Instituto Tecnológico y de Estudios Superiores de Monterrey

MODELO SIR Y SOLUCIONES POR MEDIO DE MÉTODOS ESTOCÁSTICOS

Modelación numérica de sistemas estocásticos

Equipo 3

Izael Manuel Rascón Durán - A01562240
David Camela Equihua - A01734025
Juan Vladimir Padilla Lopéz - A01655339
Karina Abboud Herrera - A01704136
Eduardo Barroso Portugal-A01028685

Octubre de 2021

1. Introducción al modelo SIR

Los seres humanos suelen colocarse en un pedestal inmortal y superior al resto de la naturaleza. Esto conlleva a que solemos olvidar que podemos ser hasta más vulnerables que los organizamos más simples. Un claro ejemplo de nuestra vulnerabilidad como especie son las epidemias. En la historia, con cada epidemia batallamos para solucionar este problema para evitar una posible extinción. Por ello, también nos aseguramos de aprender lo más posible sobre como actúan las mismas para desarrollar herramientas que nos permitan tener un mejor conocimiento y con ende adecuar los manejos y controles mas efectivos.

El modelo SIR fue propuesto por primera vez por los doctores Kermack y McKendrick en 1927, muchos años después de que el matemático Daniel Bernoulli(1700-82) iniciara el estudio para la modelación matemática de infecciones. Los recién nombrados lograron que a través de un sistema de ecuaciones diferenciales ordinarias se pudiera modelar el desarrollo de las epidemias contando con 3 funciones principales [1] [Giraldo, 2008]:

- Susceptibles $S(t)$
- Infectados $I(t)$
- Removidos $R(t)$

1.1. Susceptibles

Del total de la población, empezaremos teniendo a la mayor parte en la sección de susceptibles $S(t)$. Este sector nos indica que las personas aún no han contraído el virus pero aún pueden ser contagiados si entran en contacto con un infectado. La cantidad de población denominados susceptibles irá en disminución conforme avance el tiempo ya que se irán moviendo al sector de los infectados conforme indique la tasa de transmisión de la epidemia β.

1.2. Infectados

La función de infectados $I(t)$ describe aquellos agentes de la simulación que se encuentran infectados por el virus y además son los que pueden transmitir el patógeno. Se espera ver un comportamiento acampanado para esta función ya que es un parámetro por el que la población entra y sale. De manera similar a que el parámetro β se relaciona con la taza de infección, el parámetro γ es la capacidad de remoción de la epidemia.

1.3. Removidos

La función de removidos $R(t)$ describe aquellos agentes de la simulación que ya pasaron por un periodo de infección y de acuerdo al parámetro γ es que fueron removidos de una futura infección. Es importante no asociar este parámetro con recuperadosza que hay que tener en mente ya que la naturaleza de las pandemias también existen las personas fallecidas.

[1] Los modelos mas complejos se construyen en base a SIR pero tienen mas funciones que describen aspectos mas robustos de una epidemia. Como ejemplo, dichos modelos pueden considerar una población de vacunados y/o de asintomáticos.

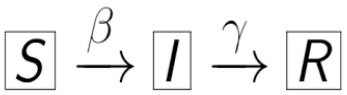

Figura 1: SIR Model

Las funciones, los parámetros y sus relaciones que se expusieron previamente se resumen en la figura 1. Una vez comprendida esta figura, es que podemos pasar a la parte matemática y física del modelo [Mollison et al., 1994]. Es en esta sección del trabajo donde se puede elevar el nivel de los conceptos. El nivel de abstracción va a ser directamente proporcional a que tan completo se desea hacer el modelo. Si se agregan variables externas o se consideran otras variables, por consecuencia, la matemática se va a hacer mas compleja. Un claro ejemplo de esto se ilustra en el articulo del doctor Gray [GRAY et al., 2011] donde se discuten las ideas detrás de un modelo SIS (Susceptible Infectado Susceptible) y como consecuencia, surge un sistema de ecuaciones diferenciales estocásticas.

Sin embargo, el modelo que se propone y trata en este documento asume una serie de consideraciones que hace de este un modelo relativamente sencillo y **deterministico**. Las consideraciones son:

- Población constante $N = S + I + R$
- Población homogénea (Mismos comportamientos)
- Periodo de latencia es despreciable-
- Inmunidad permanente

1.4. Derivación del modelo SIR

Se podría decir, que la derivación del modelo SIR es trivial. A continuación, se ofrece el pensamiento intuitivo correcto para lograr establecer las ecuaciones que rigen el modelo SIR:

Partiendo de la información que resume la figura 1 y las consideraciones expuestas, podemos intuir que lo que necesitamos describir es el **cambio** en las funciones *SIR*. Esto, por supuesto, se hace con derivadas temporales.

$$\frac{dS}{dt}, \frac{dI}{dt}, \frac{dR}{dt}$$

Es razonable intuir que el cambio en susceptibles $S(t)$ es proporcional a la cantidad de infectados, de susceptibles y la taza de infección β. Además, dicha cantidad tiene que ser negativa ya que esta función solo puede decrecer en el tiempo a medida de que las personas se van infectando y removiendo. Agregando el factor de normalización N, nos queda que:

$$\frac{dS}{dt} = -\frac{S(t)I(t)\beta}{N} \tag{1}$$

En el modelo SIR, todo lo que entra, tiene que salir [Dukic et al., 2012]. Es por esto que lo que disminuye S $\frac{ds}{dt}$ lo tiene que aumentar el cambio de los infectados. Sin embargo, también hay una salida de la infección y esta se dicta por el parámetro γ de tal forma que si hay muchos infectados, va a haber mayor probabilidad de generar recuperados. Todo esto nos deja con:

$$\frac{dI}{dt} = \frac{S(t)I(t)\beta}{N} - \gamma I(t) \tag{2}$$

Siguiendo la lógica que todo lo que sale tiene que entrar, definimos que los removidos se rigen por la siguiente ecuación:

$$\frac{dR}{dt} = \gamma I(t) \tag{3}$$

Estas 3 ecuaciones, es lo que conforma el modelo SIR. La solución a dicho sistema de 3 ecuaciones diferenciales se puede obtener con **métodos numéricos** haciéndolo un modelo **determinista**. Su solución se puede ver gráficamente ilustrada en la figura 2.

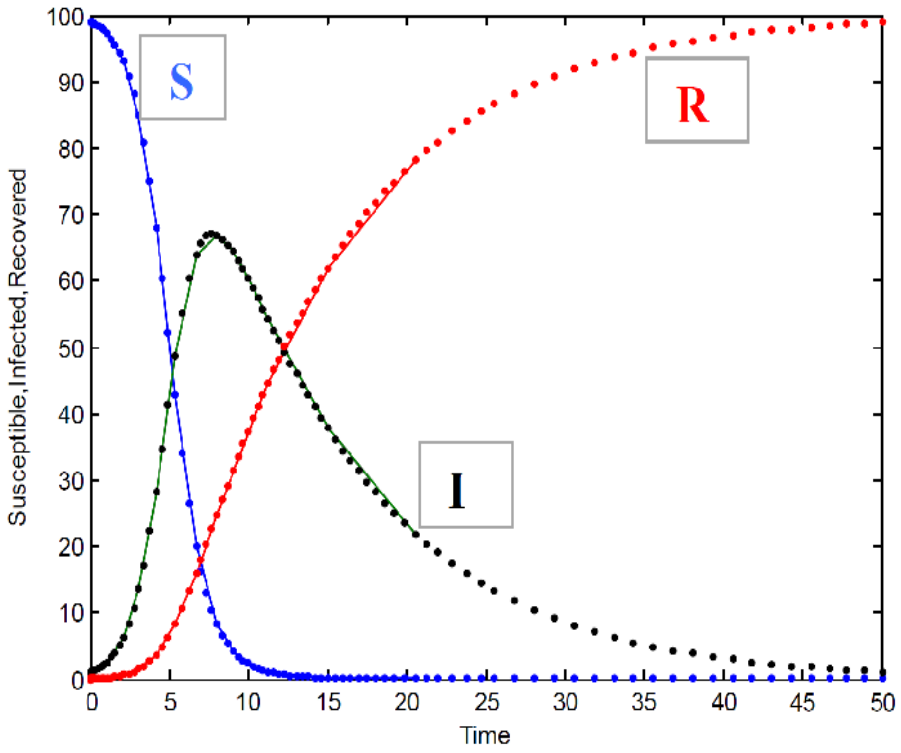

Figura 2: SIR solution

2. Difusión de un virus por caminata aleatoria

El objetivo de esta practica es comenzar a entender los procesos estocásticos que se pueden incluir en distintos modelos. A manera de ejemplo, se propone el siguiente escenario donde es necesario incluir procesos estocásticos. Recuerda que un proceso estocastico se define de la siguiente manera:

def. Proceso estocástico: *Conjunto de variables aleatorias que evolucionan de acuerdo a un parámetro. Usualmente, el tiempo.*

Escenario:

Difusión de un virus por caminata aleatoria . Se tiene una bacteria confinada en un tubo de carbono de tal manera, que la bacteria sólo se puede mover a la izquierda o derecha (haciéndolo con igual probabilidad) en cada unidad de tiempo. Si inicialmente está en x=0, y cada movimiento es de +1 o -1 metro, al considerar 100,000 unidades de tiempo:
A) ¿Cuál es el valor promedio del desplazamiento de la bacteria?
B) ¿Cuál es el valor promedio de la distancia recorrida por la bacteria?
C) Explica la diferencia entre los valores de A) y B). ¿Hay algún error cometido en la simulaciones?

Solución:

Codigo

Para la solución de la difusión de este virus se realizó un código en Matlab, la estructura básica del código consiste de dos ciclos **for**, de modo que el primer ciclo ejecuta todos los experimentos hasta $N = 10,000$ y dentro de cada ciclo, existe otro ciclo que va desde $2 : t$ donde $t = 100,000$ que son los pasos de tiempo.

```
%% Simulate displacement
for jj=1:n
    x=zeros(1,N);              %Preallocate position
    dx=zeros(1,N);             %Preallocate movment (1||0)
    Ecalc_disp=zeros(1,N);     %Preallocate dummy for expected val
    Ecalc_dist=zeros(1,N);     %Preallocate dummy for expected val

    absmoti=0;          %Absolute motion
    rig=0;              %Count right
    lef=0;              %Count left

    for j=1:N
        mov=round(rand);          %Random Movment
        absmoti=absmoti+1;        %Displacement Counter

        if mov==1
            dx(j)=1;              %distance
```

```
18              x(j+1)=x(j)+dx(j);   %Position of bacteria
19              rig=rig+1;           %Counter right
20              pr_x(rig)=rig/(j);   %Instant Probability of right
21
22          elseif mov==0
23              dx(j)=-1;            %distance
24              x(j+1)=x(j)+dx(j);   %Position of bacteria
25              lef=lef+1;           %Counter of left
26              pl_x(lef)=lef/(j);   %Instant Probability of left
27          end
28
29          Ecalc_dist(j)= (dx(j)*P_dist).^2;
30          Ecalc_disp(j)= dx(j)*P_disp;
31      end
32
33      Exval_dist(jj)=sqrt(sum(Ecalc_dist))*sqrt(2/pi);%Expected value
34      Exval_disp(jj)=sum(Ecalc_disp);                 %Expected value
35  end
```

2.1. Inciso A)

Realmente, lo que describe el valor esperado - ó promedio - del desplazamiento de la bacteria es el desplazamiento con respecto al inicio ($x = 0$) que se esperaría observar al repetir el experimento varias veces. Es importante recalcar que dicha cantidad, si considera los signos negativos. Es por esto que se debería de observar un valor cercano a cero ya que la probabilidad de moverse hacia la derecha ó izquierda es del 50 %. Esto significa que lo mismo que avanza, con regularidad es lo mismo que retrocede, por lo que al hacer un promedio y obtener un desplazamiento, obtenemos un valor aproximado a cero.

Resultado de simulación:$-8.0[m]$
Los resultados completos se muestran en la figura 3. Si se hacen varias simulaciones, se nota como este valor se distribuye en cero.

2.2. Inciso B)

Los resultados que se obtienen del valor esperado de la distancia no son triviales. Es importante conocer la distinción entre desplazamiento y distancia. Debido a que la distancia únicamente trata con valores absolutos, este parámetro realmente mide que tanto recorre la bacteria. La derivación matemática es algo compleja y fuera de lo que se trata dar a entender con este trabajo. Sin embargo, la idea es la siguiente:
Para el caso de la bacteria con pasos unitarios, se puede definir la distancia (dis) y el desplazamiento (des) de la siguiente manera:

$$des = 1 + 1 + 1 - 1 - 1 + 1 - 1 ... N_{result}$$
$$des^2 = des \propto \sqrt{N}$$

Resultado de simulación:$252.31325[m]$
Los resultados completos se muestran en la figura 3.

2.3. Inciso C)

Como se aborda en los incisos anteriores, cuando se mide el desplazamiento promedio, se esta calculando la distancia final desde el punto donde quedó la partícula al origen ignorando el camino trazado de por medio para llegar al punto final. En contraste, al calcular el promedio de distancia recorrida no se le da tanta importancia al punto final. Mas bien, se considera la sumatoria de cada paso unitario recorrido por la bacteria. Es por eso que se habla del cuadrado del desplazamiento y se llega a que la distancia es proporcional a N. Esto explica porque el aparente error de los resultados en realidad, no es un error. Mas bien reafirma que muchas veces, los resultados no son intuitivos y sin embargo, con un entendimiento mas profundo del modelo, se concluye que las matemáticas y con ende el modelo, son correctos.

2.4. Simulación y conclusiones

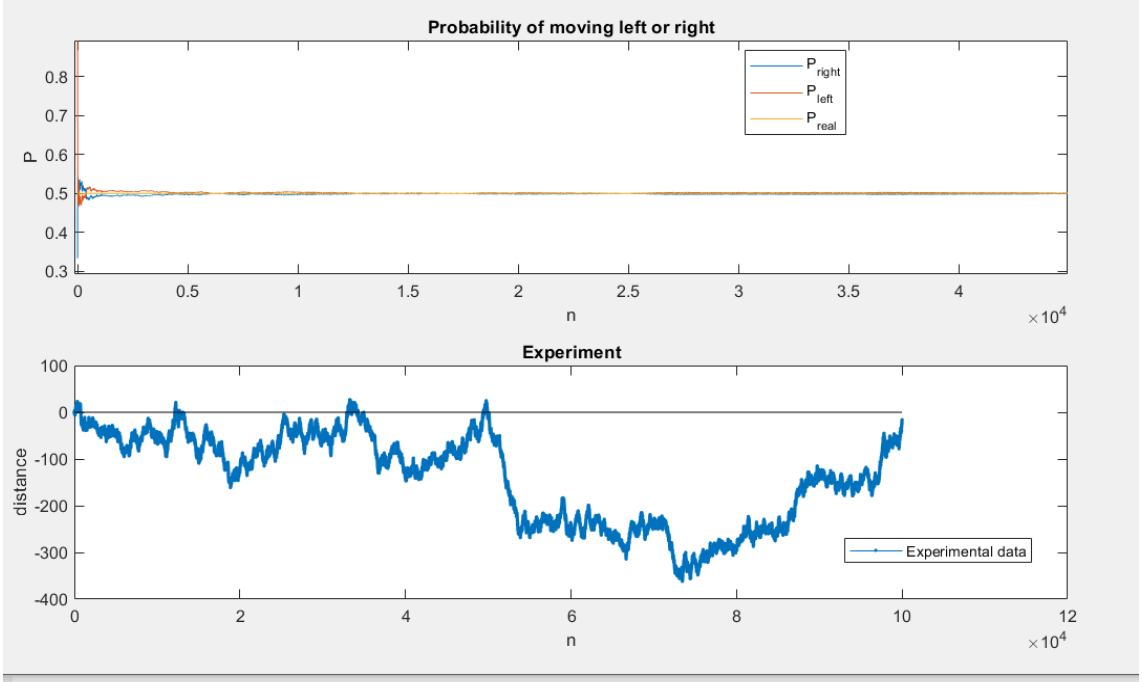

Figura 3: Resultados de la simulación propuesta.

En dicho modelo, se utilizaron métodos estocásticos en el sentido que se simulo una variable aleatoria y se modelo su evolución el tiempo. La gráfica de color azul de la figura 3 muestra la evolución de la variable a través del parámetro n.

3. Resolver el modelo SIR para algunos casos particulares.

Resolver el modelo SIR para algunos casos particulares. Se te pide obtener las gráficas de S,I y R para el modelo sin normalizar, considerando una población inicial de N = 1000, y una evolución de 100 días.

Además de las gráficas, explica el comportamiento observado en cada situación de la epidemia presentada, relacionando el comportamiento con los parámetros β y γ dados para cada caso.

3.1. Inciso i)

i) Con condiciones iniciales dadas por: [S0 = 997, I0 = 3], y parámetros $\beta = 0.5$ $\gamma = 0.5$

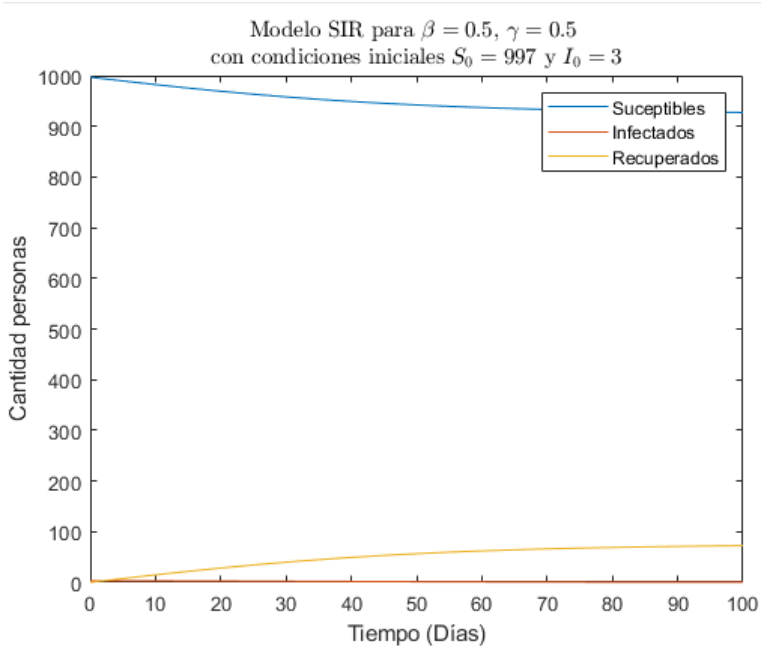

Figura 4: Desarrollo de la pandemia con $\beta = \gamma$

En este caso, como se puede observar, la disminución de los susceptibles es igual al incremento en los recuperados. Como consecuencia la población de los infectados no crecen, ya que siempre se mantiene el mismo número porque los que se enferman son exactamente igual a los que se van recuperando. En otras palabras se enferma 1, sale 1 y como iniciamos con 3 como condición inicial se mantiene en ese nivel.

3.2. Inciso ii)

ii) Con condiciones iniciales dadas por: [S0 = 990, I0 = 10], y parámetros $\beta = 0.5$ $\gamma = 0.05$

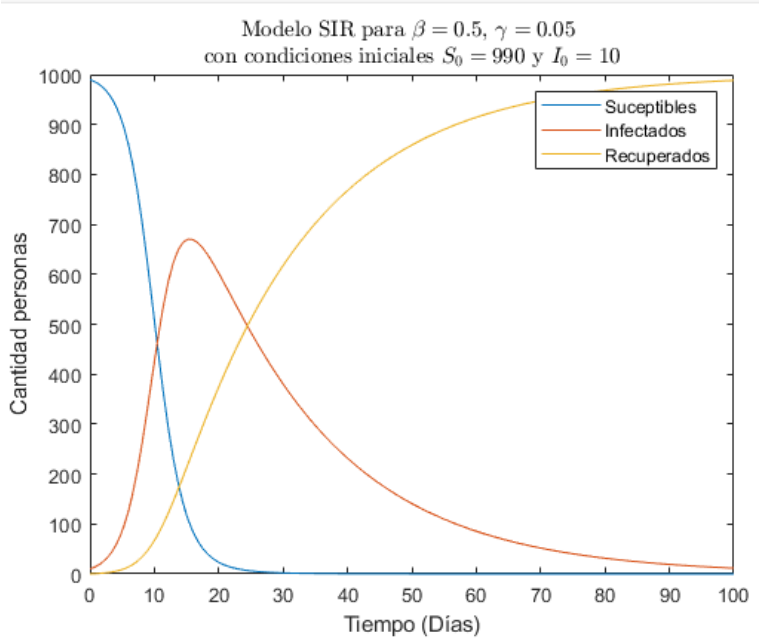

Figura 5: Desarrollo de la pandemia con $\beta = 0.5$ y $\gamma = 0.05$

Con este caso, es lo más parecido a la situación actual con la pandemia. Hay una mayor tasa de infectados contra los que se están recuperando. Llega a su pico en aproximadamente 17 días, los susceptibles desaparecen en 25 días y la población termina de recuperarse cerca de los 100 días predispuestos. Esto se debe a que la tasa de infección es 10 veces más grande que la de recuperación. Solo empieza a superar los recuperados a los infectados en el momento en que los susceptibles se ”terminan”.

3.3. Inciso iii)

iii) Con condiciones iniciales dadas por: [S0 = 998, I0 = 1], y parámetros $\beta = 0.2$ $\gamma = 0.1$

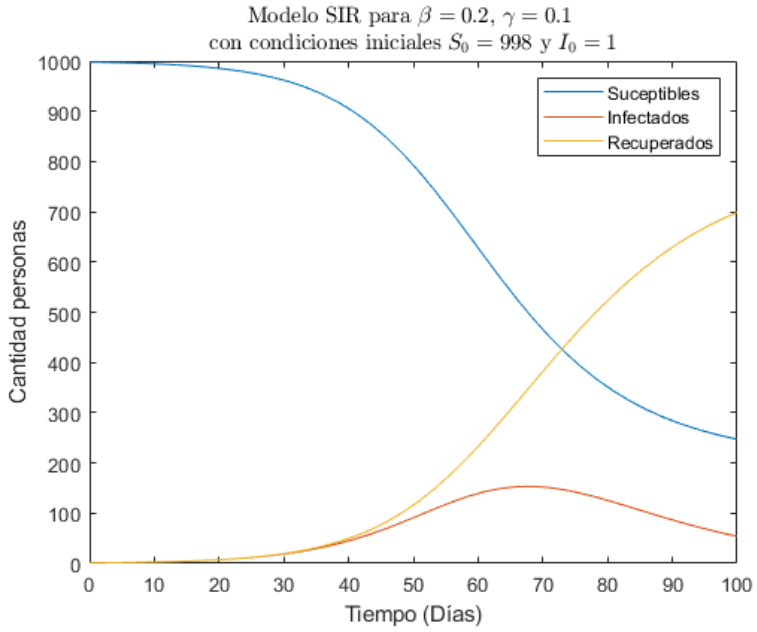

Figura 6: Desarrollo de la pandemia con $\beta = 0.2$ y $\gamma = 0.1$

Para esta ultima situación, aunque coincida que la tasa de infección sigue siendo más grande que la anterior, la diferencia recae en que es el doble que la de recuperación. Esto provoca el mismo comportamiento pero que se extienda o tarde aún más. Su pico llega hasta el día 68 y no es mayor a 170 personas. Por último podemos ver que los susceptibles jamás se eliminan y además los recuperados son mayores casi en todo el tiempo de la simulación.

4. Optimizar rutas de distribución de vacunas.

Se nos dieron tres diferentes casos para modelar las distancias más cortas a recorrer, además de conseguir sus valores y desviación estándar. También, se debía obtener que tan tardado era el proceso. Todo esto, siendo resuelto con procesos de optimización estocástica. Haciendo diez simulaciones para cada una en los tres diferentes casos, todo ello basándonos en el algoritmo de recocido simulado.

4.1. Caso 1

Para el primer caso, se nos dieron únicamente 4 ciudades, cada una con sus respectivas coordenadas x y y.
La siguiente gráfica representa el mapeo inicial de las ciudades sin optimización, es decir, con el camino que viene por *default* al mandar a graficar las ciudades.

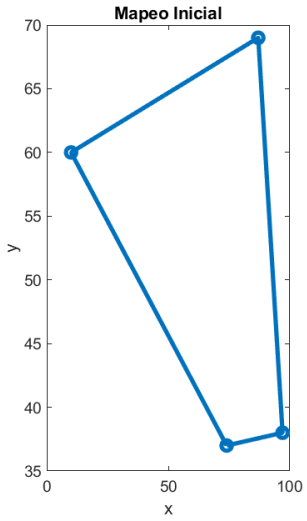

Figura 7: Mapeo de las ciudades sin optimizar.

Como se mencionó con anterioridad, se implementó el algoritmo de recocido simulado para obtener la mínima distancia requerida. La distancia antes de optimizar era de 201.1263, posterior al implementar el algoritmo, la distancia no sufrió cambios, conservando el mismo valor, esto debido a que el contexto de la figura permite que solo pueda existir esa posibilidad para hallar el mínimo de distancia.
Observemos la siguiente figura

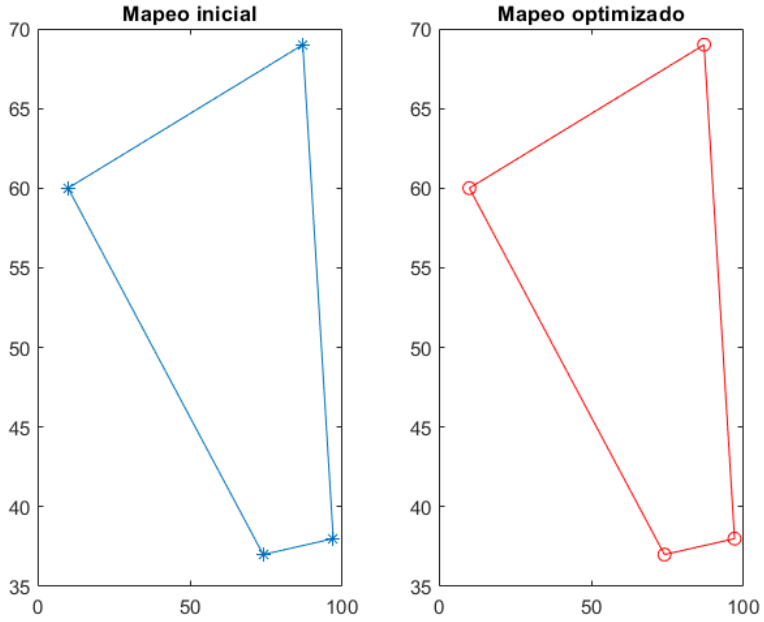

Figura 8: Comparación de ruta inicial contra ruta optimizada

Podemos notar que no existe diferencia alguna en el mapeo, esto es debido a que la simetría inicial, es al mismo tiempo, la distancia más corta.
La ruta quedó de la siguiente manera

$$Ruta = [2, 1, 4, 3]$$

De modo que empezando en la ciudad 2 hacia la ciudad 1 y así sucesivamente, obtendremos la distancia más corta.

Tiempos promedios Cada vez que se hacía una iteración, se guardaron los tiempos de ejecución en un vector, de modo que se obtuvieron los siguientes resultados con respecto al tiempo de ejecución del programa.

- Tiempo promedio: $t_{promedio} = 5.2750s$
- Desviación estándar: $\sigma_t = 0.1296s$

Podemos notar que por cada experimento realizado, existe un promedio de 5.2 segundos, con una desviación estándar de 0.1296 segundos, obviamente estos resultados pueden variar, dependiendo de especificaciones de cada computadora, pero sobre todo, por el número de ciudades, como se podrá observar en los siguientes casos.

Distancias obtenidas Como se mencionó con anterioridad, los valores de las distancias, permanecían igual, por lo que podríamos anticipar el resultado de el promedio de las distancias, sin embargo, sí se obtiene una desviación estándar, es muy pequeña, que incluso podríamos considerarlo como nula, pero esto igual nos dice que efectivamente, no existía mucha diferencia entre las distancias.

- Promedio de distancias: $dist_{promedio} = 201.1263$
- Desviación estádar distancias: $\sigma_{dist} = 2.9959 \times 10^{-14}$

4.2. Caso 2

Para el segundo caso, buscaremos la distancia más corta en un trayecto de 7 ciudades en sus respectivas coordenadas x y y con el uso de nuestro algoritmo para realizar un recocido simulado. Empezamos como anteriormente obteniendo un mapeo con una solución inicial del trayecto la cual queremos optimizar, esto nos ayuda a darnos cuenta de las posibles soluciones que deben salir y darnos idea de como esta trazado el mapa.

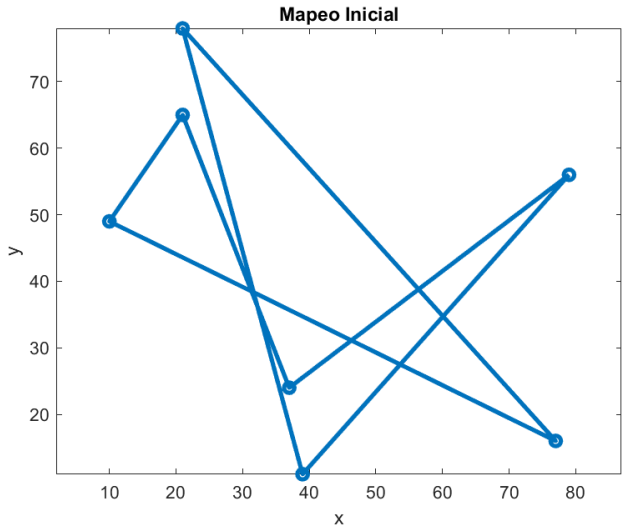

Figura 9: Ruta de la solución inicial propuesta

Esa primera propuesta nos propone un recorrido con distancia de **404.0455**.

Implementando nuestro algoritmo llegamos a una optimización donde la distancia recorrida ahora fue de **222.7759**, con lo que vemos la distancia optimizada a un 54 % y podemos tomar la misma como la distancia mínima que se puede recorrer entre las ciudades en tales coordenadas.

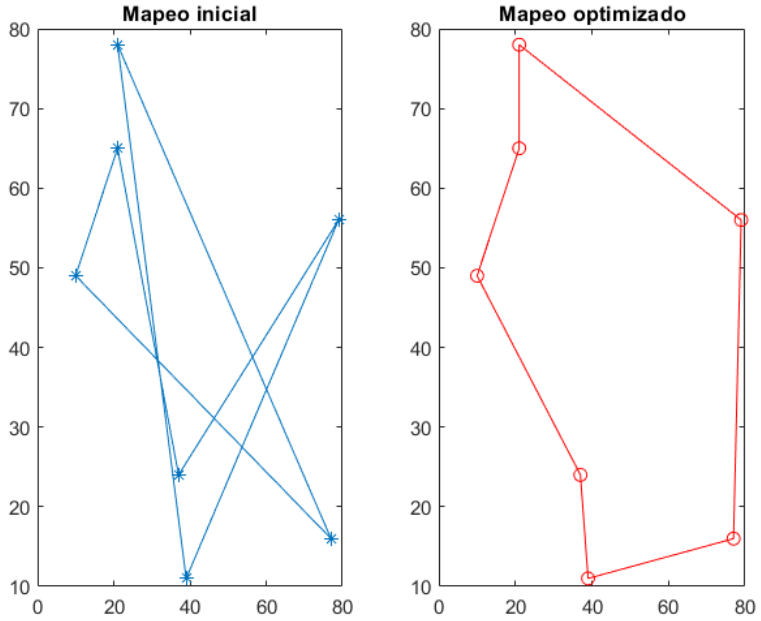

Figura 10: Ruta inicial vs optimizada

Con la figura confirmamos que es visible que la ruta que toma el programa es mejorada, y cuya ruta resultó ser:

$$Ruta = [1, 6, 3, 4, 2, 7, 5]$$

Tiempos promedios Con la almacenación de tiempos en cada iteración encontramos los siguientes promedios:

- Tiempo promedio: $t_{prom} = 7.057$ s
- Desviación estándar: $\sigma_t = 0.2099$

El tiempo ha aumentado ligeramente a comparación del Caso 1, esto debido a que la trayectoria trazada también ha sido más larga y por lo tanto implica mayor trabajo para el programa al buscar la mejor ruta.

Distancias Obtenidas

- Distancia promedio: $d_{prom} = 222.7759$
- Desviación estándar: $\sigma_d = 2.9959 \times 10^{-14}$

Esto nos indica que la distancia recorrida en cada iteración fue la misma y que la solución por lo tanto fue fácil de hallar. De igual manera, obtenemos una desviación estándar extremadamente pequeña, lo que puede ser tomada como nula y por ende aparenta ser la misma distancia siempre.

4.3. Caso 3

En este último caso, se nos proporcionaron 10 ciudades, cada una con sus respectivas coordenadas x y y, en este último caso, podemos anticipar con seguridad que será un poco más costoso computacionalmente hablando y algunas otras diferencias con respecto a los casos anteriores.
En un principio, mandamos a graficar las diferentes ciudades, unidas con cierta ruta predeterminada. Observese la siguiente figura:

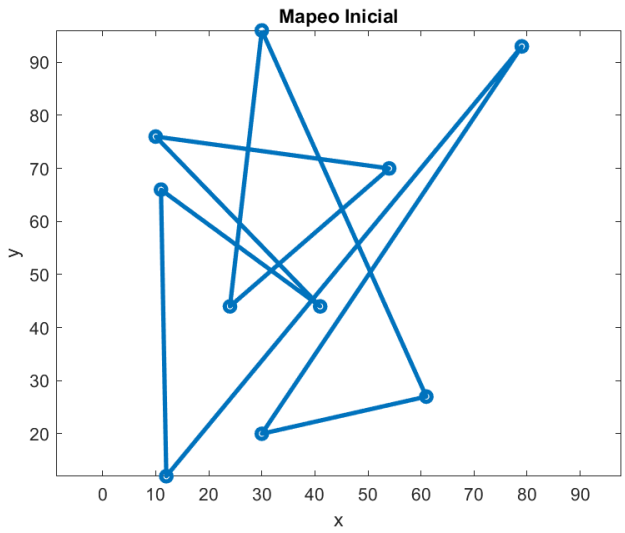

Figura 11: Mapeo de ruta de las ciudades (10) sin optimizar

En la figura anterior, podemos notar que la ruta representada está ordenada según el orden que se nos dió con una distancia inicial de **572.6796**.
Del mismo modo que se realizó en los casos anteriores, usamos el algoritmo de recocido simulado paara obtener la mínima distancia requerida.
Posterior a usar el algoritmo, obtuvimos una ditancia mínima de **287.9223**, estamos hablando de una reducción de un 50 % en las distancias.
En la siguiente figura, se muestra una comparativa entre la ruta inicial y la ruta optimizada con recocido simulado.

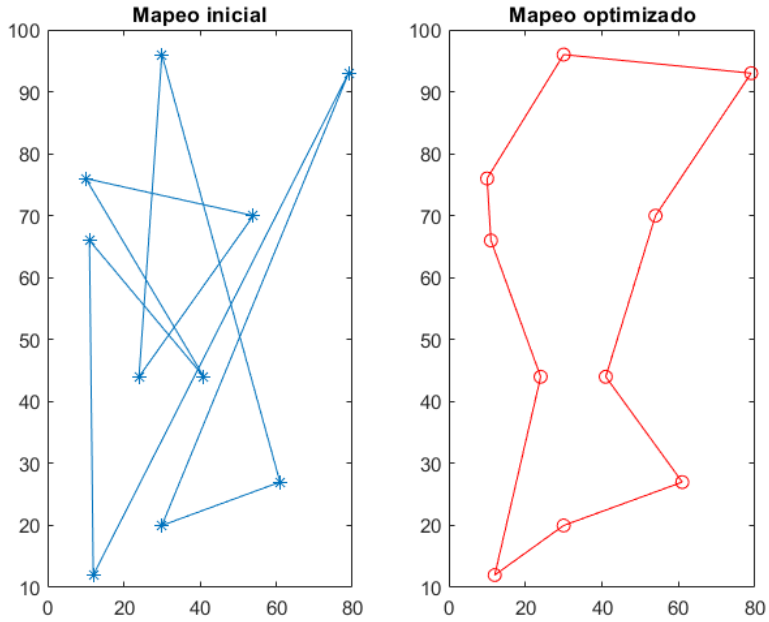

Figura 12: Comparación de ruta inicial contra ruta optimizada

Podemos notar que la diferencia del mapeo entre las gráficas es diferente, ya que la ruta ha sido optimizada y promete tener una menor distancia.
La ruta quedó con el siguiente orden.

$$Ruta = [8, 3, 5, 10, 9, 7, 2, 6, 4, 1]$$

Tiempos promedios Se realizó el mismo procedimiento que con los casos anteriores, almacenamos los tiempos de cada iteración y de este modo pudimos obtener un promedio y su desviación estándar.

- Tiempo promedio: $t_{promedio} = 9.0910s$
- Desviación estándar: $\sigma_t = 0.1146s$

Como era de esperarse, el tiempo entre cada iteración aumentó ya que teníamos más ciudades, lo que requiere de un mayor costo computacional, de la misma forma obtuvimos que la desviación estándar es de apróximadamente 0.1146 segundos, algo casi imperceptible.

Distancias obtenidas A diferencia de los casos anteriores, el promedio y la desviación estándar tuvieron cambios más significativos, notemos los siguientes resultados obtenidos:

- Promedio de distancias: $dist_{promedio} = 291.9167$
- Desviación estándar distancias: $\sigma_{dist} = 8.5818$

Los resultados anteriores, reflejan que aunque la distancia mínima (287.9223) está distanciada por casi 4 unidades de el promedio de distancias, existe una desviación estándar algo grande (8.5818), podemos deducir que esto se debe a los diferentes números aleatorios que se usan en cada simulación, haciendo que existan en ocasiones, diferencias más significativas que otras, por lo que esto puede aumentar o disminuir el promedio y la desviación estándar.

4.4. Código Implementado 'RECOCIDO SIMULADO'

```
clear;clf;
rng shuffle
% Cargar los datos
load('caso1.mat','-mat')
X = x;
Y = y;

% Guardar los datos en una matriz de "ciudades" (objetos de la clase ...
    "ciudad")
ciudades = formato(X,Y);
% [¬,X,Y] = inv_formato(ciudades);

N = length(X); % Cantidad de ciudades

% Visualizacion de la ruta inicial.
hold off
subplot(1,2,1)
plot([X' X(1)],[Y' Y(1)],'-o','linewidth',2.5)% Plot de los datos ...
    iniciales
title('Mapeo Inicial')
xlabel('x');ylabel('y')
axis fill

% Distancia de la ruta inicial
distancia_total(ciudades)

% Numero de repeticiones del proceso de optimizacion
n_repeticiones = 10;

% Alocamos los tiempos y distancias de cada proceso
tiempos = zeros(n_repeticiones,1);
mejores_distancias = zeros(n_repeticiones,1);
mejores_rutas = [];

for jj = 1:n_repeticiones
    % Solucion inicial
    Sol_act = ciudades;
```

```
36
37      % Temperatura
38      T0 = 0.5 * distancia_total(Sol_act);
39      T = T0;
40
41      % alpha
42      alph = 0.995;
43
44      % Ciclos dentro del while
45      L = 20;
46
47      % Temperatura final
48      Tf = 1E-9;
49
50      % Contador
51      cc = 1;
52
53      % Cantidad de vecinos para 2-opt
54      k = 2;
55
56      % Numero de iteraciones
57      n_iteraciones = 1000;
58
59      % Inicializar la mejor solucion
60      Sol_mejor = Sol_act;
61      distancia_mejor = distancia_total(Sol_mejor);
62
63      % Inicializamos la ciudad a analizar
64      id_i = 1; % Esta variable se usa si se quieren generar candidatos
65      % de forma no aleatoria con k2opt(Sol_act,id_i,k)
66
67
68      tic;
69
70      while true
71          for ii = 1:L
72
73              % Generamos vecinos
74              Sol_cand = rand2opt(Sol_act,k); % Aqu  se pueden ...
                    utilizar varios metodos
75
76
77              Δ = distancia_total(Sol_cand) - ...
78                      distancia_total(Sol_act) ;
79
80              % Generamos una x aleatoria (volado) y comparamos con ...
                    Boltzman
81              x = rand;
82              Pboltz = exp(-Δ/T);
83
84              if Δ<0 || x<Pboltz
85                  Sol_act = Sol_cand;
86              end
87          end
88
89          % Almacenamos el minimo en un vector
90          Cmin(cc) = distancia_total(Sol_act);
91
```

```
92          % Guardamos el mejor de todos los resultados
93          if distancia_total(Sol_act) < distancia_mejor
94              Sol_mejor = Sol_act;
95              distancia_mejor = distancia_total(Sol_mejor);
96          end
97
98          % Enfriamos
99          T = alph*T;
100         Temp(cc) = T;
101
102         cc = cc+1;
103         if cc+1== n_iteraciones
104             break
105         end
106     end
107     tiempos(jj) = toc;
108     mejores_distancias(jj) = distancia_mejor;
109     mejores_rutas = [mejores_rutas;Sol_mejor];
110 end
111
112 % Mejor distancia de este caso
113 min(mejores_distancias)
114 % Mejor ruta de este caso
115 ruta_mas_corta = find(mejores_distancias == min(mejores_distancias));
116 mejor_ruta = mejores_rutas(ruta_mas_corta(1),:);
117 mejor_ruta.id
118 [¬,X2,Y2] = inv_formato(mejor_ruta);
119 clf;
120 subplot(1,2,1)
121 plot([X' X(1)],[Y' Y(1)],'-*')
122 title('Mapeo inicial')
123 axis auto
124 hold on
125 subplot(1,2,2)
126 plot([X2 X2(1)],[Y2 Y2(1)],'-or')
127 title('Mapeo optimizado')
128 axis auto
129 hold off
130 % Promedio de los tiempos
131 mean(tiempos)
132 % Desviacion estandar de los tiempos
133 std(tiempos)
134 % Promedido de las distancias
135 mean(mejores_distancias)
136 % Desviaci n estandar de las distancias
137 std(mejores_distancias)
```

5. Suavizado de funciones por medio de redes neuronales

5.1. Introducción

Las redes neuronales son conjuntos de objetos interconectados que actúan como procesadores de información. De manera similar a las células neuronales, este conjunto de objetos matemáticos se basa en un sistema de pesos y activaciones [2] que en conjunto llegan a procesar información de una manera muy efectiva haciéndola una de las tecnologías mas disruptivas del siglo XXI.

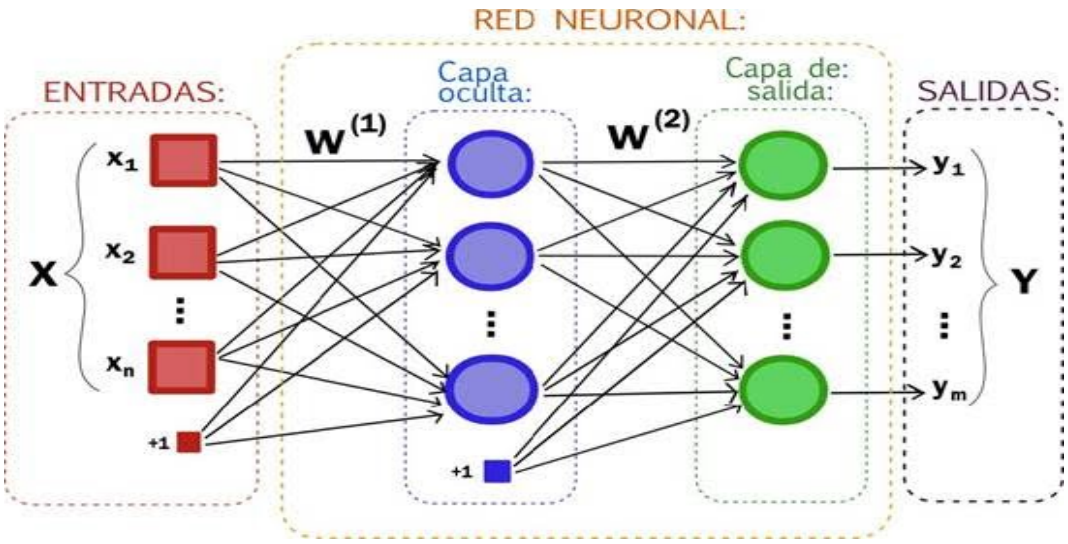

Figura 13: Estructura básica de una red neuronal.

Una de las muchas aplicaciones que las redes neuronales nos brindan es el suavizado de curvas. Por medio de un algoritmo matemático que simula el aprendizaje neuronal llamado **aprendizaje supervisado** se puede suavizar cualquier curva.

5.2. De datos a modelos

En una epidemia como la que estamos viviendo en este momento, lo que se tienen son datos contaminados. Esto en el sentido de que los datos tienen errores. Los principales errores que se encuentran en un conjunto de datos son los .ºutliers"[3] y el hecho de que la función resultante no va a ser suave.

Para simular ambos aspectos de esta naturaleza imperfecta de los datos, se resolvió un sistema deterministico SIR y se añidio el ruido usando la siguiente función:

```
1 function [f_noise] = addnoise(fun, percent)
```

[2] Se hace referencia a una activación donde el 1 representa activación y el 0 ausencia de activación.
[3] Valores que se midieron erróneamente.

```
%ADDNOISE is a function that adds noise to a discrete vector acording to
%some percent.

%General parameters
len = length(fun);
rng('shuffle')

f_noise = zeros(len,1);    %Preallocate

for j = 1:len
    pd = makedist("Uniform",-1,1);
    f_noise(j) = fun(j) + random(pd) .* percent .* fun(j);

end

end
```

Al alimentar dicha función con una función discretizada y un porcentaje de ruido percent, la misma regresa una función contaminada con ruido de acuerdo al porcentaje dado. Los resultados se observan en las figuras 15,16,17.

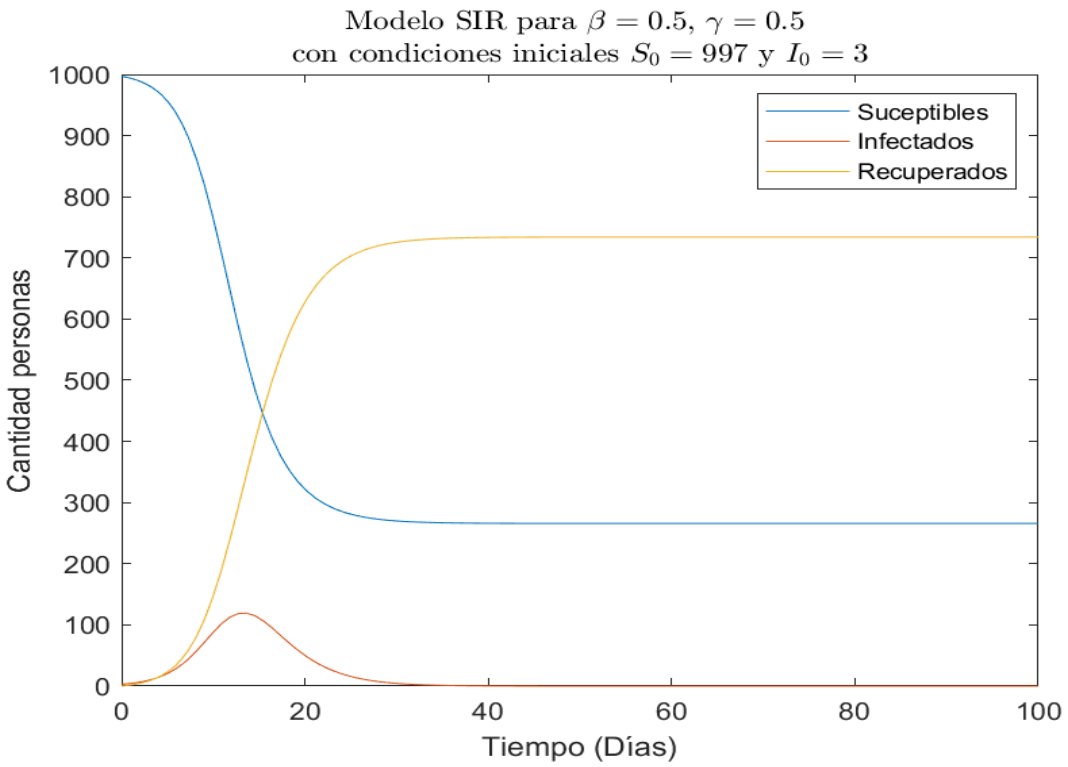

Figura 14: Solución al SIR determinista

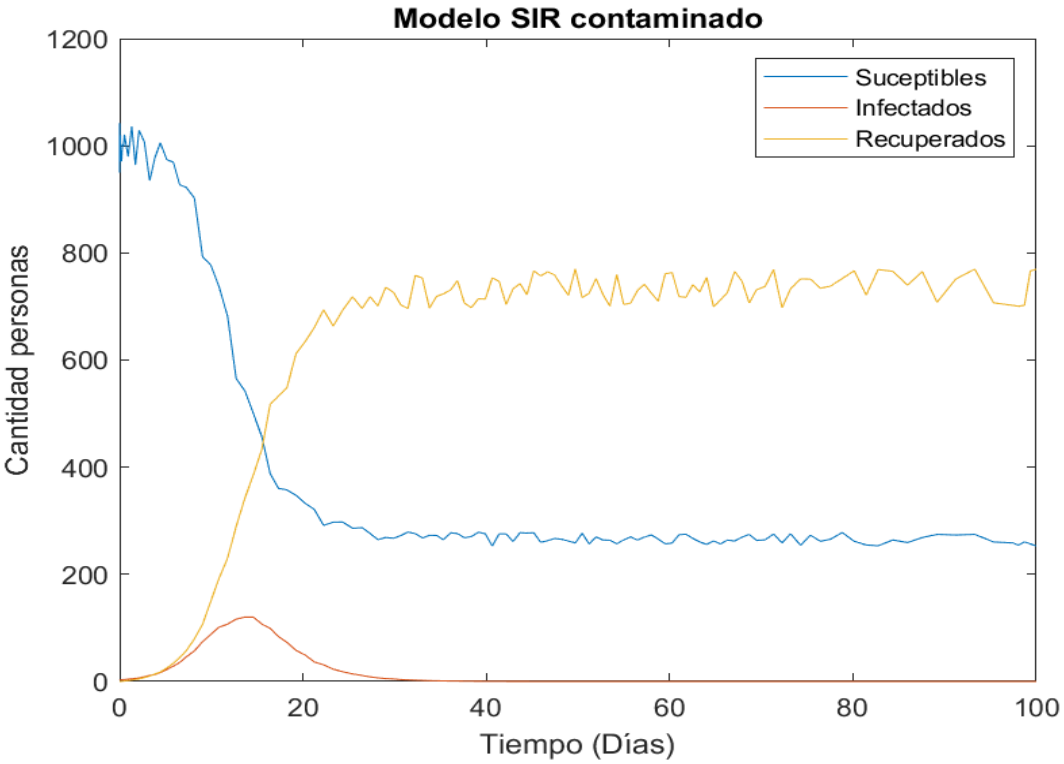

Figura 15: SIR determinista contaminado 5 %

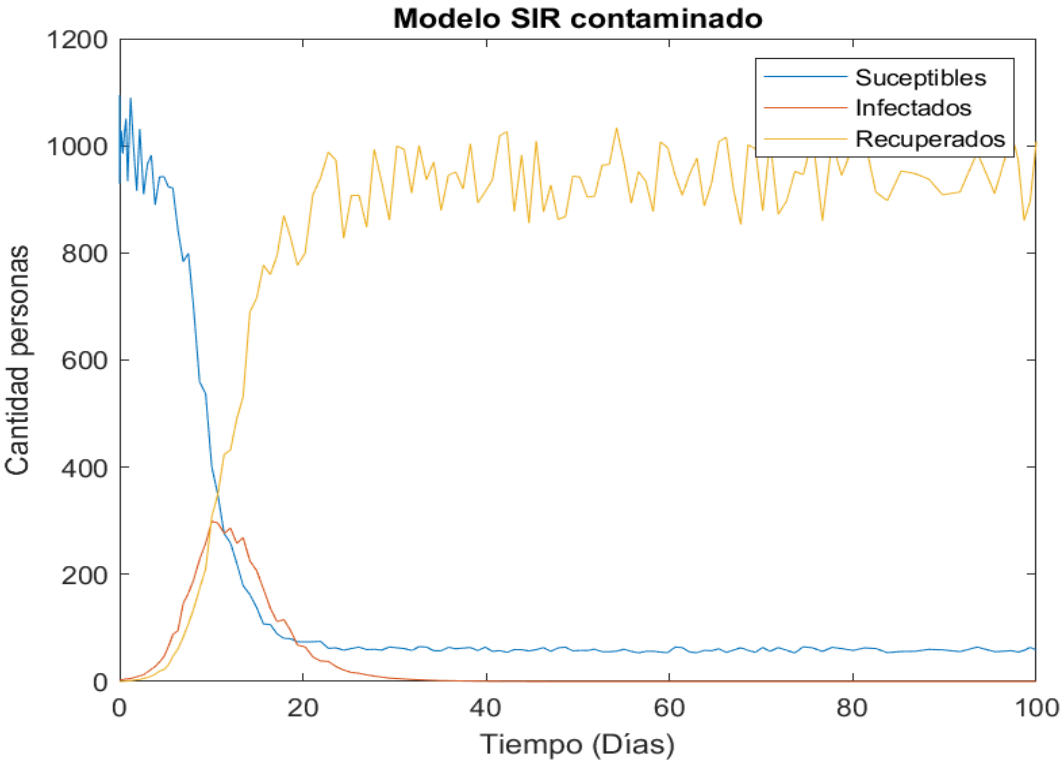

Figura 16: SIR determinista contaminado 10 %

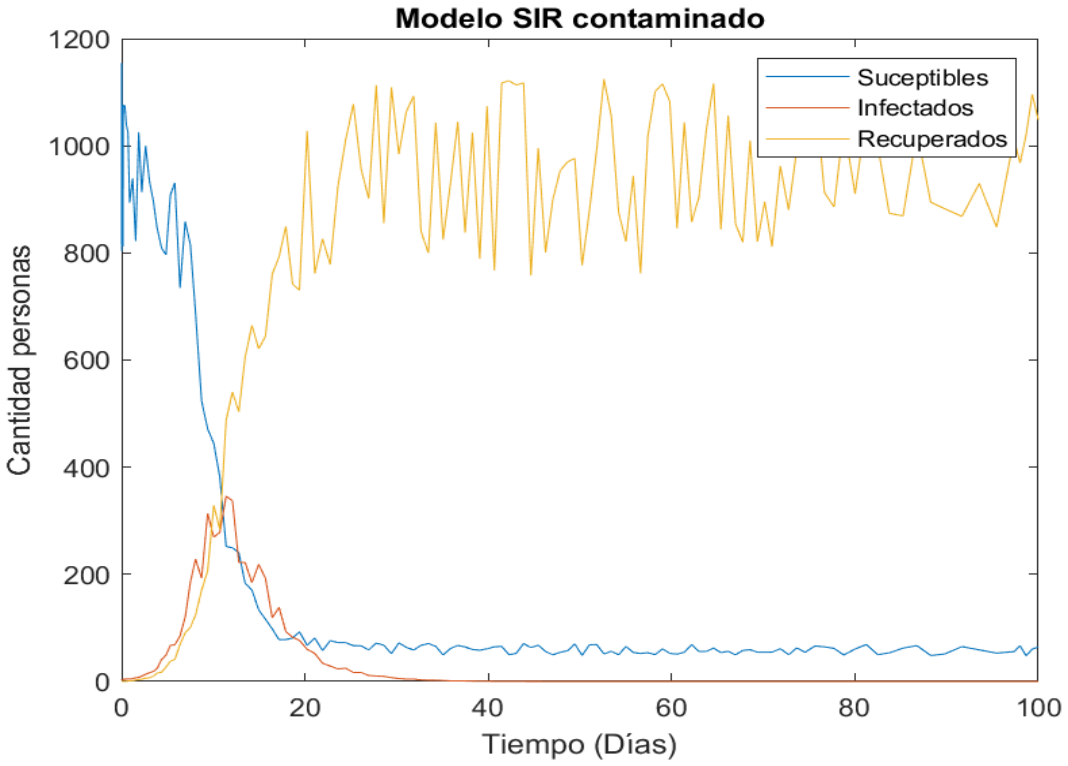

Figura 17: SIR determinista contaminado 20 %

5.3. Implementando redes neuronales

Usando la siguiente función, se hace un suavizado de las funciones que componen el modelo SIR contaminado con ruido.

```
function [fun_smooth]=neuralnetwork_smoothfun(fn,t)
%NEURALNETWORK_SMOOTHFUN is a function that smooths out a given function
%using neural networks. It requires the NN toolbox and has 3 input
%arguments: the discrete function to smooth out, a target function to ...
    train
%the neural network and the domain vector of the functions.

% Primero, vamos a definir el tipo de red
CapasOcultas = [8 8];  % Neuronas en cada capa oculta
Entrenamiento = 'trainlm';  % Gradiente
net = feedforwardnet(CapasOcultas, Entrenamiento);

% veamos la red
view(net);

% Definimos las funciones de activacion
net.layers{[1 2]}.transferFcn = 'logsig';

% Epocas
net.trainParam.epochs = 500;

% Datos de entrernamiento
rng('shuffle');
pdtest = makedist('Uniform',-2,2);
```

```
24
25 % Entrenamos
26 net = train(net, t', fn');
27
28 % Verificamos
29 fun_smooth = net(t');
30 pp = perform(net, fun_smooth, fn');
31
32 end
```

Para generar las siguientes figuras, a la función anterior se le introduce el vector del tiempo t como input a la red neuronal y la función S_{ruido} como datos de entrenamiento. El resultado obtenido cuando la red neuronal previamente entrenada se le alimenta con el vector del tiempo, es el siguiente:

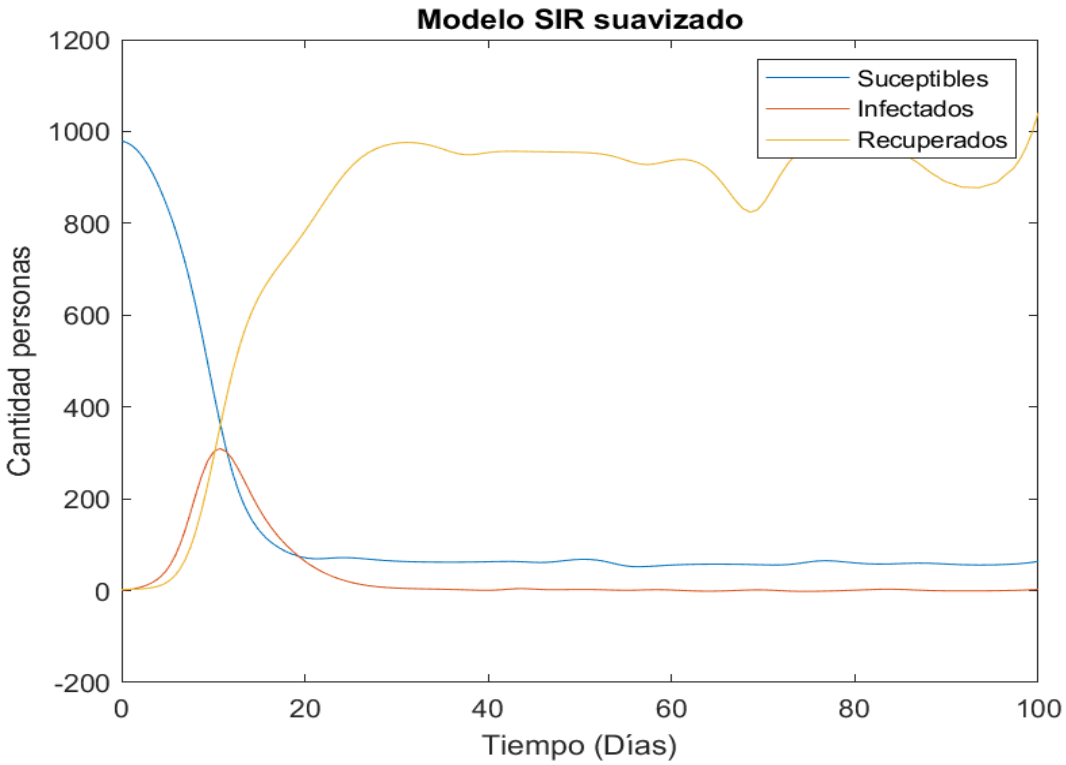

Figura 18: Suavizado al SIR determinista por medio de redes neuronales

Donde la neurona tiene las siguientes características:

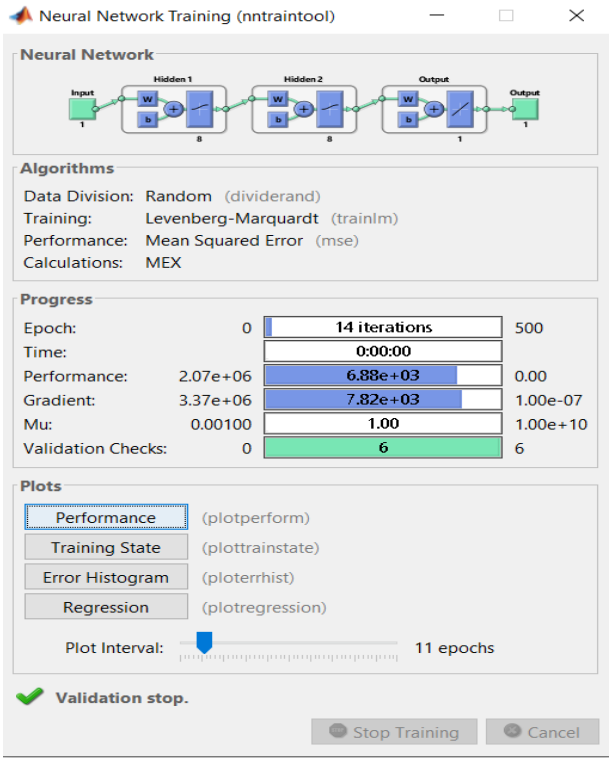

Figura 19: Características de la red neuronal

Los errores de dicho suavizado se calculan con la formula del error cuadrático. Donde la matriz de $[SIR]$ es el parámetro objetivo y la matriz $[S_{suavizada}I_{suavizada}R_{suavizada}]$ es lo que usamos como parámetro observado.

Error	S	I	R
20 %	53.37	37.9	681.77
10 %	24.72	39.9	590.15
5 %	17.32	4.47	128.44

Si se observa los resultados de la tabla anterior, se nota que el error cuadrado (Error) del vector que describe a los removidos (R) es el que tiene mayor valor. Esto se debe a que el ruido introducido es proporcional a la magnitud del vector en un dado punto. Es por esto que hay mas ruido en el vector de de removidos y con ende se esperaría un mayor error.

Finalmente, es importante introducir una discusión sobre el resultado de la red neuronal obtenida. En este ejemplo, la red no se ajusta perfectamente a los datos de entrenamiento ;como se observa en la figura 18. Esto es una buena señal ya que una red con un error muy bajo puede resultar en un caso de sobre entrenamiento y esto hace que la red no se pueda ajustar a otros ejemplos.

6. Introducción Fase 3

En la siguiente fase, podremos darnos cuenta de la importancia de las redes neuronales aplicadas para poder suavizar las expresiones del modelo SIR, en general, esto es posible gracias a el entrenamiento que le damos a las neuronas para tener un resultado bastante

parecido al esperado. Posterior a la suavización de las curvas también vemos la aplicación de los algoritmos genéticos en la búsqueda de los valores para gamma y beta, el cual realiza una selección, cruza y mutación de para dar con los datos que mejor se ajusten a la función fitness.
Cabe mencionar que igual podemos mejorar estos resultados con un nuevo concepto, llamado *moving average* o en español, *Media Móvil*

6.1. Media móvil

La media movil es un cálculo utilizado para analizar conjuntos de datos para crear series de promedios. En otras palabras, las medias móviles es una lista de diversos datos, donde cada dato de la lista contiene un promedio de un subconjunto de los datos originales. Existen 4 tipos principales de media móvil.

- Media movil simple
- Media móvil central
- Media móvil ponderada
- Media móvil exponencial

6.1.1. Media móvil simple

Una media móvil simple es una media aritmética de n datos anteriores. En este caso, cuandto n sea más grande, mayor será la influencia de los datos anteriores, mientras que con la selección de una n más pequeña, se tendrán en cuenta datos más recientes para nuestra predicción.
Esto es de suma importancia, ya que podemos decir que la elección de nuestra n influye en gran manera en la predicción. Podemos decir que el uso de una n pequeña es recomendado para aquellos casos donde existan variaciones significativas en los datos de un periodo a otro.

6.1.2. Media móvil central

En este caso podemos decir que no solo utilizamos los datos anteriores, sino también usamos los datos posteriores a aquel del cual se quiere obtener la media.

6.1.3. Media móvil ponderada

Esta es una media que se multiplica por ciertos factores, que le dan cierto peso a determinados datos.Esta es una técnica que desarrolla y mejora las aplicaciones de la media móvil simple. Al hacer este tipo de proceso, se superan los inconvenientes que ofrece la técnica de media móvil simple, ya que este nos permitirá decidir si darle mayor importancia a datos más antiguos o más recientes.

6.1.4. Media móvil exponencial

La media móvil exponencial, como su nombre lo dice, es una media ponderada exponencialmente. Básicamente, esta es la media aritmética de los n valores anteriores con factores de ponderación que decrecen de manera exponencial.

7. Datos de una epidemia contaminados con ruido.

7.1. Uso de una Red Neuronal

Se inicia la primera fase del reto con un código de matlab donde se carga el archivo que se pide y que permite leer los datos proporcionados para las curvas del modelo SIR: susceptibles, infectados y removidos. Así se observa que las curvas contienen mucho ruido el cual no facilita la manipulación de los datos para generar R0. Por lo tanto, en vista de las circunstancias, es necesario realizar una red neuronal al principio que logre suavizar las curvas hasta que sean lo suficientemente manejables para que al meter los datos en un algoritmo genético los resultados salgan claros y efectivos.

El código comienza por cambiar los vectores columnas que nos fueron dados a vectores filas, ya que de esta manera podremos insertarlos en las redes neuronales. Posteriormente, se escribe el toolbox de matlab con el cual se realizara el entrenamiento de las neuronas. En el toolbox se le da valor al número de capas ocultas con cierto número de neuronas, en este caso nuestro equipo ha optado por una sola capa de 5 neuronas, que en teoría suena a poco, pero que en realidad evita que el sistema reciba, lo que es conocido como un overfeeding, con el cual las redes neuronales en lugar de optimizar la información comienza a simular a la misma y entonces el ruido que se quisiera eliminar seguirá ahí; tampoco se busca un underfeeding con lo cual se perdería la trayectoria de los datos completamente la cual, muchas veces termina siendo una recta. Con la configuración elegida, el programa nos regresará curvas que asimilan la trayectoria de la curva original, pero que se puede observar mucho más lisa y clara, algo parecido al modelo SIR determístico, justo como se busca para el siguiente paso. Otros aspectos importantes de las redes neuronales que se usaron fue que el tipo de entrenamiento definido como gradiente para buscar la optimización del sistema, es decir, que busca minimizarla, se definió que la red neuronal tuviera una propagación hacia adelante y que la función de activación fuera sigmoide (de modo que es continua) y por último definimos que tendría 1000 épocas para una mayor efectividad.

El siguiente paso es definir una variable con la cual se evaluara la red como valor x, y ya con esto realizar un entrenamiento para cada una de nuestras variables (susceptibles, infectados y removidos). Finalmente con las redes entrenadas ya suavizar los datos renombrando las 3 variables con sus respectivas redes entrenadas evaluadas en x.

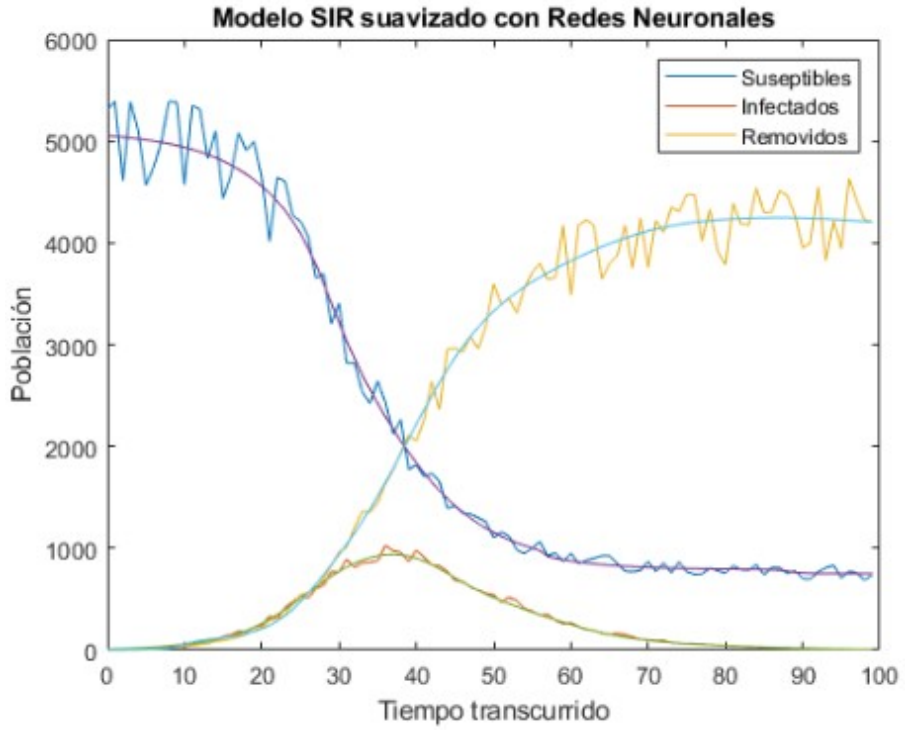

Figura 20: Datos del modelo SIR antes y después de ser suavizados

Se puede tomar de la gráfica que los susceptibles comienzan con una población arriba de 5000 personas, éste número va disminuyendo conforme pasan los días y la infección avanza. Se puede ver que el punto más alto de contagio se alcanza alrededor de los 40 días y podemos inferir que la taza de remoción será mayor a la taza de transmisión debido a que la curva de removidos incrementa rápidamente, acabando así con la infección antes de que todos los susceptibles fueran contagiados, de modo que poco menos de 1000 personas se quedan en el estado de susceptibles al final del trayecto de 100 días.

7.2. Uso de Algoritmos Genéticos para obtener R_0

Ahora que se tiene una mejor modelación de los datos, se busca generar una función objetivo, la cual es evaluada en una nueva función que evita que sus valores sean negativos, lo que finalemnte resultara en la función fitness que se evaluará en un algoritmo genético, el cual seleccionará los valores más óptimos para γ y β bajo un modelo de selección natural. Una vez obtenidos γ y β, se prosigue a la búsqueda de R_0 ya que:

$$R_0 = \frac{\beta}{\gamma} \tag{4}$$

Este resultado indica el tipo de enfermedad con el que la población del problema ésta lidiando, pero tratando con redes neuronales y algoritmos genéticos sabemos que los resultados, cada vez que se corra el programa, pueden variar debido a que éstos dependen del nivel de aprendizaje que adquiere el programa al ejecutarse. Va a variar tanto en tiempo como en valores que se generan para β y γ, así como la calidad del suavizado. Por lo tanto, nuestro equipo decidió lidiar con este problema buscando que el propio programa

realice 30 ciclos del algoritmo genético de forma que al final se cuenta con 30 valores de R_0 y a los cuales les podemos sacar una media que nos dará un valor mucho mejor estimado. Esto, gracias a que cuenta con la comparación de resultados de mayor error y con los de menor error, a diferencia de correr el código una sola vez sin saber si el resultado tiene un alto grado de errores o no.

Dicho lo anterior y teniendo en cuenta el sistema SIR que tenemos. Donde:

$$N = S + I + R \tag{5}$$

Aquí se considera la N como una constante, lo que significa que siempre habrá la misma población solo que repartida en estos 3 grupos. Entonces, los cambios en cada uno se evalúan de la siguiente manera:

$$\frac{dS}{dt} = -\beta\frac{SI}{N} \tag{6}$$

$$\frac{dI}{dt} = \beta\frac{SI}{N} - \gamma I \tag{7}$$

$$\frac{dI}{dt} = \gamma I \tag{8}$$

Y estas se relacionan como:

$$\frac{dS}{dt} + \frac{dI}{dt} + \frac{dI}{dt} = 0 \tag{9}$$

Por lo que nuestra función fitness consta de una sumatoria de sus 3 cambios en los grupos, la cual se puede interpretar como una integral pero para esta situación se definirá de la siguiente manera:

$$f = (\frac{dS}{dt} + \beta\frac{SI}{N})^2 + (\frac{dI}{dt} - \beta\frac{SI}{N} + \gamma I)^2 + (\frac{dR}{dt} - \gamma I)^2 \tag{10}$$

Por lo que, el algoritmo genético buscará reducir su valor de la misma o que sea lo más cercano a cero por medio de la razón de infección y de remoción. El único problema que tenemos es que desconocemos el valor de todas las derivadas en los cambios, ya que gracias a la red de neuronas conocemos S,I,R y N, además de dejar a β y γ sin definir ya que serán los elementos a evaluar. Al ser un método estocástico, buscamos obtener sus derivadas de los mismos vectores con el método de diferencias centrales hacia adelante, definida como:

$$\frac{dX}{dt} = \frac{X(n+1) - X(n)}{\Delta t} \tag{11}$$

Siendo X cualquiera de las diferencias que deseamos obtener. Al ser solamente 100 valores por los 100 días, se utiliza la diferencia central hacia atrás para el último valor. Una vez con estos valores, se puede definir bien la función fitness de una forma vectorizada para reducir el tiempo de ejecución en el código. Posteriormente se realizan las 30 interaciones previamente explicadas donde se designa las características del algoritmo genético y por

último se rescatan los últimos valores de las razones por ser las más adecuadas y así recuperar R_0, el cual nos ayudará a saber que tipo de enfermedad se estaba propagando. Como último paso se calcula el promedio y la desviación estándar del vector creado de diferentes R_0 y se decide.

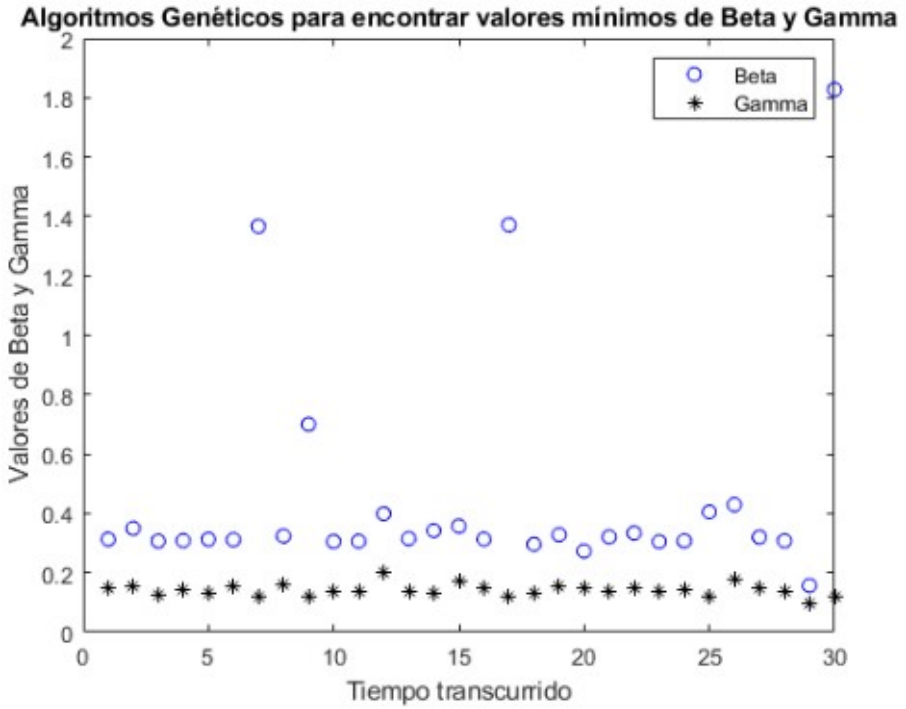

Figura 21: Resultado de las mejores β's y γ's a lo largo de las 30 interaciones

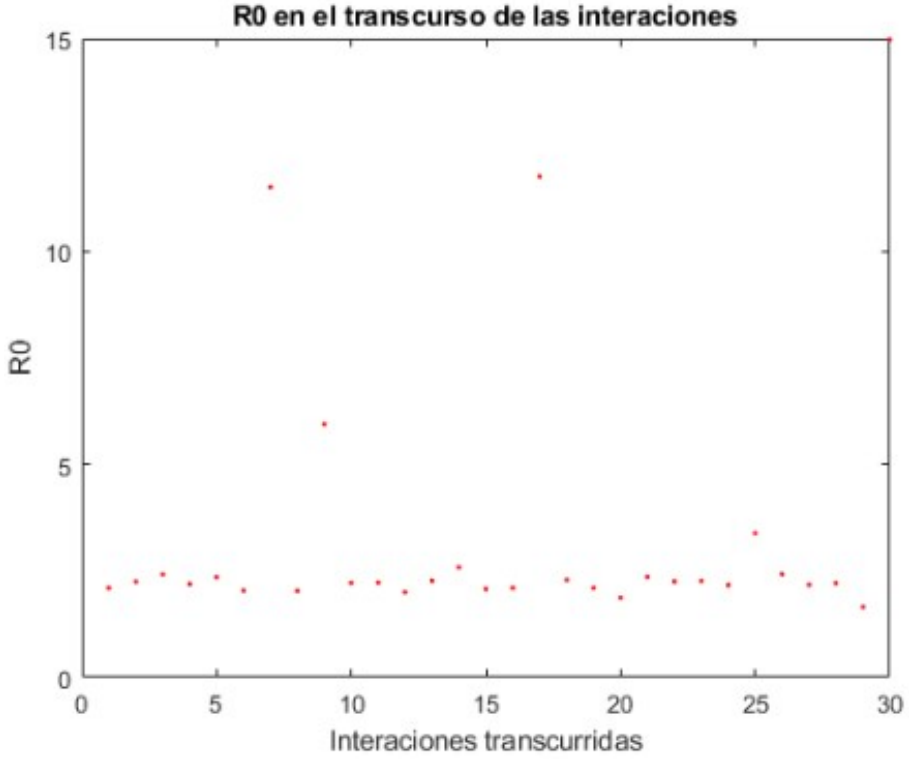

Figura 22: Resultado de R_0 a lo largo de las 30 interaciones

Finalmente, obtuvimos un resultado de $R_0 = 3.4074$ con una desviación estándar de $std = 3.2898$, lo cual, comparado con la tabla previamente entregada, podemos decir que la enfermedad transmitida en dicho lugar sería COVID-19.

R_0	Enfermedad
1 - 2	Gripe porcina
2 - 5	Covid-19
5 - 7	Paperas
7 - 12	Varicela
12 - 18	Sarampión

7.3. Código

```
clear; clc; close all;

% Cargar el archivo
load('reto1.mat');
s = S';
i = I';
r = R';

% Toolbox de redes neuronales
CapasOcultas = [5];  % Neuronas en cada capa oculta
Entrenamiento = 'trainlm';  % Gradiente
net = feedforwardnet(CapasOcultas, Entrenamiento);
net.layers{1}.transferFcn = 'logsig';
net.trainParam.epochs = 1000;
rng('shuffle');
pd = makedist('Uniform',0,1);
xx = 0:1:length(s)-1 ; %random(pd,1,100);

% Train
nets=train(net,xx,s);
neti=train(net,xx,i);
neter=train(net,xx,r);

% Resultados
ss=nets(xx);
ii=neti(xx);
rr=neter(xx);

% Despliegue suavizado
plot(xx,s,xx,i,xx,r)
hold on
plot(xx,ss,xx,ii,xx,rr)
hold off
title('Modelo SIR suavizado con Redes Neuronales')
legend({'Suseptibles','Infectados','Removidos'})
xlabel('Tiempo transcurrido')
ylabel('Poblaci n')

% Diferencias centrales hacia adelante
```

```
for n = 2:length(t)-1
    ds(n) = ss(n+1)-ss(n);
    di(n) = ii(n+1)-ii(n);
    dr(n) = rr(n+1)-rr(n);
end

% Ultimo valor de las diferencias
ds(n+1) = ss(n)-ss(n-1);
di(n+1) = ii(n)-ii(n-1);
dr(n+1) = rr(n)-rr(n-1);

% Funci n fitness
ff = @(x)  (ds + x(:,1).*ss.*ii/N).^2 + (di - (x(:,1).*ss.*ii)/N + ...
    x(:,2).*ii).^2 + (dr - x(:,2).*ii).^2;

% Interaciones
for kk = 1:30
    rng shuffle
    %opciones de optimizaci n
    options = optimoptions('gamultiobj',...%'PlotFcn',@gaplotbestf,...
              'MaxGenerations',1000,...
              'MaxStallGenerations',500,...
              'PopulationSize',length(ds),...
              "FunctionTolerance",1E-15,...
              'UseVectorized',true);
    solution = gamultiobj(ff,2,[],[],[],[],[0  0],[],[],options);

    beta=solution(:,1);
    gamma=solution(:,2);

    % Valores rescatados
    b(kk) = beta(end);
    g(kk) = gamma(end);
    r0(kk)= beta(end)/gamma(end);
end

% Comportamiento de beta y gamma a lo largo de las interaciones
plot([1:kk],b,'bo',[1:kk],g,'k*')
title('Algoritmos Gen ticos para encontrar valores m nimos de Beta ...
    y Gamma')
legend({'Beta','Gamma'})
xlabel('Tiempo transcurrido')
ylabel('Valores de Beta y Gamma')

% Comportamiento de R0
plot([1:kk],r0,'r.')
title('R0 en el transcurso de las interaciones')
xlabel('Interaciones transcurridas')
ylabel('R0')

% Promedio y desviacion estandar
pr0 = mean(r0)
sr0 = std(r0)
```

8. Caso a) de modelación de una epidemia con agentes.

Ahora bien, una buena forma de conocer si nuestro método para obtener β y γ es comparándolo con una simulación basada en agentes hecha en *NetLogo* que arroja datos bastante cercanos a los reales. En este caso se utilizó el modelo *epiDEM Basic* como base para realizar una simulación SIR, haciendo las modificaciones que se muestran a continuación:

8.1. Cambio de parámetros iniciales

La primera modificación hecha, fue necesaria para establecer los parámetros iniciales de este caso que fueron: [initial people 500, infection chance 2, recovery chance 10 average recovery time 50]. La modificación fue echa ya que los "sliders" no permitían insertar estos parámetros.

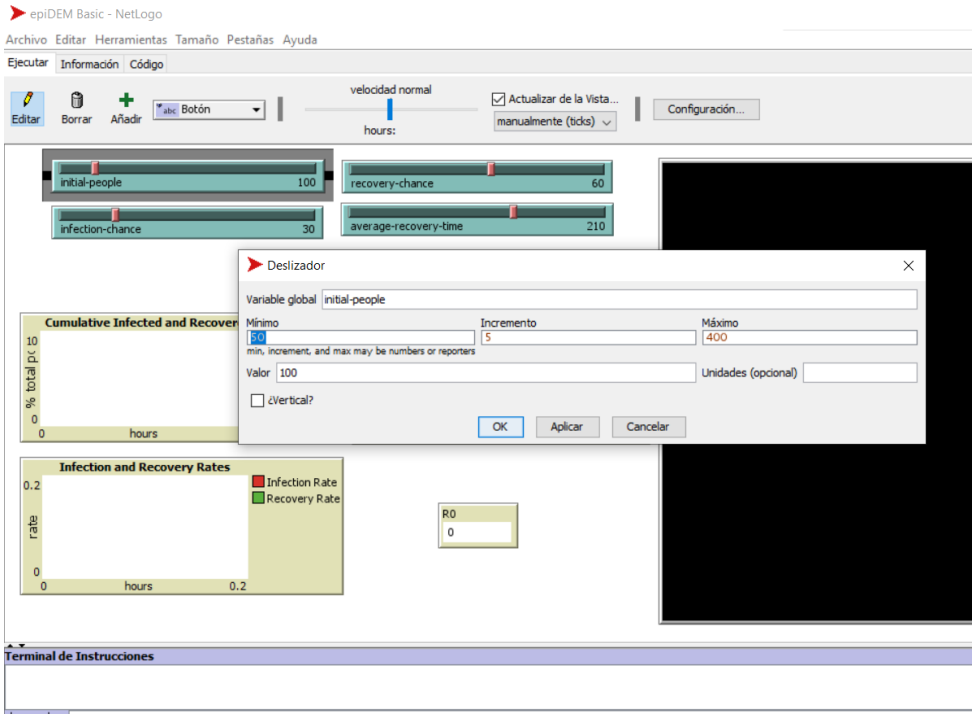

Figura 23: Captura de pantalla del modelo epiDEM Basic con su configuración inicial.

8.2. Gráfica del modelo SIR

Para obtener los datos del modelo SIR, es decir, la cantidad de personas susceptibles, infectadas y recuperadas según el tiempo, se insertó un gráfico que recopilara estos datos como se muestra en la siguiente figura:

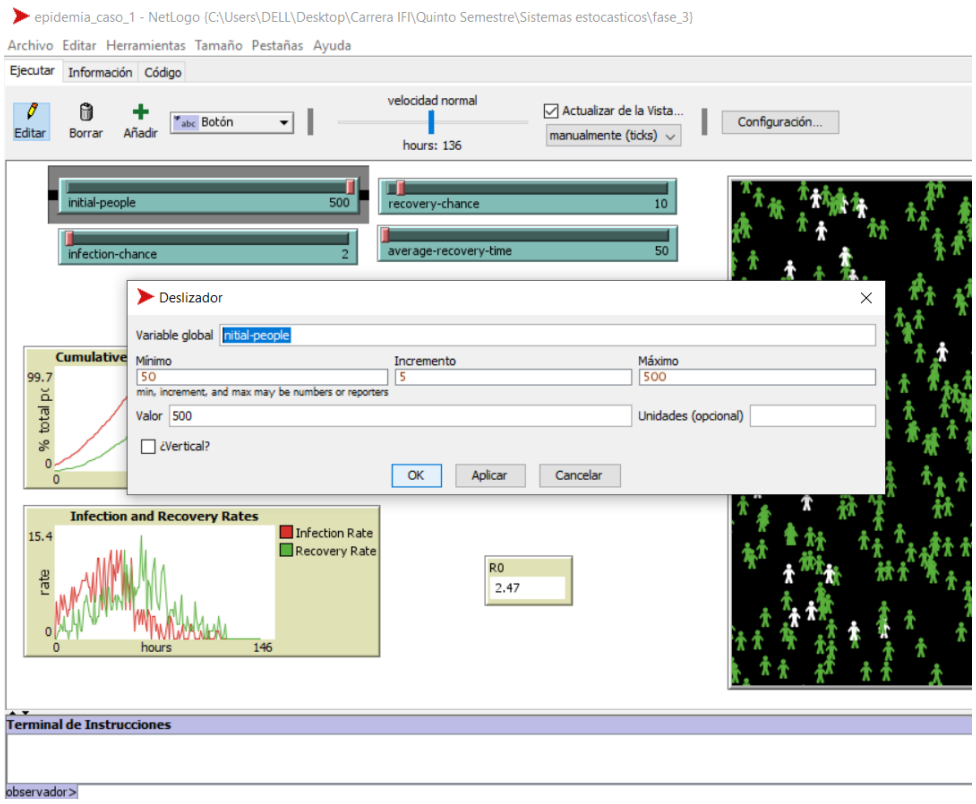

Figura 24: Captura de pantalla del modelo epiDEM Basic con los parámetros modificados de acorde al caso a modelar.

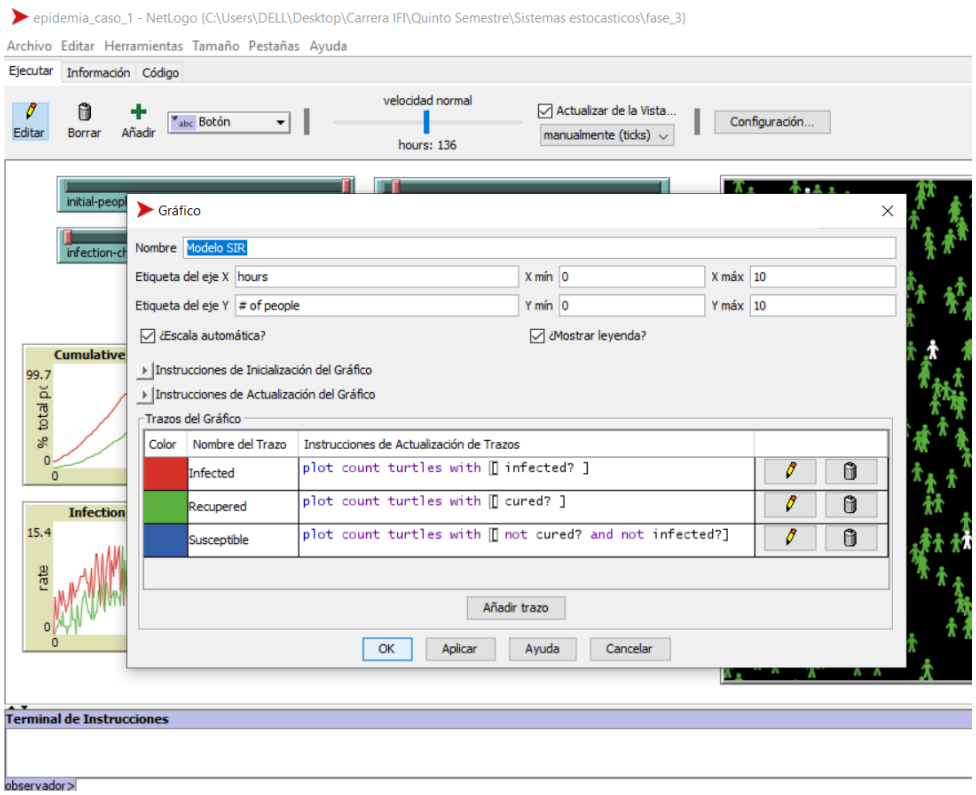

Figura 25: Captura de pantalla de la configuración del gráfico que permitió la captura de datos del modelo SIR.

8.3. Resultados obtenidos

Finalmente de correr la simulación en NetLogo, se obtuvieron los siguientes datos, los cual da la primera impresión de corresponder de forma efectiva a los de una epidemia modelada con el modelo SIR, donde la R_0 calculada corresponde a $R_0 = 2.47$, que correspondería con un caso de Covid-19.

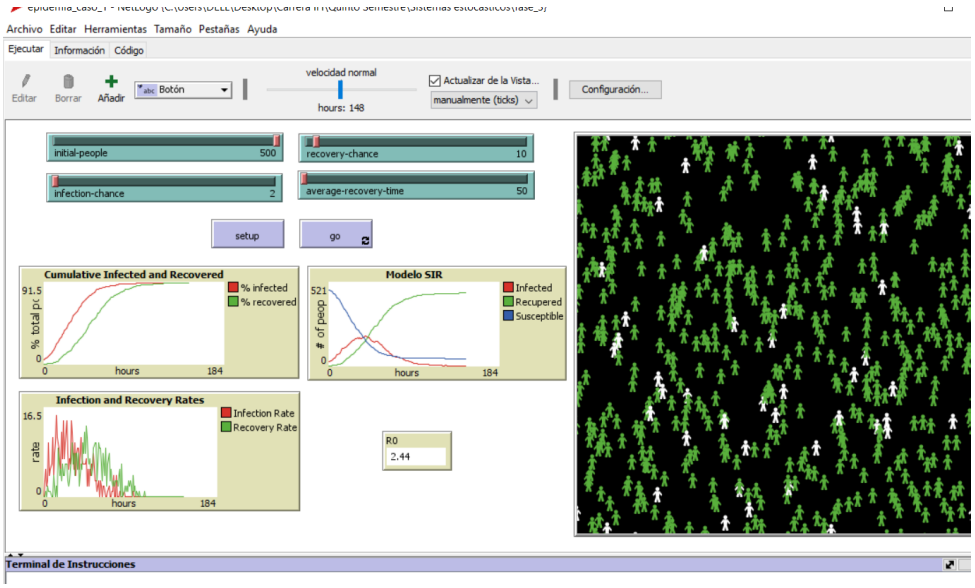

Figura 26: Captura de pantalla de los resultados de la simulación de una epidemia en NetLogo.

8.4. Exportar los datos a MATLAB y R_0

Para esto, NetLogo tiene una opción en la pestaña de archivos llamada “Exportar” donde simplemente puedes seleccionar exportar los datos de una gráfica a un archivo .csv, el cuál se puede importar facilmente a MATLAB con el comando readmatrix(). Con el código de la siguiente subsección se obtuvo por medio de un algoritmo genético el valor de R_0 que fue aproximadamente de 2.7047 según el resultado de MATLAB. Esta diferencia se puede deber a que el proceso para obtener R_0 en NetLogo es muy distinto, pero ambos resultados predicen la misma enfermedad, así que son aceptables.

8.5. Código

```
clear;
datos = readmatrix('datos_caso_1.csv');

t = datos(8:end,1)';
ii = datos(8:end,2)';
rr = datos(8:end,6)';
ss = datos(8:end,10)';
plot(t,ss,t,ii,t,rr)
legend({'Susceptible','Infected','Recupered'})

```

```
for n = 2:length(t)-1
    ds(n) = ss(n+1)-ss(n);
    di(n) = ii(n+1)-ii(n);
    dr(n) = rr(n+1)-rr(n);
end

ds(n+1) = ss(n)-ss(n-1);
di(n+1) = ii(n)-ii(n-1);
dr(n+1) = rr(n)-rr(n-1);
N = 500;

ff = @(x)  sum((ds + x(:,1).*ss.*ii/N).^2 + (di - (x(:,1).*ss.*ii)/N ...
    + x(:,2).*ii).^2 + (dr - x(:,2).*ii).^2);

for kk = 1:10
    rng shuffle
    %opciones de optimizaci n
%     options = optimoptions('gamultiobj',... %'PlotFcn',@gaplotbestf,...
%               'MaxGenerations',1000,...
%               'MaxStallGenerations',500,...
%               'PopulationSize',length(ds),...
%               "FunctionTolerance",1E-15,...
%               'UseVectorized',true);
    [solution funval(kk)] = ga(ff,2,[],[],[],[],[0  0]);

    beta(kk)=solution(1);
    gamma(kk)=solution(2);

    r0(kk)= beta(end)/gamma(end);
end
```

9. Caso b) de modelación de una epidemia con agentes.

Este caso se trata masomenos de lo mismo que el anterior, es decir, establecimos los parámetros iniciales de la simulación en NetLogo: [initial people 500, infection chance 2, recovery chance 90, average recovery time 50]. Luego de esto se corrió la simulación y se extrajeron los datos de la misma forma, y este proceso se repitió 10 veces con los mismos parámetros, así obteniendo 10 archivos nombrados datos_caso_2_n.csv, donde n es el número de repetición de la simulación. De esta forma se facilitó el importar los datos en MATLAB y realizar el procesamiento de los datos para obtener R_0 de forma automática con un for loop.

9.1. Resultados

El R_0 obtenido de todo este proceso fue en promedio de 1.5748 con una desviación estándar de 0.0923, lo que sorprende bastante, ya que significa que el promedio calculado de R_0 de hecho es una aproximación bastante precisa según el algoritmo genético implementado, a diferencia de NetLogo, que da resultados bastante variados, pero aún así se mantienen alrededor de este valor, lo que sigue estando dentro de la categoría de enfermedad de

Gripe porcina.

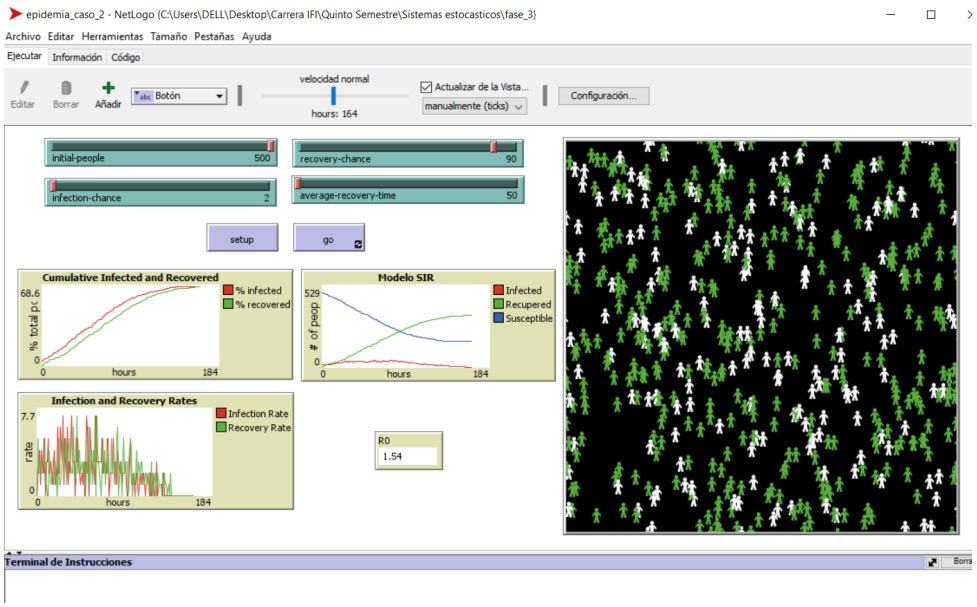

Figura 27: Captura de pantalla de los resultados una de las simulación de una epidemia en NetLogo en este caso.

9.2. Código

```
clear;
for jj = 1:10
    numero_archivo = char(string(jj));
    nombre_completo = ['datos_caso_2_' numero_archivo '.csv'];
    datos = readmatrix(nombre_completo);
    t = datos(8:end,1)';
    ii = datos(8:end,2)';
    rr = datos(8:end,6)';
    ss = datos(8:end,10)';
    ds = 0;
    di = 0;
    dr = 0;
    for n = 2:length(t)-1
        ds(n) = ss(n+1)-ss(n);
        di(n) = ii(n+1)-ii(n);
        dr(n) = rr(n+1)-rr(n);
    end

    ds(n+1) = ss(n)-ss(n-1);
    di(n+1) = ii(n)-ii(n-1);
    dr(n+1) = rr(n)-rr(n-1);
    N = 500;

    ff = @(x)  sum((ds + x(:,1).*ss.*ii/N).^2 + (di - ...
        (x(:,1).*ss.*ii)/N + x(:,2).*ii).^2 + (dr - x(:,2).*ii).^2);


    rng shuffle
```

```
28      %opciones de optimizaci n
29  %     options = optimoptions('gamultiobj',... %'PlotFcn',@gaplotbestf,...
30  %                'MaxGenerations',1000,...
31  %                'MaxStallGenerations',500,...
32  %                'PopulationSize',length(ds),...
33  %                "FunctionTolerance",1E-15,...
34  %                'UseVectorized',true);
35      [solution funval(jj)] = ga(ff,2,[],[],[],[],[0  0]);
36
37      beta(jj)=solution(1);
38      gamma(jj)=solution(2);
39
40      r0(jj)= beta/gamma;
41
42
43  end
44  mean(r0)
45  std(r0)
```

10. Datos de una epidemia con modelacion estoc astica.

10.1. Planteamiento

A continuación, se presenta la situación donde se nos dieron unos datos que provenían de un modelo estocástico del tipo Doob-Gillspie. Estos datos no están contaminados por lo que es necesario hacer un filtrado de los mismos para poder tener una mejor aproximación de los parámetros esperados.
A continuación se presenta una gráfica del modelo SIR con datos provenientes del modelado estocástico: Doob-Gillspie.

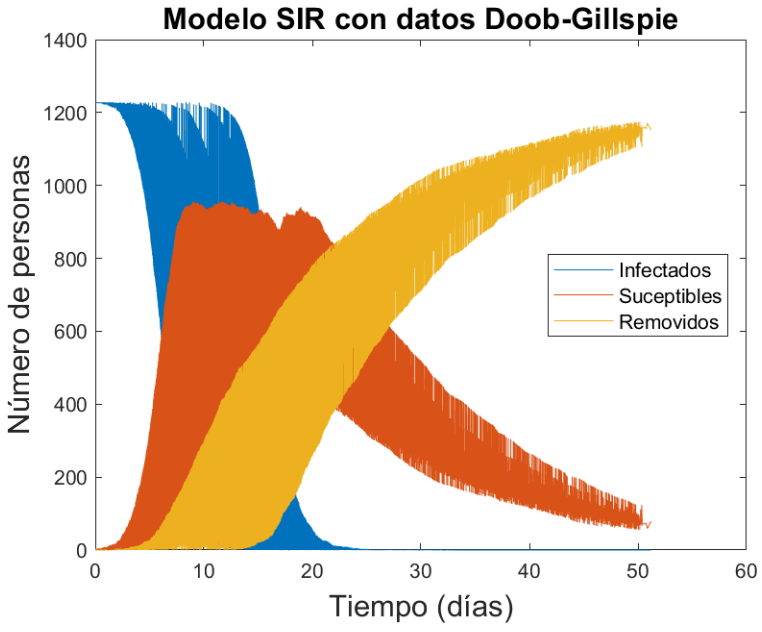

Figura 28: Datos con Doob-Gillspie

Para poder quitar el ruido que se puede ver en la figura (28) se utilizó el método de media móvil central, donde se toman valores tanto de atrás como de adelante, en este caso usamos un $n = 1000$, recordemos que la mejor aproximación se obtiene al tener un beun valor de n por lo que después de probar diversos parámetros, vimos que con esta última n seleccionada, teníamos una buena aproximación. Llegamos a probar una n mucho mayor, pero nos dimos cuenta que la función en vez de verse cuadrada, se empezaba a tornar a una figura un poco más cuadrada.

Para limpiar los datos ilustrados en la figura anterior se hace una serie de técnicas de suavizado. En particular, para este ejemplo se propone usar el metodo moving average y el suavizado por medio de redes neuronales.

La técnica de limpieza de datos conocida como *moving average* consiste en tomar el promedio de los n datos mas cercanos al punto de interés y posteriormente, asignar el valor del promedio a ese punto. Si se hace este calculo para cada punto de la función se obtienen los siguientes resultados:

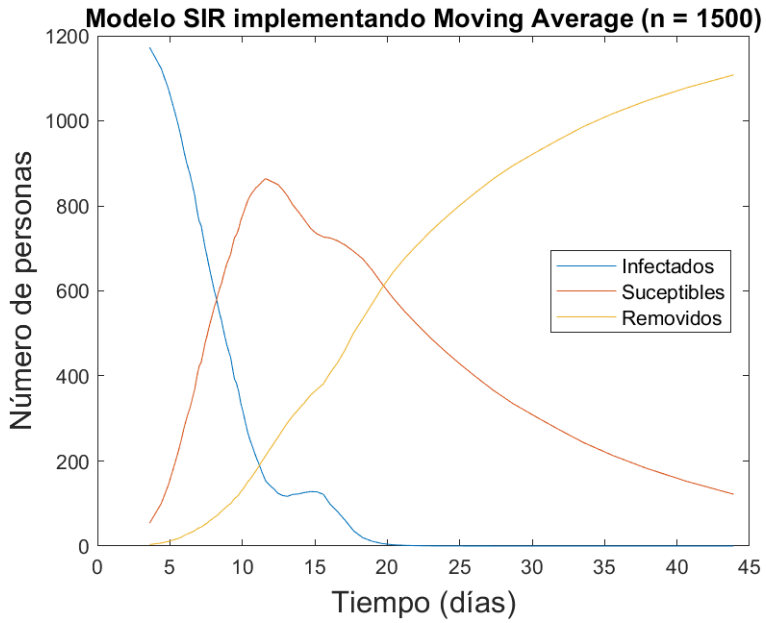

Figura 29: Modelo SIR implementando Moving Average Centrado

La técnica de suavizado por medio de redes neuronales consiste en entrenar a una red neuronal con una cantidad *critica* de números. Critica, en el sentido de que no puede ser mucho ni tampoco pocos datos. Al alimentarle justo una cantidad óptima, la red entrena para regresar puntos clave de la función (que corresponden a el dominio *critico* de entrenamiento), suavizando todos los valores que quedan en los intervalos. A continuación, se muestra dicho resultado. Nota que las figuras que se van exponiendo, respetan un orden consecutivo [4].

[4]Es decir, primero se usa moving average y posteriormente se suaviza en una segunda iteracion usando redes neuronales.

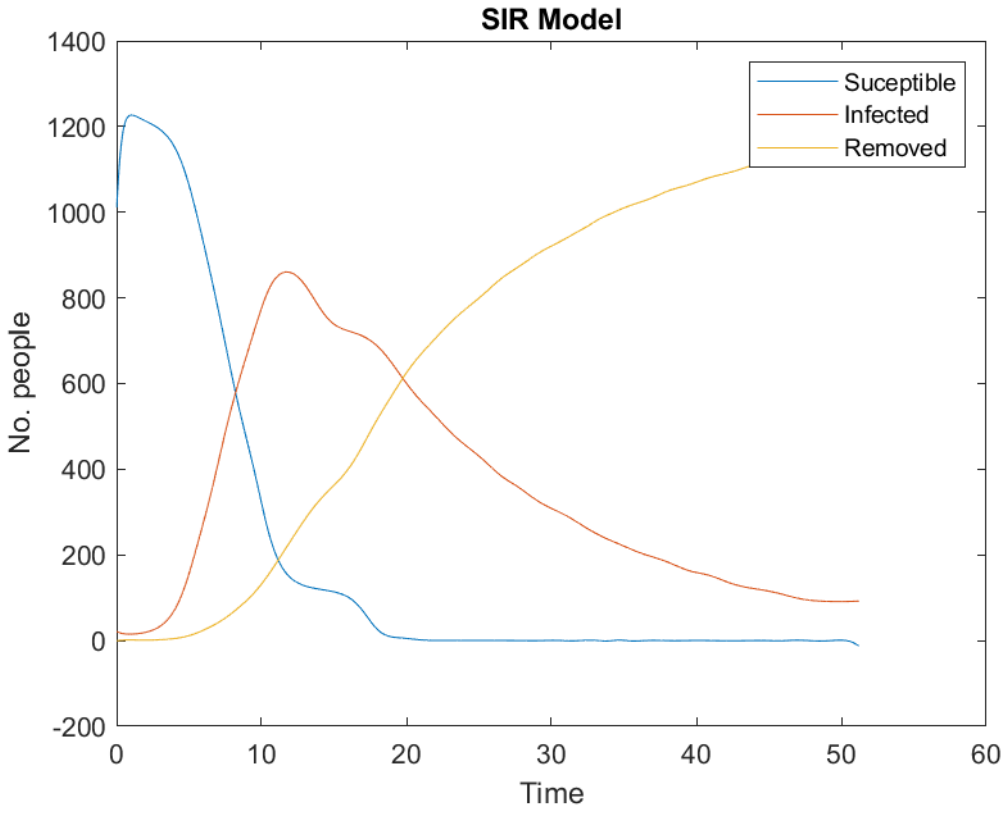

Figura 30: Suavizado con Redes Neuronales

. Como se ha expuesto previamente, el uso de algoritmos genéticos es muy útil para encontrar mínimos de funciones. Para este caso, recuerda que se tienen que encontrar los parámetros β y γ. De tal forma de que podemos definir una función de costo que dependa de ambos parámetros y además tenga una solución cuando se iguala a cero. Dicha función, se plantea de la siguiente manera.

$$\Sigma[(S'(t) + SI/N)^2 + (I'(t) - (SI)/N + \gamma I)^2 + (R'(t) - \gamma I)^2] = 0 \tag{12}$$

Nota que los términos de la función de costo son las ecuaciones del modelo SIR igualadas a cero. Es por esto, que se pueden plantear en la forma de un error cuadrático e igual a cero. Una vez planteada dicha ecuación, se usa la técnica de algoritmos genéticos para encontrar el par de parámetros: β y γ que minimicen el valor de dicha función de costo. El resultado obtenido es el siguiente:

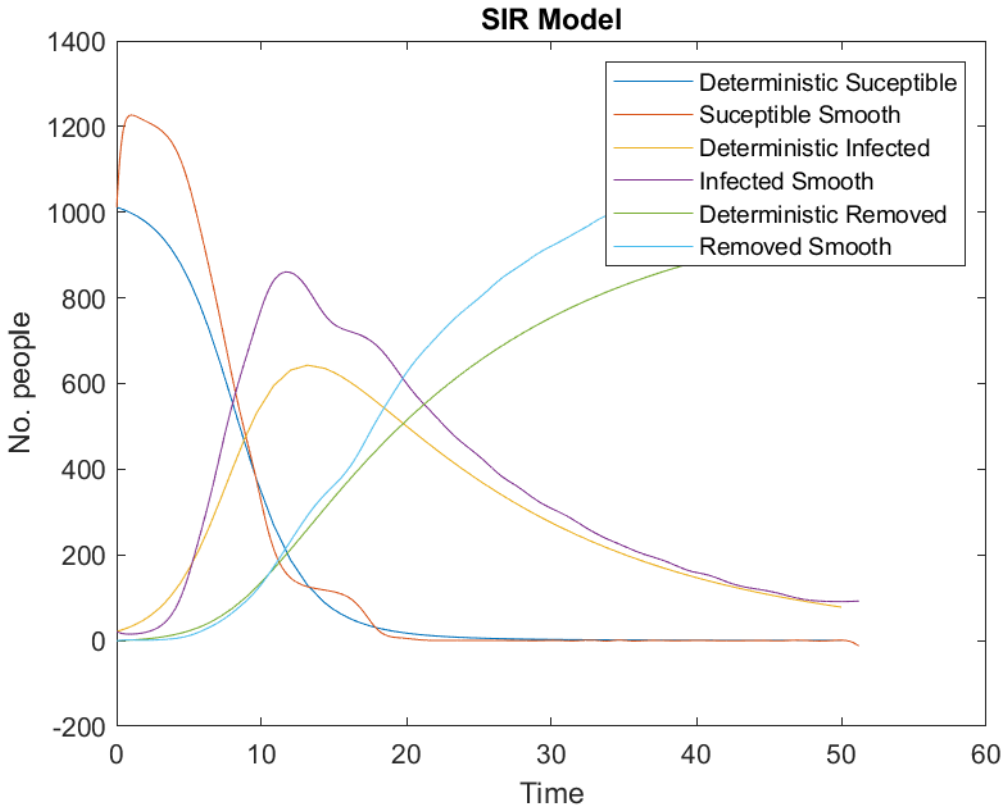

Figura 31: SIR determinista con parámetros del AG & el SIR suavizado.

10.1.1. Predicciones R_0

En nuestro caso, se obtuvieron los siguientes parámetros para β, γ y R_0.

$$\beta = 0.6158 \qquad \gamma = 0.0634 \qquad \mathbb{R}_0 = 9.70806$$

Se hizo una breve investigación para ver en que enfermedad caía nuestro coeficiente R_0, donde se encontró la siguiente tabla. [Eisenberg, 2021]

Enfermedad	Número de reproducción R_0
Ebola (2014)	1,51 a 2.53
H1N1 Influenza (2009)	1.46 a 1.48
Gripe Estacional	0.9 a 2.1
Sarampión	12 a 18
MERS	$\sim$1
Polio	5 a 7
SARS	<1 a 2.75
Viruela	5 a 7
SARS -Cov-2 (causa COVID-19)	1.5 a 3.5

Podemos ver que el valor de R_0 no se encuentra dentro de la tabla, pero podemos suponer que esto se debe a ciertos parámetros dentro de nuestra programación de nuestro algoritmo genético, por lo que si corremos la simulación diferentes ocasiones y probando diferentes parámetros podremos llegar a un resultado más cercano al esperado.

Comparando nuestros resultados con los de la otra tabla mostrada anteriormente en la fase 2, podemos concluir que nuestro R_0 pertenece a la enfermedad de varicela

11. Reflexiones

11.1. Eduardo Barroso

Para esta Fase, pudimos relacionarnos más con la evolución y desarrollo de los métodos estocásticos para la resolución del modelo SIR y del ejercicio para la competencia en clase. Pudimos llevar a un mayor nivel el entendimiento con los programas y los métodos utilizados para la simulación. Todos los tipos de conceptos, además de los paquetes incluidos en Matlab facilitaron su realización, después de la gran indagación realizada y desarrollada por nuestra cuenta.

11.2. David Camela

Con el desarrollo de esta última entrega, he aprendido diversos algoritmos que son de suma importancia para poder encontrar mínimos o predecir otro tipo de cosas como con el uso de redes neuronales o algoritmos genéticos. Se me han hecho super interesantes cada uno de ellos, también con el desarrollo de las entregas, considero que he podido refrescar un poco mi programación en Matlab y he podido llegar a optimizar algunos de los algoritmos que nos dan para que puedan funcionar mejor y de manera más eficiente. Finalmente espero que en un futuro pueda estudiar más a fondo todo este tipo de algoritmos que considero tienen infinidad de aplicaciones en diversas industrias, se me hace un tópico en el que aún hay muchas cosas que están por descubrirse.

11.3. Izael Rascón

Me gustó bastante esta fase del reto debido a que al trabajar con método metahurísticos, no es tan necesario meterse a matemáticas y uno puedo diseñar y modificar el método de una forma bastante artesanal, lo que hace que algo complejo se le pueda dar una aproximación más sencilla. Disfruté en especial las redes neuronales, tenía muchas ganas de llevar un pequeño cursito y no me arrepiento.

11.4. Karina Abboud

El reto fue demasiado sencillo para nuestras dotadas mentes... ah no se crea profe, si nos costo sudor y lagrimas post-examen. Fue una gran experiencia para aplicar los conocimientos de la clase y tener la oportunidad de comparar sus utilidades en casos que pudieran ser trabajos reales y no solo académicos. Aprendí sobre la importancia de saber diferenciar los métodos estócasticos de los determinísticos así como las funciones para cada uno; también adquirí conocimiento sobre como realizar números .[a]leatorios.[en] programas de computadora, aunque realmente en la tecnología no existe tal cosa como la aleatoridad, sino que todos los valores que tomamos como pseudoaleatorios pertenecen a

complejos patrones pertenecientes a distribuciones probabilísticas o funciones que igualmente complican la predicción del siguiente resultado, pues crear con nuestras propias manos el valor del desconocimiento es aún un poder que no poseemos.

En el reto tomamos una probadita de lo que pueden ser estos sistemas estocásticos, desde realizar lo más simple como fue la Caminata Aleatoria hasta realizar modelos que aprendan por su cuenta a reconocer datos y optimizarlos como hicimos con las redes neuronales y algoritmos genéticos. Inclusive aprendimos a optimizar usando recocido simulado y buscar los mínimos de una función. Los temas vistos fueron de gran interés para mi ya que nunca había tenido que hacer algo parecido con dichos métodos, y me pareció increíble como basándonos en el comportamiento de la naturaleza podemos hallar tantas respuestas y las que faltan por hallar, increíble como pensar que gracias a la tecnología podemos saber como funciona la naturaleza y terminar copiando eso que se aprende para generar más tecnología. Son temas tan bellos como complejos, pues llevar a cabo los procesos paso a paso toma su tiempo, pero cuando salen los resultados correctos uno se lleva buen sabor de boca. En conclusión, los métodos estocásticos nos abrieron las puertas a nuevos métodos de programación que ya no se limitan únicamente a datos conocidos, sino que pueden partir de una semillita para predecir el comportamiento a futuro, por eso son tan importantes en casos como son las epidemias donde nos basamos mucho en la probabilidad y el movimiento de variables cuyas trayectorias no son claras.

11.5. Vladimir Padilla

Dentro de todo lo desarrollado en el curso hasta esta fase ha sido muy retador e interesante. La idea que los diferentes proceso que hemos visto estén inspirados en la naturaleza o en el desarrollo de otros materiales para su propia aplicación, ayuda extremadamente a que estos se puedan entender de un a forma mucho más sencillos. Además, la conexión que tiene con los conceptos computacionales, realmente permite ver ese pensamiento computacional utilizado para su implementación y el porque facilita y/o optimiza su sucesión. Cabe mencionar que era divertido el analizar su concepto y su desarrollo a mano pero se agradece en extremo que, por ejemplo, tanto los sistemas de neuronas como los algoritmos genéticos tengan un paquete en Matlab que facilite su uso. En general, la clase se podría disfrutar aún más en un esquema más amplio, ya que todos los conceptos y los métodos implementados son bastante complicados pero creo que con un poco más de tiempo, realmente se podrían recrear de una mejor manera e indagar mejor en ellos, causando que su estudio sea más ameno y que se pueda disfrutar más.

Referencias

[Dukic et al., 2012] Dukic, V., Lopes, H. F., and Polson, N. G. (2012). Tracking epidemics with google flu trends data and a state-space seir model. *Journal of the American Statistical Association*, 107(500):1410–1426.

[Eisenberg, 2021] Eisenberg, J. (2021). Qué es el r0, el número que siguen los científicos para ver la intensidad del coronavirus.

[Giraldo, 2008] Giraldo, J. O., . P. D. H. (2008). Deterministic sir (susceptible-infected-removed) models applied to varicella outbreaks. *Epidemiology and Infection*, 72(2):169–184.

[GRAY et al., 2011] GRAY, A., GREENHALGH, D., HU, L., MAO, X., and PAN, J. (2011). A stochastic differential equation sis epidemic model. *SIAM Journal on Applied Mathematics*, 71(3):876–902.

[Mollison et al., 1994] Mollison, D., Isham, V., and Grenfell, B. (1994). Epidemics: Models and data. *Journal of the Royal Statistical Society. Series A (Statistics in Society)*, 157(1):115–149.