```
using Plots, Symbolics,Latexify, DataFrames
```

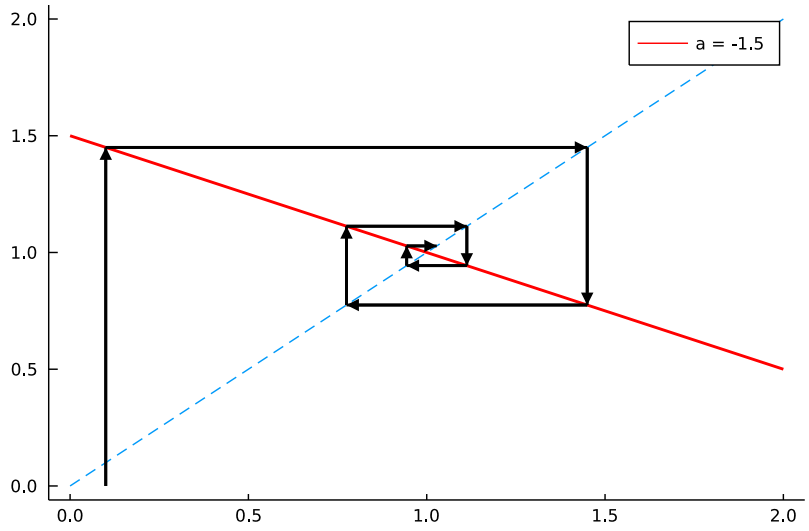
Linear dynamics

Sketch the trajectory for $x' = x + a(x-1)$ for $a = -1.5, a = -2.0$, and $a = -2.5$. This should be fairly simple. We can whip up a function to plot trajectories of the given function and then we'll just plug in the values for a. One way to do this is to use quiver plots that follow the trajectory of x at each time step. This can be achieved by calculating the values of x for each t, given some initial x value. We can store the coordinates (x, x') for each time step in a pair of vectors. This gives us the starting point for each quiver/arrow. To get the direction and length of the arrow shafts we simply subtract the coordinates (x, x') for each t from those of the corresponding $t + 1$.

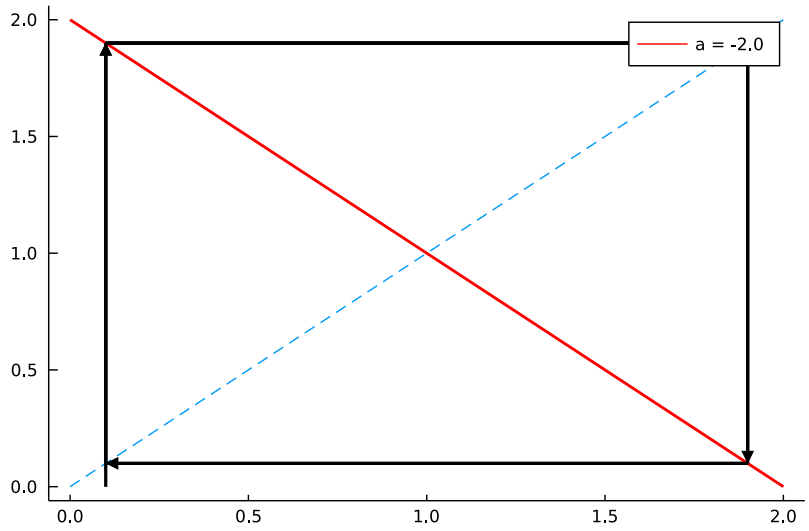
```
x′(x,a) = x + a*(x-1)
function spider(a, n = 10,x_0 = 0.1 )
N = 2*n
    # First we are going to calculate the values for the quivers
    u,v = [x_0], [0.0]
    for i in 1:n
        # hit the line of the function
        push!(u,u[end])
        push!(v,x′(u[end],a))
        # then the line of x' = x
        push!(u,v[end])
        push!(v,v[end])
    end
    U,V = u[2:end]  .- u[1:N], v[2:end]  .- v[1:N]
# get a range of x values
x = collect(LinRange(0,2,100))
# plot x = x'
plt = plot(x,x, linestyle = :dash, label = false, grid = false)
# plot the function
plot!(x,x′.(x,a), label = "a = $(a)",color = :red, linewidth = 2)
# plot the arrows
quiver!(u[1:N],v[1:N],quiver=(U,V), color = :black,
arrow=(:closed, 2.0),
 linewidth = 2)
display(plt)
end
```

```
spider (generic function with 3 methods)
```

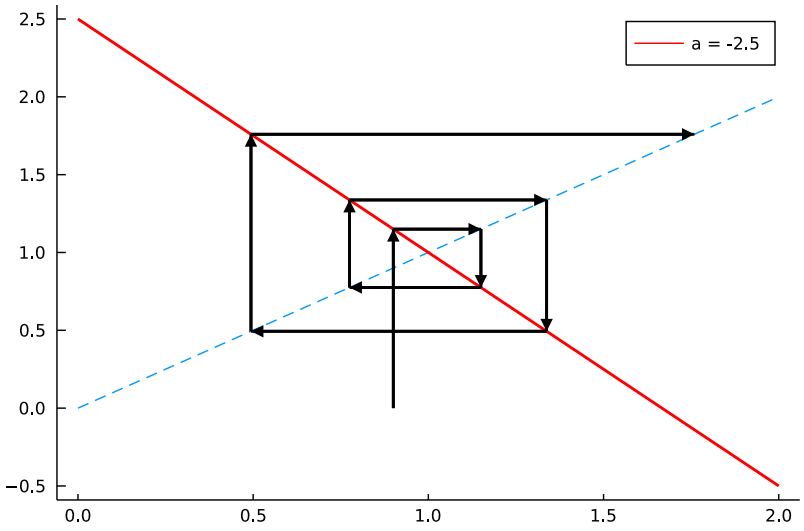
```
spider(-1.5,5)
```




```
spider(-2.0,5)
```




```
spider(-2.5,5, 0.9)
```



Two way mutation

$$p' = p - m_{12}p + m_{21}(1-p)$$

Show that this system has a single stable equilibrium at $\hat{p} = \frac{m_{21}}{m_{12}+m_{21}}$
At equilibrium we know that in general $p' = p$. Alternatively, we can also see from the problem description that $m_{12}p = m_{21}(1-p)$ at equilibrium. Thus,

$$\begin{aligned} 0 &= -m_{12}p + m_{21}(1-p) \\ &= -m_{12} + m_{21}/p - m_{21} \\ \frac{1}{\hat{p}} &= \frac{m_{12}+m_{21}}{m_{21}} \\ \hat{p} &= \frac{m_{21}}{m_{12}+m_{21}} \end{aligned}$$

This recovers the equation from the book. To show that this equilibrium is stable we will use linear stability analysis. We evauluate the derivative of the recursion, at equilibium ($p = \hat{p}$). And if the value of the derivative is smaller than 1 and greater than -1, then the equilibrium is stable.

$$\begin{aligned} p' &= p - m_{12}p + m_{21}(1-p) \\ \frac{dp'}{dp} &= \frac{d}{dp}(p - m_{12}p + m_{21}(1-p)) \\ &= 1 - m_{12} - m_{21} \end{aligned}$$

We can also do this in Julia

```
@variables p m12 m21
p′  = p - m12 * p + m21 *(1-p)
dpdt = Symbolics.derivative(p′, p)
latexify(dpdt)
```

$$1 - m12 - m21 \tag{1}$$

This is not a function of p. 1 minus two non-zero values which are both by definition smaller than 1 must be smaller than 1 and geater than minus 1. Thus, this is a (globally?) stable equlibrium.

Social learning and replicator dynamics

An organism that is capable of social learning can learn one of two behaviours. One has a higher payoff, but individuals do not always recognise which behaviour has the higher payoff. They acquire the more profitable behaviour $1 - e$ of the time and the less profitable one e of the time. We can make a mating table *programatically* by storing the relevant values in dictioanries and the using list comprehensions to do the maths. This is probably overkill in this instance! We can do the sums by hand and we could also find a more elegant solution using arrays. But I think using dictionaries scales well to more complicated problems.

```
@variables e p Δp p′
freq = Dict("A" => p, "B" => 1-p)
prob =  Dict("A" => 1-e, "B" => e)
self = ["A","A","B","B"]
other = ["A","B","A","B"]
probability = [freq[self[i]] * freq[other[i]] for i in 1:4]
Pr_A = [sum([self[i],other[i]] .=="A") for i in 1:4] .*
prob["A"]
Pr_B = [sum([self[i],other[i]] .=="B") for i in 1:4] .*
prob["B"]
Pr_A,Pr_B = (Pr_A ./ (Pr_A .+ Pr_B)),(Pr_B ./ (Pr_A .+ Pr_B))

matingTable = DataFrame([self,other,probability,Pr_A,Pr_B],

["self","other","probability","Pr(A)","Pr(B)"])

latexify(matingTable)
```

self	other	probability	Pr(A)	Pr(B)
A	A	p^2	1	0
A	B	$p\left(1-p\right)$	$1-e$	e
B	A	$p\left(1-p\right)$	$1-e$	e
B	B	$\left(1-p\right)^2$	0	1

So this is the correct mating table. We next need to show that the change in the frequency of the more profitable trait is

$$\Delta p = p(1-p)(1-2e)$$

First we make a vector of the probability od A being learned in any given interaction type weighted by the probability (as a function of p) of that interaction taking place.

```
# Let's create an array of matingTable, as I find them easier
to manipulate.
matingArray = Array(matingTable)
# now we get our weighted probability vector by taking the
product of columns 3
# and 4 along the 2nd (row) dimension
weighted_A = prod(matingArray[:,3:4], dims = 2)
latexify(weighted_A)
```

$$\begin{bmatrix} p^2 \\ p\left(1-e\right)\left(1-p\right) \\ p\left(1-e\right)\left(1-p\right) \\ 0 \end{bmatrix} \tag{2}$$

Now we sum all the weighted probabilities (of learning behaviour A) to get the total frequency of A at $t+1$.

```
latexify(p′ ~sum(weighted_A) )
```

$$p' = p^2 + 2p\left(1-e\right)\left(1-p\right) \tag{3}$$

To get the quation in the book

$$\begin{aligned} p' &= p^2 + 2p\left(1-e\right)\left(1-p\right) \\ &= p^2 + 2p\left(1-p\right) - e2p\left(1-p\right) \\ &= p^2 + 2p - 2p^2 - e2p\left(1-p\right) \\ &= p + (p - p^2) - e2p\left(1-p\right) \\ &= p + p(1-p)(1-2e) \end{aligned}$$

Then to get the *change* in p we subtract p from both sides.

$$\Delta p = p(1-p)(1-2e)$$

When fitness is not constant

Let w_{AA}, w_{AS} and w_{SS} be the fitness of each genotype of a diploid organism with alleles A and S.

```
@variables w_AA w_AS w_SS p
```

$$\begin{bmatrix} w_{AA} \\ w_{AS} \\ w_{SS} \\ p \end{bmatrix} \tag{4}$$

The change in frequency of allele A is given by

$$\Delta p = p(1-p)\frac{w_A - w_S}{\bar{w}}$$

Find expressions for w_A, w_S and $\bar{w}$

The fitness of each allele is the fitness of each genotype weighted by the likelihood of a gamete of the given allele ending up in that genotype. This likelihood is (assuming random mating) the frequency of the other allele making up the genotype.

```
w_A = p* w_AA + (1-p) *w_AS
w_S = p* w_AS + (1-p) *w_SS
wbar = p * w_A + (1 - p) * w_S
latexify(string(wbar))
```

$$p \cdot (p \cdot w_{AA} + w_{AS} \cdot (1-p)) + (1-p) \cdot (p \cdot w_{AS} + w_{SS} \cdot (1-p))$$

show that the system has an equilibrium at

$$\hat{p} = \frac{w_{AS} - w_{SS}}{2w_{AS} - w_{AA} - w_{SS}}$$

At equilibrium $\Delta p = 0$. so we can see straight away that theres an equilibruim when either of the right hand side (rhs) terms $= 0$. There are 2 boring ones at $p = 0$ and $p = 1$ i.e., 1 allele is at fixation. Lets find the values for which $\frac{w_A - w_S}{\bar{w}} = 0$. Clearly, this is satisfied when $w_A = w_S$.

```
expand(w_A - w_S)
```

$$w_{AS} + pw_{AA} + pw_{SS} - w_{SS} - 2pw_{AS} \tag{5}$$

$$
\begin{aligned}
0 &= w_{AS} + pw_{AA} + pw_{SS} - w_{SS} - 2pw_{AS} \\
2pw_{AS} - pw_{AA} - pw_{SS} &= w_{AS} - w_{SS} \\
p(2w_{AS} - w_{AA} - w_{SS}) &= w_{AS} - w_{SS} \\
\hat{p} &= \frac{w_{AS} - w_{SS}}{2w_{AS} - w_{AA} - w_{AA}}
\end{aligned}
$$

(Using excel!!) plot the dynamis of the system

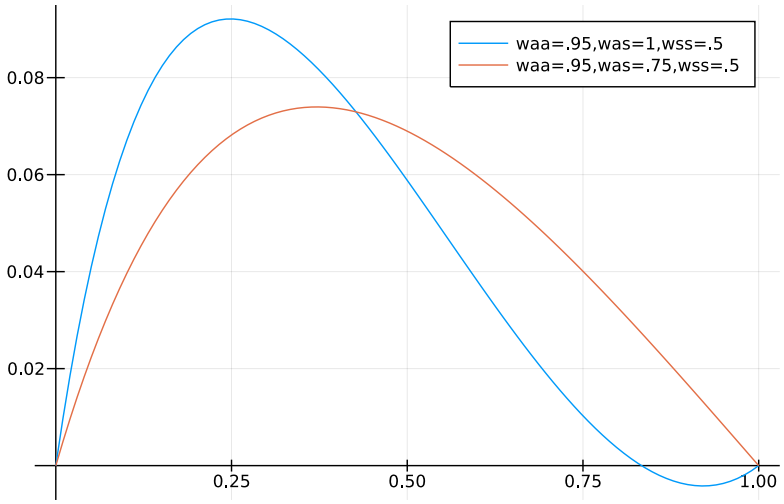
So, I'm not ging to use excel. Instead I'l use this as an excuse to try out Symbolics.jl function building. We should be able to convert ymbolic expressions from the earlier parts of the problem in th generic Julia functions. **buil_function** generates Julia code, and then we use **eval** to assign the generated code to a function name. We will be reusing expression names as variables and variable names as functions etc. so we will protect ourselves by wrapping everything in a *let* block. This means that variable assignments will only be valid within this block.

```
let ΔP(p,w_A, w_S,wbar) = p *(1-p)*(w_A-w_S)/wbar
    # Now we've defined our difference equation,
    # we can build some functions from symbolic expressions
    WA = eval(build_function(w_A, w_AA,w_AS, p))
    WS = eval(build_function(w_S, w_AS,w_SS, p))
    WBAR = eval(build_function(wbar,w_AA, w_AS,w_SS, p))
    # and we can wrap it all up in a single function
    function trajectory(w_AA,w_AS,w_SS,
            P = collect(LinRange(0,1,100)))

        w_A = WA.(w_AA,w_AS, P)
        w_S = WS.(w_AS,w_SS, P)
        wbar = WBAR.(w_AA, w_AS,w_SS, P)

        P′ = ΔP.(P,w_A, w_S,wbar)
    return(P, P′)
    end
    plot(trajectory(0.9,1,0.5), framestyle = :origin,label =
"waa=.95,was=1,wss=.5")
    plot!(trajectory(0.9,0.75,0.5), label =
"waa=.95,was=.75,wss=.5")

end
```



When environments vary with time

Next we consider a haploid population with two genotypes , in an environment with 2 seasons, wet and dry. Individuals are born at the start of each season. They reproduce and die by the end of the season hey are born in. Genotype A has fitness 1 in either season. B has fitness 0.2 in the dry season and 2 in the wet season. Show that the only stable equilibrium is a popolation consiting of just genotyoe A.

Let's start with the dry season at t, thus t' is the wet season of the same year and $t+1$ is the dry season of the following year. The frequency of genotype A is p, which transitions to p' and p'' as t goes to t' and $t+1$.

$$\begin{aligned} p' &= \frac{p}{p + 0.2(1-p)} \\ &= \frac{p}{0.8p + 0.2} \end{aligned}$$

$$\begin{aligned} p'' &= \frac{p'}{p' + 2(1-p')} \\ &= \frac{p'}{2 - p`} \end{aligned}$$

```
@variables p
p′ = p/(0.8 * p + 0.2)
p′′ = p′/(2 - p′)
```

$$\frac{p}{\left(2 - \frac{p}{(0.2+0.8p)}\right)(0.2+0.8p)} \tag{6}$$

This equation is pretty ugly and neither *expand* nor *simplify* fix the problem. I'm sure it wouldn't be hard to simplify with pen and paper. However, there is a more direct approach to calculating p'' in terms of p. We can treat this as a two step selection process, just like viability and fecundity selection Box 1.2 in the book. Which means we can directly calculate p'' using the products of the fitness in wet and dry seasons.

$$\begin{aligned} p'' &= \frac{p}{p + 0.4(1-p)} \\ &= \frac{p}{0.4 + 0.6p} \end{aligned}$$

$$\implies \Delta p = p(1-p)\frac{0.6}{0.4+0.6p}$$

```
plot(p″,0:0.05:1, linestyle = :dash, label = "ugly eq") #you can
"plot()" a symbolic expression. Cool.
plot!(0:0.05:1,[p/(0.4 + 0.6 *p) for p in 0:0.05:1],
linestyle = :dot, label = "elegant" )
```

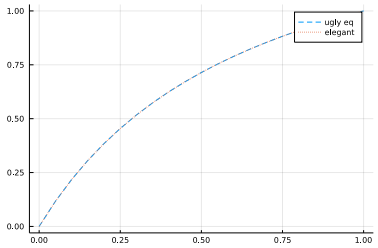


Figure 1: Lets do a quick sanity check, to make sure both metods give the same answer. Yep.

```
plot(p′ ′,0:0.05:1, linestyle
= :dash, label = "ugly eq")
#you can plot() a symbolic
expression. Cool.
plot!(0:0.05:1,[p/(0.4 +
0.6 *p) for p in 0:0.05:1],
linestyle = :dot, label =
"elegant" )
```

The only way to make the second term $= 0$ is if $0.6p = -0.4$, which is logically impossible. So we are left with fixed points at $p = 0$ and $p = 1$. A glance at the difference equation shows that there's no value of p between 0 and 1 that would cause a decrease in p. But we won't always be so lucky, so lets use linear stability analysis again. Remember, we take the derivative of the recusrion **not** the difference equation. So we can use our ugly looking recursions we calculated using Symbolics.

```
substitute(Symbolics.derivative(p″, p),Dict(p=> 0))
```

$$2.5 \tag{7}$$

$$> 1 \implies \text{unstable}$$

```
substitute(Symbolics.derivative(p″, p),Dict(p=> 1))
```

$$0.3999999999999999 \tag{8}$$

$$> -1, < 1 \implies \text{stable}$$

Horizontal cultural transmission, again

Now we consider a population of individulas who live forever and learn 1 of 2 behaviours A or B, with $V(A) > V(B)$. Every time period individuals are paired at random, they observe the behaviour and payoff of their partner and if their partner has a higher payoff, they swith with probability β(partner's payoff − own payoff) (the problem statement in the book is incorrect). p is the frequency of A. And we are to show that

$$\Delta p = \beta(V(A) - V(B))$$

We can start by constucting a mating table

```
@variables β p VA VB p′
freq = Dict("A" => p, "B" => 1-p)
self = ["A","A","B","B"]
other = ["A","B","A","B"]
probability = [freq[self[i]] * freq[other[i]] for i in 1:4]
Pr_A = []
Pr_B = []
for i in 1:4
    if self[i] == "A"
        push!(Pr_A,1)
        push!(Pr_B,0)
    elseif other[i] == "A"
        push!(Pr_A, β * (VA - VB))
        push!(Pr_B,1-(β * (VA - VB)))
    else
        push!(Pr_A,0)
        push!(Pr_B,1)
    end
end
matingTable = DataFrame([self,other,probability,Pr_A,Pr_B],

["self","other","probability","Pr(A)","Pr(B)"])

latexify(matingTable)
```

self	other	probability	Pr(A)	Pr(B)
A	A	p^2	1	0
A	B	$p\left(1-p\right)$	1	0
B	A	$p\left(1-p\right)$	$\beta\left(VA-VB\right)$	$1-\beta\left(VA-VB\right)$
B	B	$\left(1-p\right)^2$	0	1

Again, this could easily be done by hand, probably in less time. But forcing yourself to write down a set of rules for the computer to follow can actually be a good way to confirm you've understood the *general* solution to the problem, rather than just the specific case. Next we again take a vector of Pr_A weighted by the probability of the relevant pairing occcuring in the population.

```
matingArray = Array(matingTable)
```

```
weighted_A = prod(matingArray[:,3:4], dims = 2)
latexify(weighted_A)
```

$$\begin{bmatrix} p^2 \\ p\left(1-p\right) \\ p\beta\left(VA-VB\right)\left(1-p\right) \\ 0 \end{bmatrix} \tag{9}$$

And sum all the weighted probabilities (of learning behaviour A) to get p'.

```
latexify(p′ ~sum(weighted_A) )
```

$$p' = p^2 + p\left(1-p\right) + p\beta\left(VA-VB\right)\left(1-p\right) \tag{10}$$

And then we simplify

$$\begin{aligned} p' &= p^2 + p\left(1-p\right) + p\beta\left(VA-VB\right)\left(1-p\right) \\ &= p + p\left(1-p\right)\beta\left(VA-VB\right) \end{aligned}$$

Subtracting p form both sides gives

$$\Delta p = p\left(1-p\right)\beta\left(VA-VB\right)$$

Seems to be another typo in the book.