
  value <- cbind(x_value,y_value)
  return(value)
  }
```

Chaotic dualist with $(x_0, y_0) = (.412, .763)$ and 500 iterations.



Dualist variant of Half-Sayer

X: X is true to half the extent that Y is true.

Y:Y is as true as X is false.

We'll translate these as

$$x_{n+1} = 1 - \text{Abs}(.5 * y_n - x_n) y_{n+1} \qquad = 1 - \text{Abs}((1 - x_{n+1}) - y_n)$$

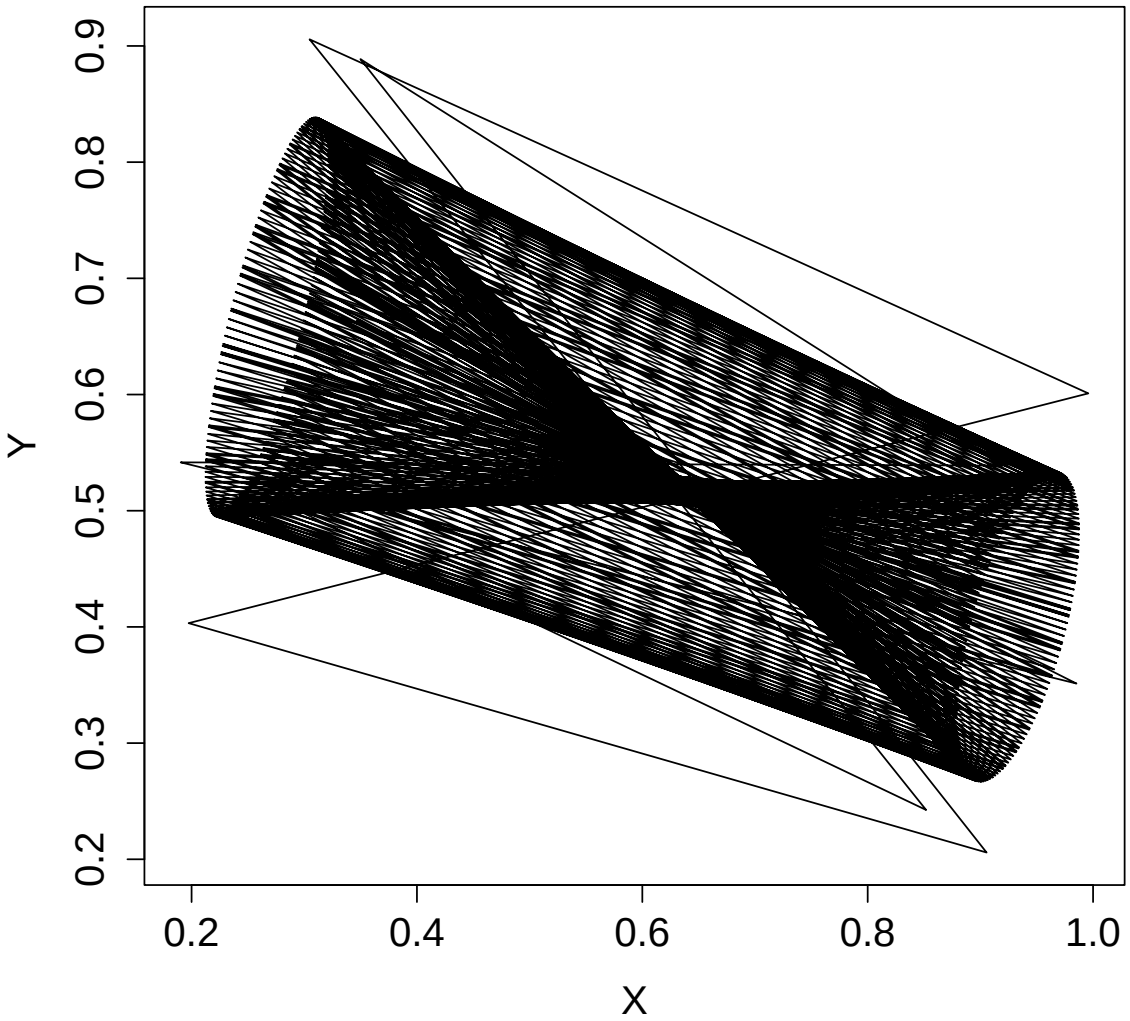
```
dualistc <- function(x,y) {
  x_value <- 1-abs(x-y/2)
  y_value <- 1-abs(1-x_value-y)
  return(c(x_value,y_value))
}

dualistc2 <- function(x,y,maxterms) {
  x_value <- rep(-1,maxterms)
  y_value <- rep(-1,maxterms)
  x_temp <- x
  y_temp <- y
  for(i in 1:maxterms) {
    x_value[i] <- dualistc(x_temp,y_temp)[1]
    y_value[i] <- dualistc(x_temp,y_temp)[2]
    x_temp <- x_value[i]
    y_temp <- y_value[i]
  }
  value <- cbind(x_value,y_value)
  return(value)
  }
```

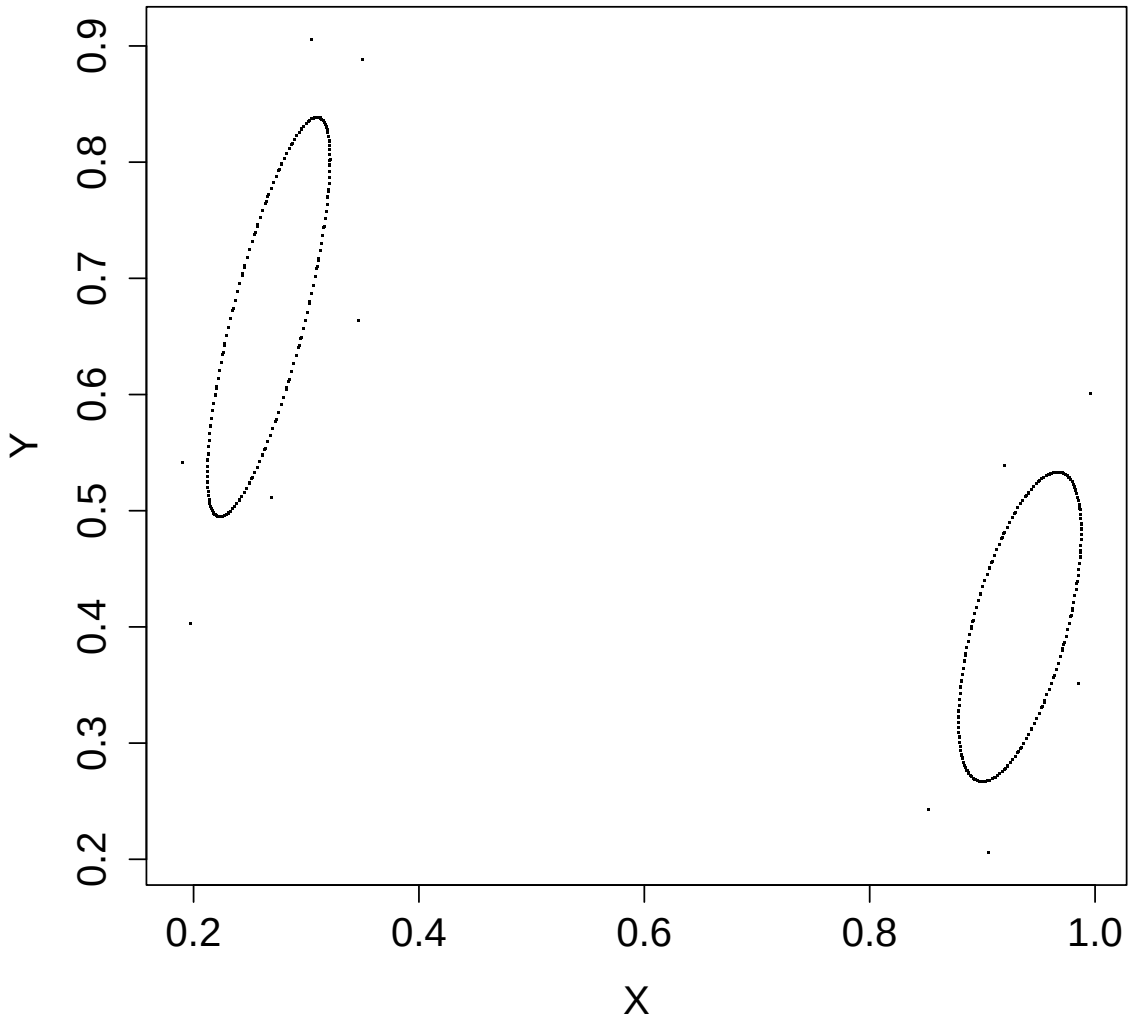
Here I plot for $(x_0, y_0) = (0.812, 0.317)$. You get very different looking plots depending on whether you connect successive points or not. Here is the R code.

```
values <- dualistc2(.812,.317,400)
plot(values[,1],values[,2],type="l",ylab="Y",
   xlab="X",cex.lab=1.4,cex.axis=1.4)
plot(values[,1],values[,2],pch=".",ylab="Y",
   xlab="X",cex.lab=1.4,cex.axis=1.4)
```

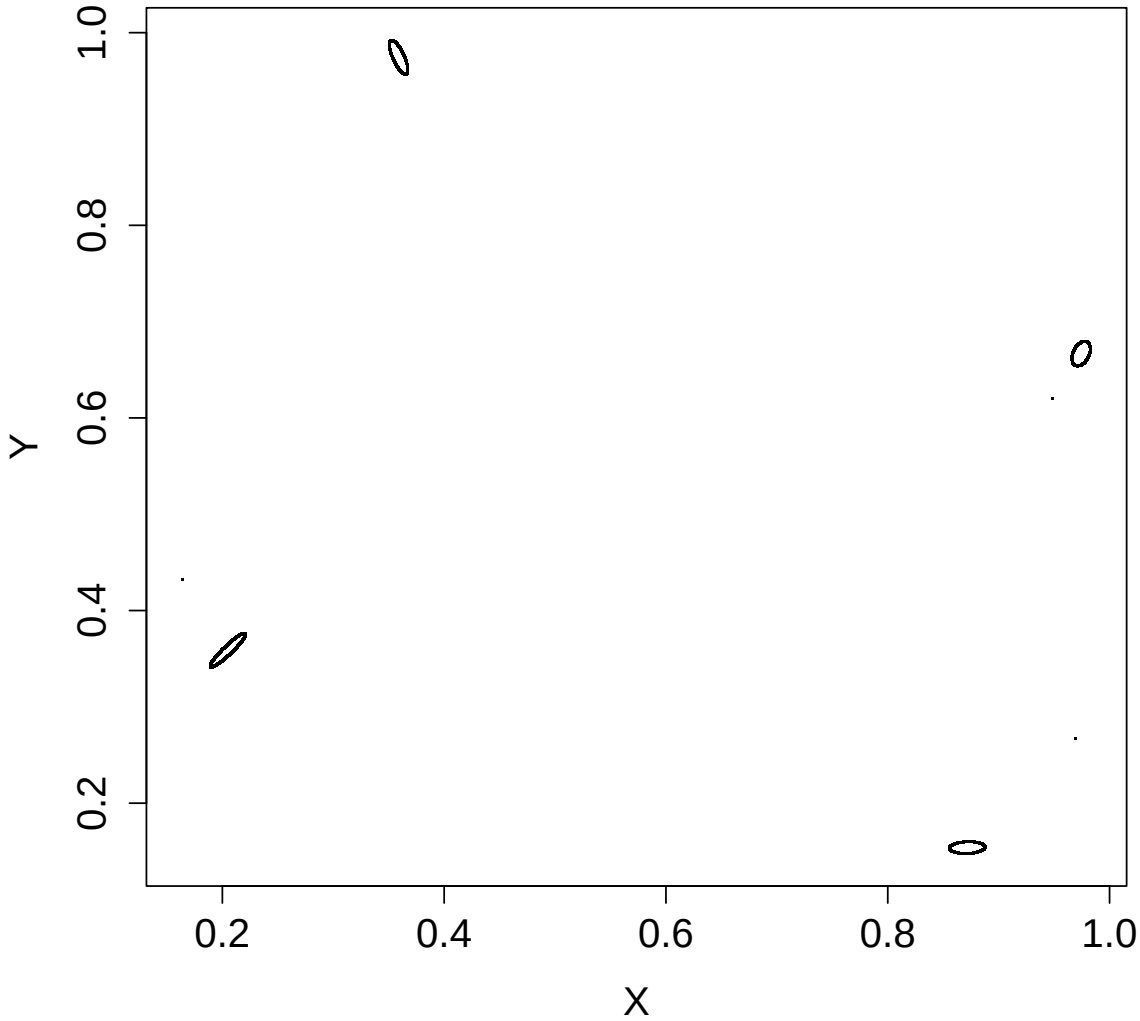
Half-Sayer dualist with $(x_0, y_0) = (.813, .317)$ and 400 iterations.



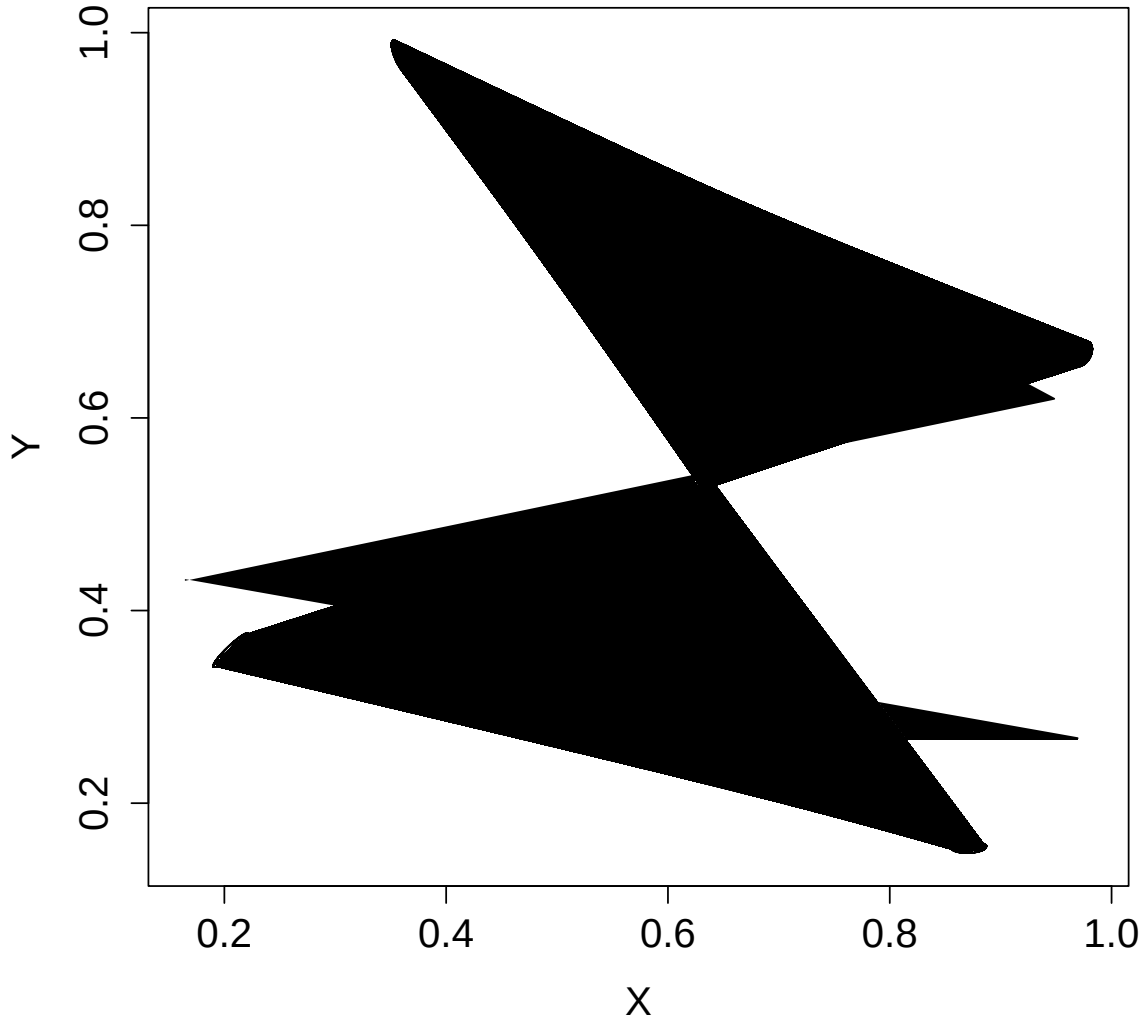
Half-Sayer dualist with $(x_0, y_0) = (.813, .317)$ and 400 iterations.



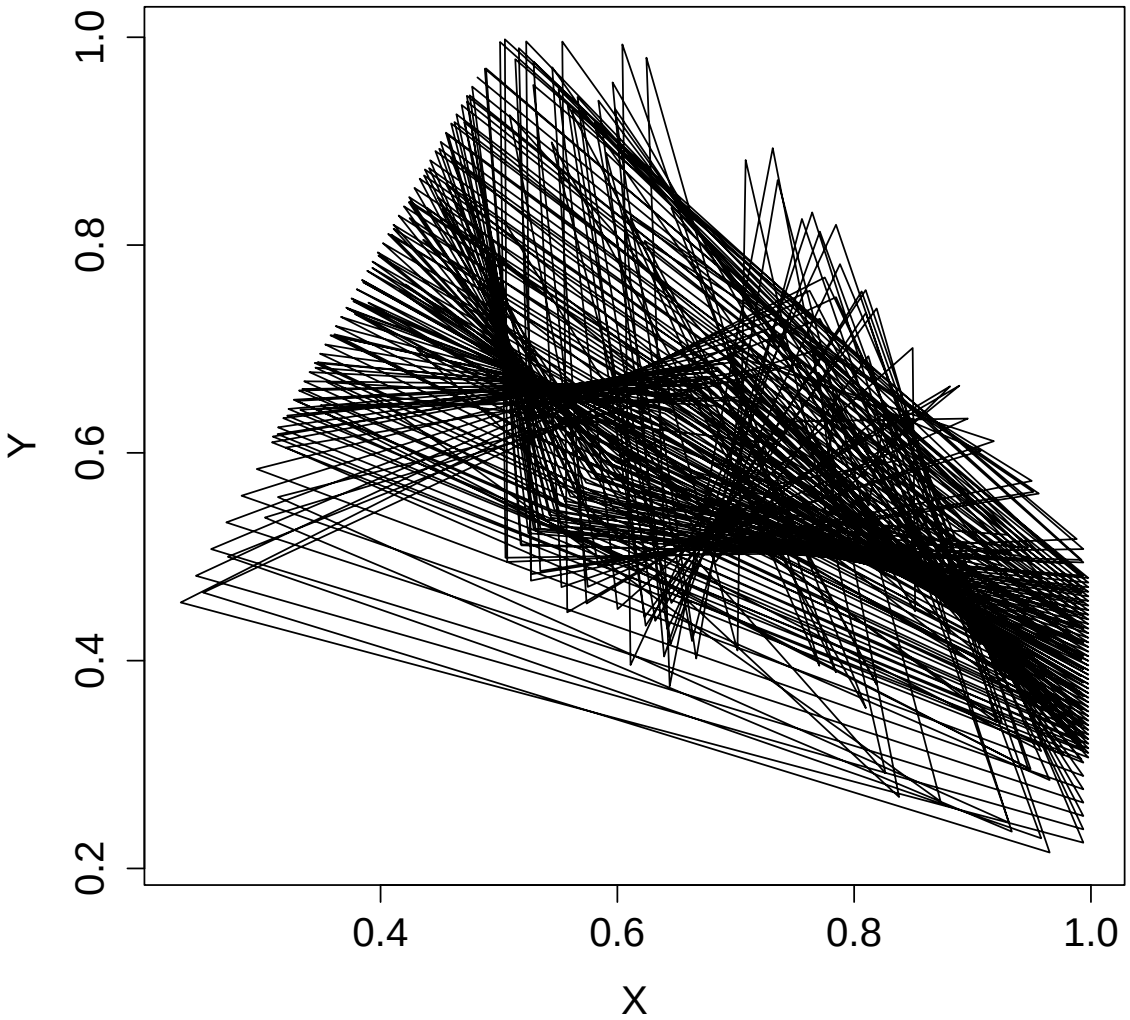
Dualistb with $(x_0, y_0) = (.412, .763)$ and 400 iterations.



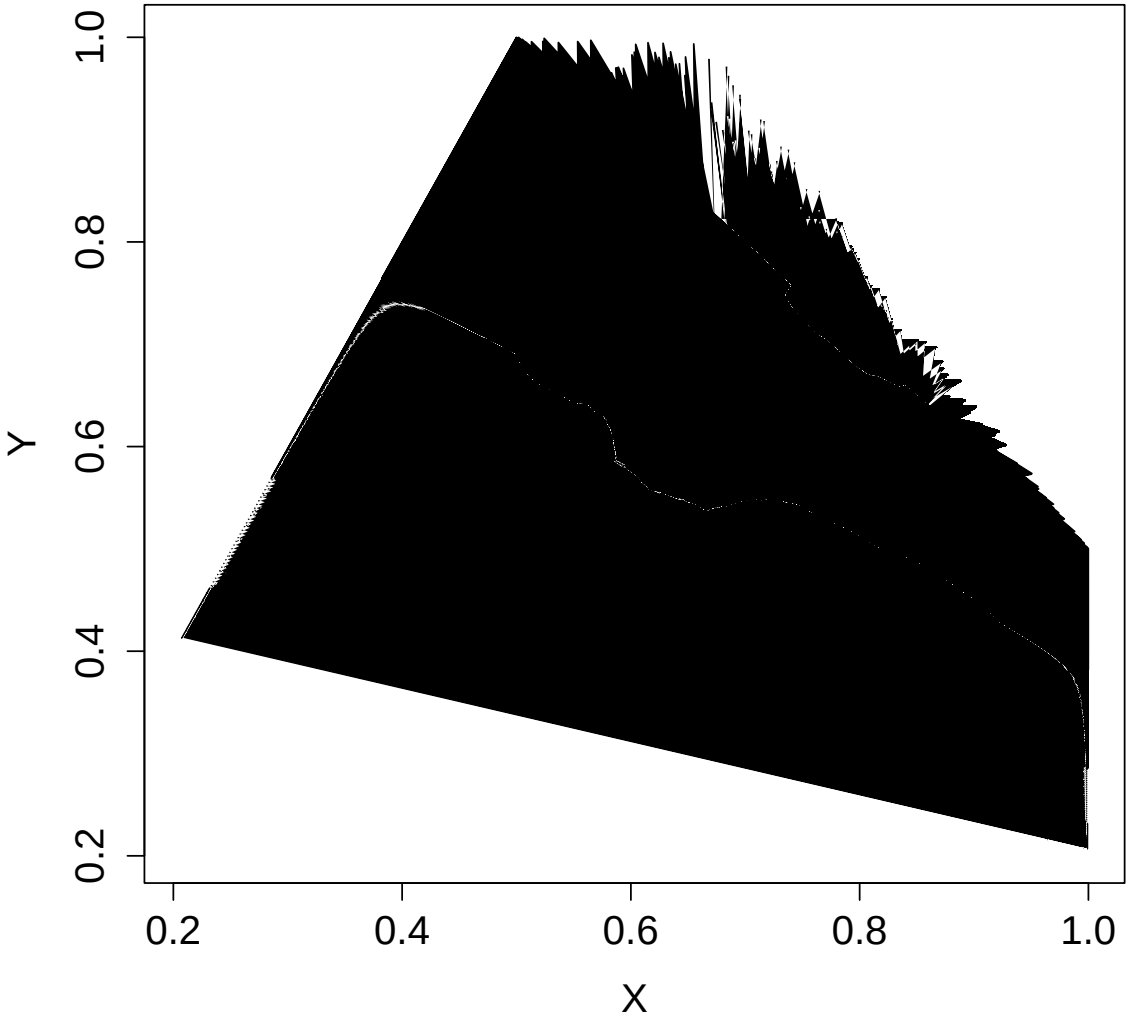
Dualistb dualist with $(x_0, y_0) = (.412, .763)$ and 400 iterations.



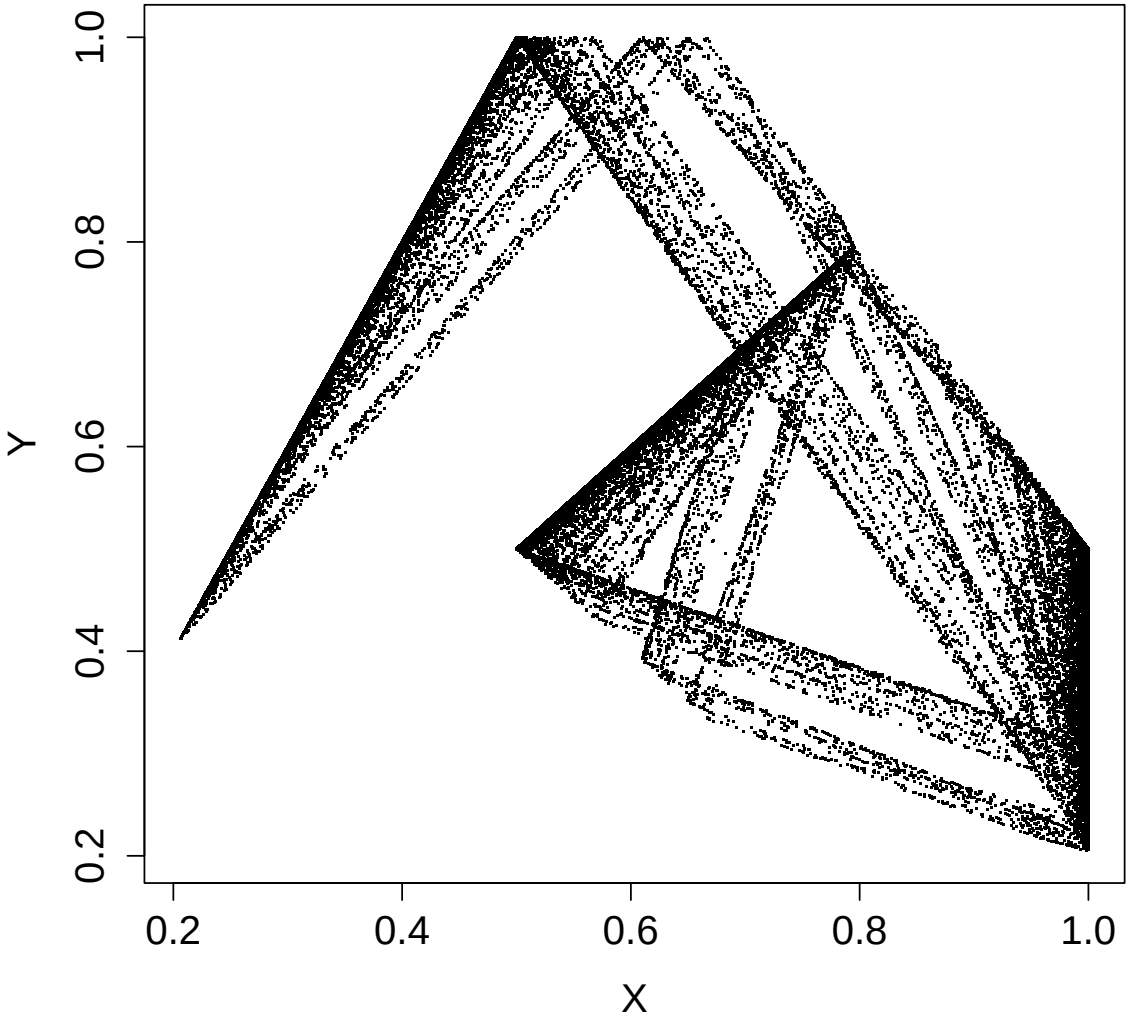
Dualistb dualist with $(x_0, y_0) = (.412, .763)$ and 400 iterations.



Dualistb dualist with $(x_0, y_0) = (.812, .317)$ and 40000 iterations.



Dualistb dualist with $(x_0, y_0) = (.812, .317)$ and 40000 iterations.



The Prisoner's Dilemma

The Prisoner's dilemma was developed and studied initially around 1950 by Merrill Flood and Melvin Drescher at the Rand Corporation. Here is one version of the story:

Two people, A and B, are arrested for a crime (say, committing burglary together). When arrested, the two prisoners are separated and can't discuss anything with each other. The district attourney tells player A:

The evidence is pretty weak. If you both stonewall and refuse to give information, you'll get 2 years of prison each. If you don't give any information but player B confesses, then we'll let him off, and you will get 5 years. But if you confess and B stonewalls, then B will get 5 years, and you go free. If you both confess, you'll get four years each.

We assume that the DA is telling the truth, and that the prisoner's believe the DA is telling the truth. Those really are the payoffs. The question is, what is it rational for the players to do?

For player A, if player B cooperates (stonewalls), then either player A stonewalls, which case player A gets two years, or player B confesses, in which case player A goes free, and B gets five years. So if player B cooperates, then player A is better off confessing (called defecting.

On the other hand, if player B defects (confesses), then if player A confesses, then player A gets four years. If player A doesn't confess (cooperates), then player A gets five years.

Therefore, no matter what player B does, player A is better off (less jail time) confessing. Player B is in the same situation, so it is rational for player B to confess. If both confess, then they get four years each, considerably worse, than if they both cooperate, getting four years each.

This can be summarized in a matrix depicting how many years each gets

	Player B	
Player A	Cooperate	Defect
Cooperate	$(2, 2)$	$(5, 0)$
Defect	$(0, 5)$	$(4, 4)$

Here (a, b) means that prisoner A gets a years, and Prisoner B gets b years. Another interesting feature is that we can think about the total amount of prison time for the two players. The total amount is minimized by the two prisoners cooperating. However, Player A is not concerned with minimizing the total, but rather just the amount for Player A. For both columns (the possible decisions of player B), a is lowe if A defects. Simillar, for both rows (the possible decisions of player B), b is lower if B defects.

The main point is that the payoff matrix is structured so that both players would be better off if they cooperated, but individually, each player is better off defecting.

Here's another example where this might come up. Imagine two countries that are adversaries. Both countries could have more money for health care, education, etc. if they don't have a military. Both agreeing to not have a military would be cooperation, and would have large economic advantages. If Both countries have a military, neither will be willing to engage in war because it would be too costly and uncertain who would win. However if Country A has a military an country B doesn't, then Country A can mistreat country B. If country B has a military and country A doesn't, then country B can mistreat country A. Mistreating could take the form of invasion, unfair trade deals, tariffs, etc.

Suppose you go to a restaurant. As a customer you might either leave a small tip (defect) or a big tip (cooperate). Suppose you have already paid, including the tip, but are still talking with your family after the meal. Suppose the waiter doesn't know what tip you've left. The waiter can either refill your drink (cooperate), or ignore your table since you have already paid (defect).

In this case, cooperation just means effort in the form of paying more money or giving your time by offering a service, and the amount of effort isn't equal for the two parties. But it still has a similar form as a prisonner's dilemma. The waiter has nothing to gain (other than goood will) by refilling your glass after you've paid, so is better off defecting, regardless of what tip you gave. And you are financially better off giving a small tip. So ignoring psychological factors, like feeling good about leaving a generous tip or providing good service, both parties are individually better off defecting.

Environmental policy is another one. If all countries work to lower greenhouse emissions, that comes at a cost, because the infrastructure might be expensive, but by cooperating, greenhouse gases could be lowered. If all countries comply but one, but one country does nothing to lower greenhouse emissions, then that country is getting the benefit of all other countries complying without the cost of more expensive infrastructure. If each country reasons that it is better off not paying for the infrastructure because they can just benefit from other countries doing it, you end up with no countries investing in the infrastructure. A country that doesn't participate when other countries do might be considered a "Free Rider" /

An exmaple from biology is vervet monkeys who look out for predators and scream to alert the group of a potential predator such as leopard, eagle, or snake. By screaming, the monkey is potentially putting itslef in danger. This is a form of cooperation since it comes at some cost, but if every vervet monkey in the group does this, then the group is safer overall. If a vervet monkey takes a turn as a lookout but stays silent when spotting a predator, that vervet monkey is defecting, maybe because by being quiet, the predator will not notice the vervet monkey. But this is a defection to the group.

The Prisoner's dilemma gets more interesting when it is repeated. In the restaurant example, if you go to a restaurant repeatedly and have the same waiter, then you might express appreciation for being waited on after paying by giving a larger tip next time. Thus, even if you (the customer) defected in round 1 but the waiter copperated, it might be more likely that both will cooperate in round 2. Similarly, if the waiter noticed a big tip in round 1 (say after you have left), then the waiter might be more willing to service your table after you pay in the future, causing increased cooperation on both sides.

Another example of a repeated encounter like this is with tutoring. Let's say you tutor someone in math for money. This is a repeated encounter and is only likely to work if you are getting paid regularly and providing service regularly. However, if it is the tutor's last time to see you, there might be a temptation to cancel the appointment and not pay, which is a kind of defection.

In iterated prisoner's dilemmas, we can imagine there is memory of what has happened in the past (has this person cooperated or defected before?), and that your decision to cooperate or defect might depend on this past history.
An example of a real-life repeated prisoner's dilemma type of situation is a tenant paying rent for an apartment, and a landlord maintaining service (paying for utilities if they are included, fixing appliances or plumbing). For the tenant, cooperation is paying rent and defection is not paying rent. For the landlord, cooperation is making needed repairs and maintaining including utlilities, while defection would be ignoring the tenant's needs and requests. There is a temptation for the tenant to not pay the last month's rent, because the dilemma will no longer be repeated, and there might not be much penalty for doing so.

There are different possible strategies in repeated prisoner's dilemmas, and these have been studied using computer simulations extensively. David Axelrod made a computerized tournament in 1980 and wrote a book about it called *The Evolution of Cooperation*.

In the tournament, users submittted different strategies to compete against each other. The idea is that depending on the interaction, each strategy (computer program) woulld get different amounts of points, depending on the combination of cooperation and defection for that encounter, and each program had many encounters (including repeated encounters) with different programs.

The simplest strategies only remember what happened in the last interaction with another strategy, not remembering a longer history.. Each strategy can recorded as a triple (i, c, d) where i is the initial move (0=defect, 1=coperate), c is what the strategy will do if the last encounter showed the other agent cooperating. And d is what the strategy will if the last encounter showed the other agent defecting. There are eight possible strategies of this type.

The eight reactive strategies, plus a random one where you cooperate with probability p.

i	c	d	reactive straategy
0	0	0	Always defect (AD)
0	0	1	Suspicious Doormat
0	1	0	Suspicious Tit For Tat
0	1	1	Suspicous Quaker
1	0	0	Deceptive Defector
1	0	1	Gullible Doormat
1	1	0	Tit for Tat (TFT)
1	1	1	Quaker (always cooperate)
p	p	p	Random

Results

First I'll mention that there are other strategies if you allow longer memories. For example one strategy called Grim, but could be called Unforgiving, initially cooperates but as soon as the opponent defects, it defects with that opponent forever, even if the opponent starts cooperating again. Once a defector, always a defector, in the eyes of Grim.

In the tournament TFT won overall, even though it often doesn't beat individual other strategies. But in an environment of different strategies, it did the best on average. In fact, TFT doesn't do better than any individual strategy if TFT only plays against one opponent.

TFT is also similar to this version of the Golden Rule: Do unto others as they do unto you.

After Axelrod's results, researchers Nowak and Sigmund investigated more probabilistic strategies. Part of the motivation is thinking that you might be mistaken about whether the other person defected or cooperated.

A very successful strategy in their work is called Generous Tit For Tat, with strategy $(1, 1, 1/3)$. The idea is that it cooperates as long as previous encounters involved cooperation. If a previous encounter was a defect, then it defects with probability $2/3$ and copperates with probability $2/3$.

They also studied cases where 0 and 1 were not allowed, so all parameters had to be strictly between 0 and 1. The strategy of always defecting would be replaced with something like $(0.01, 0.01, 0.01)$, a strategy that cooperates with probability 0.01 regardless of the encounter.

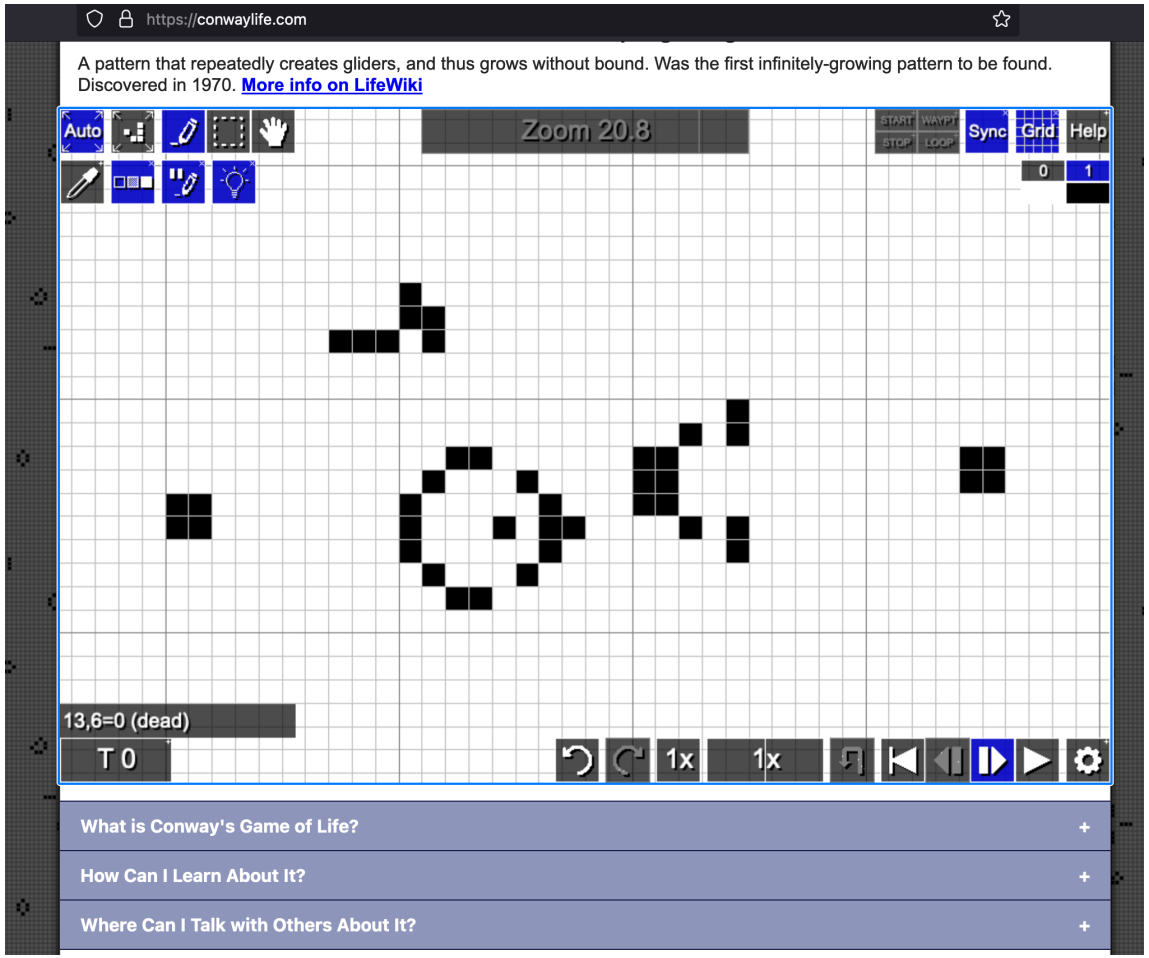
In one study, 121 strategies were used with (c, d) values distributed between .01 and 0.99. In this simulation, the more "selfish" strategy of nearly always defecting did better initially, but eventually strategies similar to GTFT dominated.

Spatialization of the Prisoner's Dilemma

The idea of a spatialized version of the prisoner's dilemma is that strategies are given locations on a grid, and strategies are more likely to interact with closer neighbors. In this case there might be clusters of different strategies.

An area of research dealing with agents interacting in a 2D world is cellular automata. The most famous example was developed by mathematician John Conway and is called The Game of Life.

There are online versions that you can play. Check out https://conwaylife.com/


https://conwaylife.com
A pattern that repeatedly creates gliders, and thus grows without bound. Was the first infinitely-growing pattern to be found.
Discovered in 1970. More info on LifeWiki
Auto
Zoom 20.8
START
WAYPT
STOP
LOOP
Sync
Grid
Help
0
1
13,6=0 (dead)
T 0
1x
1x
What is Conway's Game of Life?
How Can I Learn About It?
Where Can I Talk with Others About It?

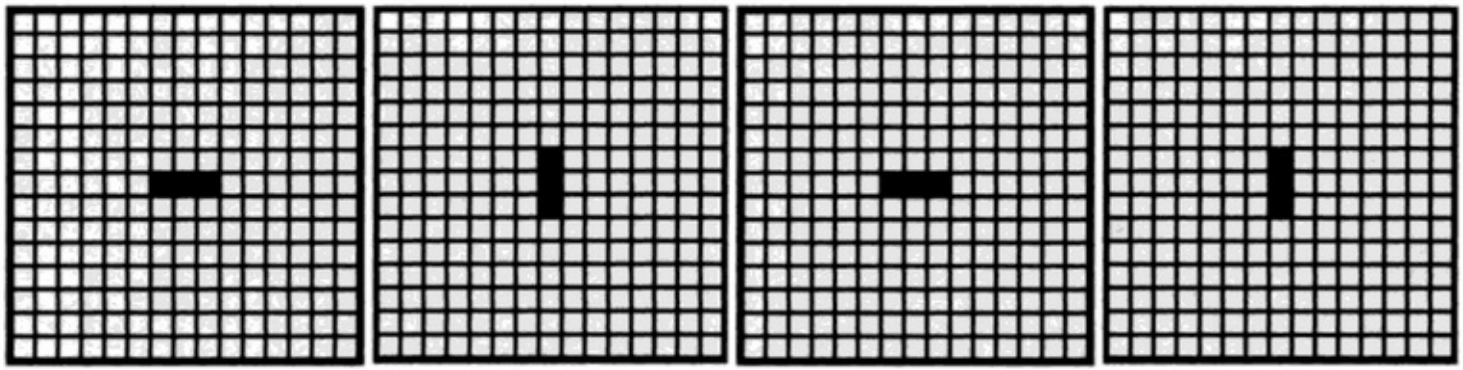
In the Game of Life, each cell is either on or off. Time is measured in discrete time points. Each cell gets updated in the next round depending on two rules:

Birth rule: A cell dead at generation t will come alive at generation $t + 1$ if exactly three neighbors are alive at t

Survival rule: A cell that is alive at generation t will remain alive at $t + 1$ if it is surrounded at t by precisely two or three live live neighbors.

To interpret the rules, a cell is born if it has three parents (neighbors). It can die from loneliness (0 or 1 neighbors) or overcrowding (4 neighbors).

The Blinker



The *r*-Pentomino.

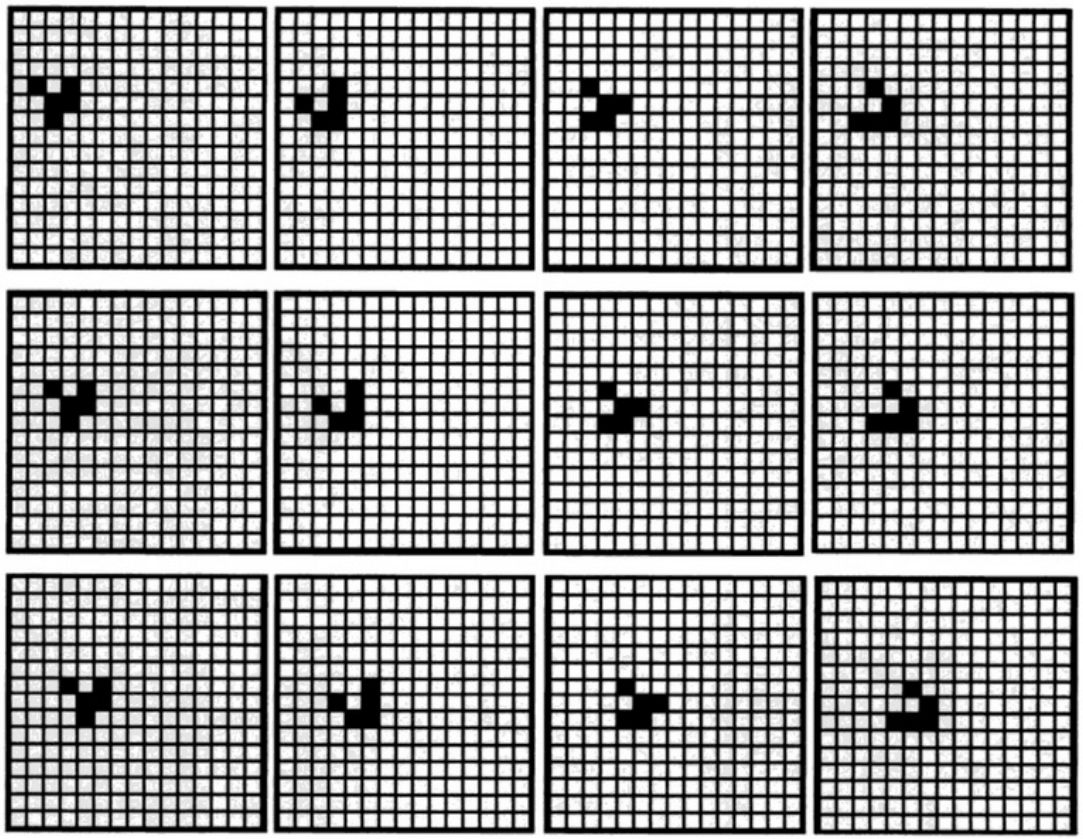


Figure 4 Movement of the Glider in Conway's Game of Life.

The Glider.

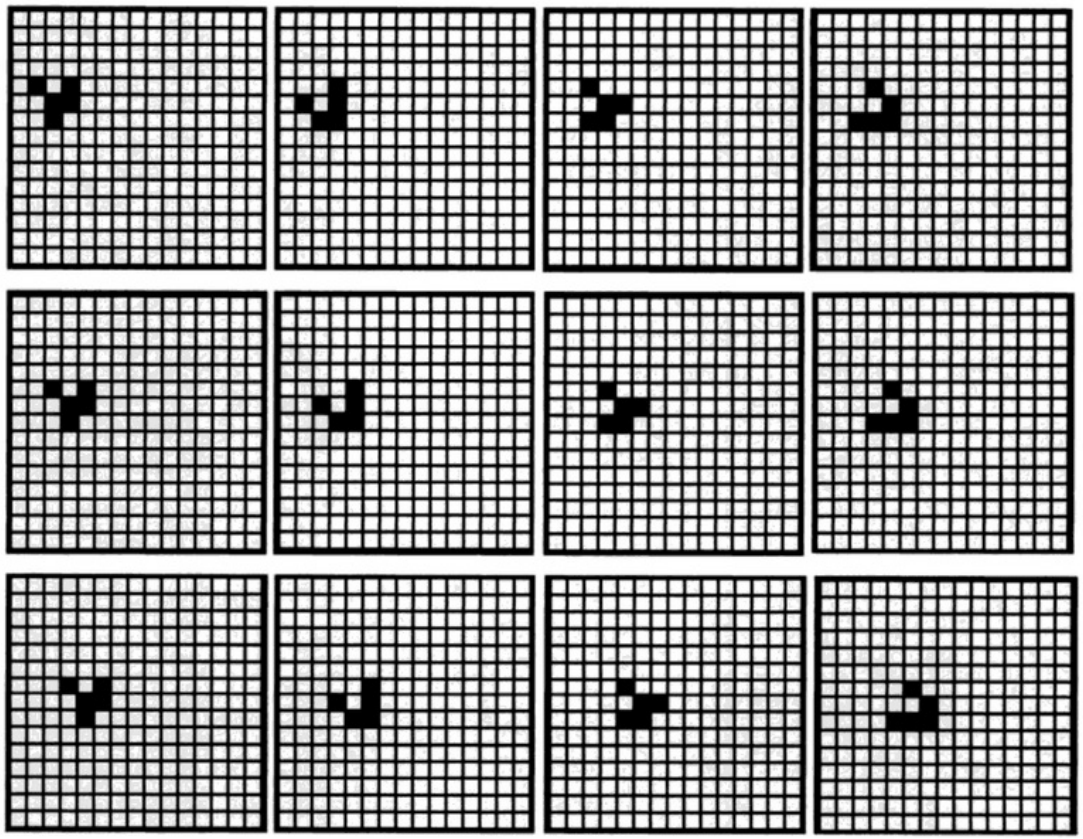
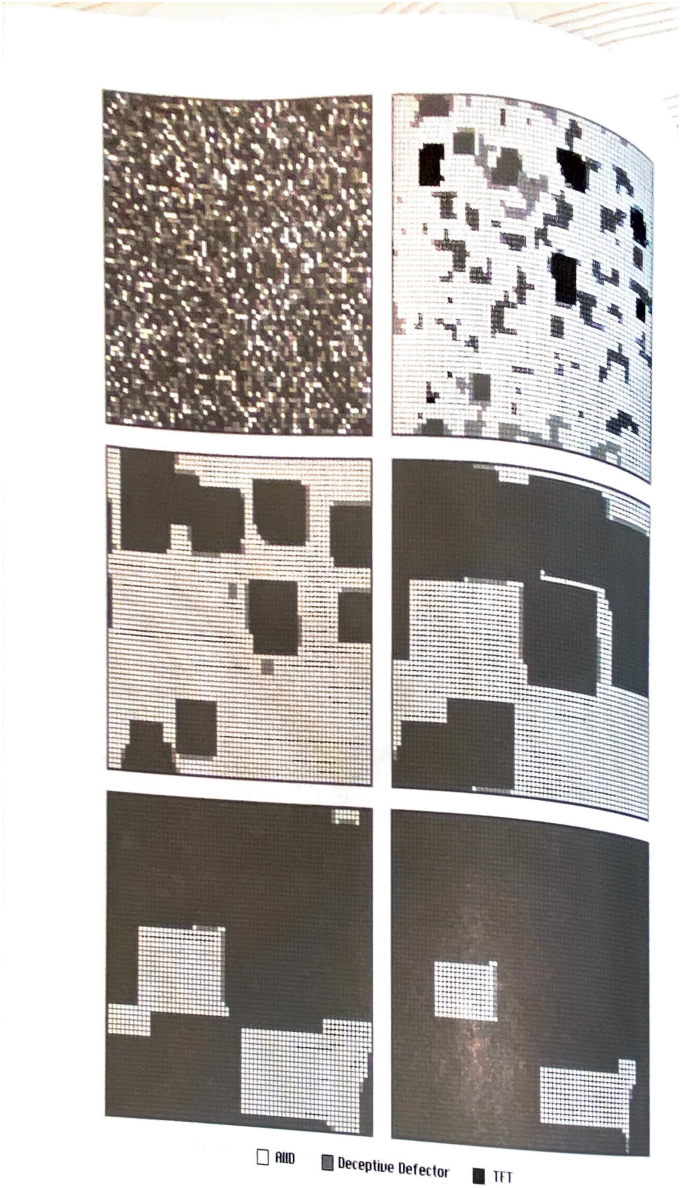


Figure 4 Movement of the Glider in Conway's Game of Life.

Cellular Automata version of Prisoner's dilemma

Start with a 64×64 array. Each cell is given one of 8 Prisoner's dilemma strategies. Each cell plays against each of it's neighbors 200 times and gets a score. At each state, if a neighbor has a higher score, a cell converts to the neighbor's strategy.

Spatial Prisoner's Dilemma.



176 Chapter 4