MODELO SIR: CONCEPTOS

Los modelos SIR formalizan la que es probablemente la forma más simple posible de pensar en una epidemia:

- Población susceptible de ser infectada: esta población cuando pasa el tiempo se vuelve en población infectada
- Población que ya está infectada: esta población cuando pasa el tiempo se vuelve en población recuperada
- Población que ya se ha recuperado

- antes de que comience la epidemia, hay una población en riesgo de contraer la enfermedad.

- la enfermedad se propaga de personas infectadas a susceptibles.
- después de un tiempo, las personas ya no están infectadas.
- una vez infectadas, las personas nunca más se infectan.
- ignoramos todas las otras diferencias entre las personas.

Usamos el símbolo S para la cantidad de personas que todavía están en riesgo de contraer la enfermedad. "S" es la abreviatura de "susceptible", lo que significa una persona que no ha tenido la enfermedad y podría contraerla. Pensamos en S como una función del tiempo, es decir, creemos que a medida que avanza la epidemia, S cambia. Por lo tanto, escribimos $S(t)$ para el número de susceptibles en el momento t.

El número $S(0)$, que es el número de susceptibles al comienzo de la epidemia, es un valor clave en este modelo.

El número de personas infectadas en el momento t viene dado por $I(t)$. Suponemos que $I(0) > 0$, porque de lo contrario la epidemia no puede comenzar.

Finalmente, las personas previamente infectadas pero que ya no son infecciosas se llaman "eliminadas". (Desafortunadamente, esto no significa que se hayan recuperado. Todavía pueden estar enfermos, o como sucede trágicamente a menudo con el Ébola, pueden estar muertos). Usamos $R(t)$ para simbolizar el número de personas eliminadas en el momento t.

¡NÓTESE BIEN! No es necesario que S, I y R sean números enteros. Este es un modelo, y todo lo que queremos es que los valores pronosticados de S, I y R estén cerca de lo observado. Por cierre queremos decir dentro de un pequeño porcentaje. De hecho, debido a que los modelos son tan simples y la situación real tan variable, obtener números dentro de un pequeño porcentaje es un logro importante.

Desafortunadamente, esto generalmente es posible solo en retrospectiva. Mostraremos que un modelo SIR puede funcionar bastante bien para la epidemia de EVD de África Occidental de 2014, pero será en retrospectiva. El curso de la epidemia no se predijo con éxito hasta que casi terminó.

ECUACIÓN MODELO SIR

Modelaremos S, I y R en pasos de tiempo discretos. De hecho, utilizaremos el mismo período de tiempo dt para cada paso. Denotamos por t_i el tiempo después de i pasos. Como suponemos que $t_0 = 0$, tenemos $t_i = i\,dt$.

El modelo pasa de t_i a t_{i+1}, así que lo que queremos hacer es usar los valores de $S(t_i)$, $I(t_i)$ y $R(t_i)$ para predecir los valores $S(t_{i+1})$, $I(t_{i+1})$ y $R(t_{i+1})$. Entonces nuestro modelo está configurado en términos de tres ecuaciones.

Todas estas ecuaciones tienen la forma **"valor nuevo = valor antiguo + ganancias - pérdidas"**. El modelo especifica cómo calculamos las ganancias y pérdidas.

- λ Tasa de infección, cada virus tiene una tasa de infección determinada.
- γ Tasa de recuperación

Para susceptibles, solo hay un término de pérdida. Los modelos SIR más simples provienen de la llamada **"ley de acción de masas"**. Esta ley sostiene que todos los susceptibles tienen las mismas posibilidades de encontrarse con una persona infectada. Y lo mismo se supone para las personas infectadas. Entonces, el número de reuniones de infectados y susceptibles es proporcional al producto SI. Usamos el símbolo λ para la constante de proporcionalidad, e interpretamos λSI como la tasa a la que se infectan los susceptibles. Esto es, por supuesto, una tarifa por día. Entonces, la pérdida real durante el paso de tiempo dt es en realidad $\lambda SI\,dt$, entonces la ecuación para S es

1 ... $S(t_{i+1}) = S(t_i) - \lambda S(t_i) I(t_i)\,dt.$

Por otro lado, la tasa de pérdida para las personas infectadas es solo una probabilidad constante de recuperación por unidad de tiempo. Entonces, una fracción constante γ se elimina de los infectados por día. Esto da una pérdida de γI infectados por día, que es $\gamma I\,dt$ pérdida de infectados por paso de tiempo. La ganancia de los infectados debe ser exactamente los susceptibles que se infectan, por lo que la ecuación de I es

2 ... $I(t_{i+1}) = I(t_i) + \lambda S(t_i) I(t_i)\,dt - \gamma I(t_i)\,dt.$

En un modelo SIR, la ecuación de población eliminada solo tiene un término de ganancia, sin término de pérdida. Lo que se pierde de los infectados se gana con las eliminaciones, por lo que la ecuación de R es

3 ... $R(t_{i+1}) = R(t_i) + \gamma I(t_i)\,dt.$

Nuestro modelo SIR discreto consta de las ecuaciones 1, 2 y 3 para S, I, R.

Coódigo en Julia: la función para un paso de tiempo

Así que ahora consideramos la mejor manera de escribir el código de Julia. Tenemos varias opciones: podríamos escribir un bucle "for" con las tres ecuaciones dentro; podríamos

hacer tres funciones, una para cada S, I y R; podríamos combinar los tres en una matriz y escribir una función para actualizar los tres al mismo tiempo. Haremos eso aquí.

Las funciones de Julia tienen acceso a todas las variables que son visibles en el nivel en que se crea la función

Esta característica de Julia sorprenderá a muchos programadores experimentados. Si asignamos un valor a la variable b y luego llamamos a una función que usa b sin mencionar b en el argumento de la función, la función usará el valor de b según sea necesario. Por ejemplo:

[2]
```
b = 4
f(x) = b*x
f(6)
```

```
24
```

Utilizaremos esta función para pasar los valores de $lambda$ y $gamma$ a la función de actualización. Es decir, al escribir el código para la función, le pasamos solo los argumentos S, I, R y tomamos los valores de los parámetros del contexto en el que se llama la función. En realidad, esto es increíblemente conveniente, especialmente cuando queremos experimentar con diferentes valores de parámetros con el fin de ajustar la salida del modelo a los datos.

Esto significa que la función se ve así:

[3]
```
function updateSIR(popnvector)
    susceptibles = popnvector[1];
    infecteds    = popnvector[2];
    removeds     = popnvector[3];
    newS = susceptibles - lambda*susceptibles*infecteds*dt
    newI = infecteds + lambda*susceptibles*infecteds*dt -
gam*infecteds*dt  #note abbreviation for gamma (see below)
    newR = removeds + gam*infecteds*dt
    return [newS newI newR]   # NB! spaces only to make this a
one row of a two-dimensional array
    #                   and note the use of "return" to specify
what the function output should be
end
```

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

Siempre se debe hacer al menos una comprobación mínima de que una función funcione como se espera.

Para el código profesional, uno debe hacer una verificación exhaustiva, tanto de la salida como de la entrada ... de hecho, debe escribir la función de modo que si falla, falla con gracia, y darle cierta protección contra las condiciones de error. ¡Pero no estamos escribiendo código profesional!

Por otro lado, aprovechemos también para ilustrar algunos errores típicos y cómo resolverlos.

[4]
```
# first we set the parameters
dt = 0.5                    # so we are taking two steps per day
lambda = 1/200; gam = 1/10  #note that gamma is a function in the
stats packages, so let's not use it as a name

# then we specify the input vector to the function
s, i, r = 1000., 10, 20  # multiple assignment
vec = [s i r]          # followed by creating the input vector;
spaces again so that it is a row of an array
updateSIR(vec)         # finally the actual function call to test
the function
```

```
1×3 Array{Float64,2}:
 975.0  34.5  20.5
```

Ahora consideramos la estructura de bucle. No podemos ejecutar el modelo indefinidamente, por supuesto, por lo que debe haber un tiempo de finalización definido T_f. Usaremos $T_f = 610$ días, que es solo un poco más que el número de días epidémicos en los datos de Wikipedia.

El número de pasos es, en principio, $n = T_f/dt$, pero aquí hay un momento delicado en el uso de las computadoras. La aritmética de la computadora no es infinitamente precisa, por lo que no podemos estar seguros de que si usamos la computadora para calcular n de esta manera, tendremos exactamente $n\,dt = T_f$ como esperamos sobre la base del álgebra. Me gusta establecer el tiempo como una variable separada, hacer un seguimiento explícito del número de paso i y continuar durante n pasos. Entonces, si al final el tiempo no es exactamente T_f, no importa, porque estará extremadamente cerca de él.

El mensaje final es NO CONFÍE EN LA IGUALDAD EXACTA, incluso si puede esperarlo razonablemente. Las computadoras normalmente están muy cerca de los números en los que usted, el programador, está pensando (excluyendo nuevamente la exactitud que los programadores profesionales pueden alcanzar).

Conocer el valor de n también nos permite inicializar una matriz para contener todos los valores a medida que se recopilan, una fila de tres elementos para cada valor del tiempo. Esto terminará siendo $n + 1$ valores, por supuesto.

Así que aquí está el código conceptual:

Corriendo el modelo

Ahora podemos ejecutar el modelo. Para ilustrar los pasos, pego la celda de arriba en la celda de abajo y proporciono los bits faltantes, esto incluye los valores de los parámetros que configuramos antes, pero no la función updateSIR, ya que no deseamos modificar eso.

[10]
```
# set the values that define the current run
lambda = 1/20000   # infection rate parameter (assumes rates are
per day)
gam = 1/10         # recovery rate parameter  (ditto)
dt = 0.5           # length of time step in days
tfinal = 610;      # respecting community values: lowercase only in
the names
s0 = 10000.0       # initial susceptibles, note that we use the
type Float64 from the start
i0 = 4.            # initial infecteds; set this to 1. to  mimic an
epidemic with an index case
r0 = 0.            # not always the case, of course

# initialise
nsteps = round(Int64, tfinal/dt)    # note the use of round()
with type Int64 to ensure that nsteps is an integer
resultvals = Array{Float64}(undef, nsteps+1, 3)  #initialise
array of type Float64 to hold results
timevec = Array{Float64}(undef, nsteps+1)        # ... ditto for
time values
resultvals[1,:] = [s0, i0, r0]  # ... and assign them to the
first row
timevec[1] = 0.                 # also Float64, of course.

for step  = 1:nsteps
    resultvals[step+1, :] = updateSIR(resultvals[step, :])  # NB!
pay careful attention to the rows being used
    timevec[step+1] = timevec[step] + dt
end
```

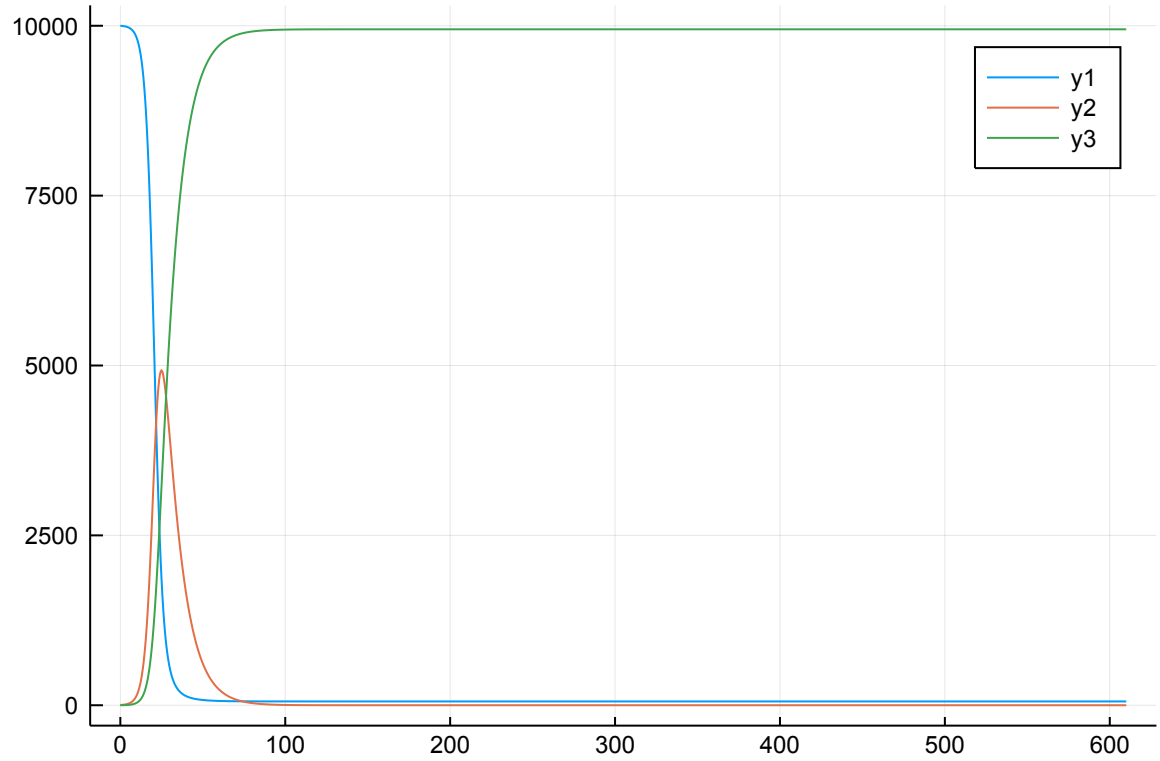
Graficando los resultados

[11]
```
using Plots
```

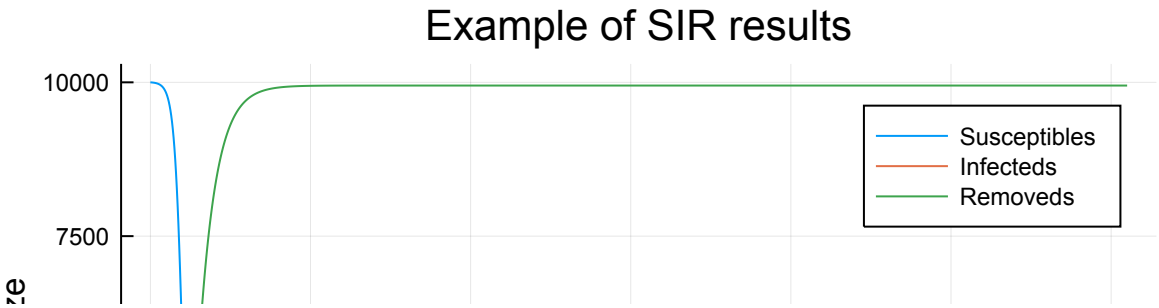
[12]
```
gr()
```

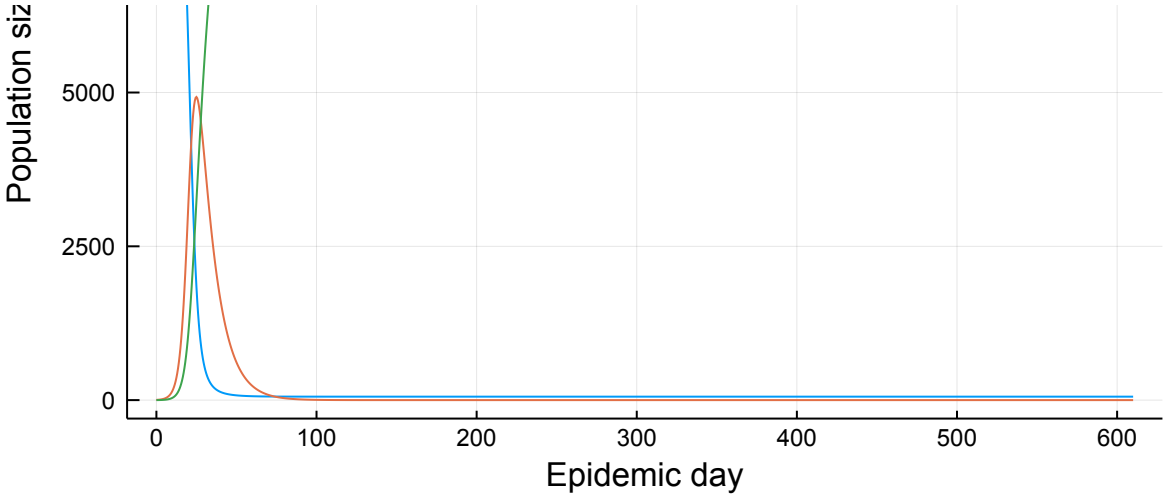
```
Plots.GRBackend()
```

[13]
```
plot(timevec, resultvals)
```



[14]
```
plot(timevec, resultvals,  # we should of course at a minimum
provide some labels
title  = "Example of SIR results",
xlabel = "Epidemic day",
ylabel = "Population size",
label  = ["Susceptibles" "Infecteds" "Removeds"]
)
```




Population siz
5000
2500
0
0
100
200
300
400
500
600
Epidemic day