In [1]:
```
using Plots
using Random
using Distributions
using LinearAlgebra
using Dates
using LaTeXStrings
using BenchmarkTools
using MAT
using Statistics
```

```
[ Info: Precompiling BenchmarkTools [6e4b80f9-dd63-53aa-95a3-0cdb28fa8baf]
```

In [15]:
```
Δr=vcat(rs,rs[1,:]')##Trayectoria
D=abs.(diff(Δr,dims=1));
Ds=norm(sum(D,dims=1));


#Dss=zeros(n)
Dss=Ds
Data_ty=Dict();
Data_ty[1]=rs;
Ds
#####Ordenar el vector

norms = [norm(row) for row in eachrow(rs)];
indices=sortperm(norms);
Rs=rs[indices,:];
DR,ΔR=dista(Rs)
DR
Dss=Ds
if DR<Ds
    Data_ty[1]=Rs;
    Data_ty[2]=ΔR;
    Dss=DR
end;
print("DS=",Ds)
print("  DR=",DR)
```

```
DS=1130.5096195964013  DR=833.1170385966188
```

Importar datos

In [2]:
```
data = matread("C:\\Users\\Arif\\Downloads\\datat5.mat")
rs=hcat(data["x"],data["y"],data["y"].*0)
N=length(rs[:,1])
```

Out[2]:
```
150
```

Numero de veces a correr el recocido simulado N

$$N = 100$$

In [20]:
```
NN=100
m=zeros(NN,2);
```

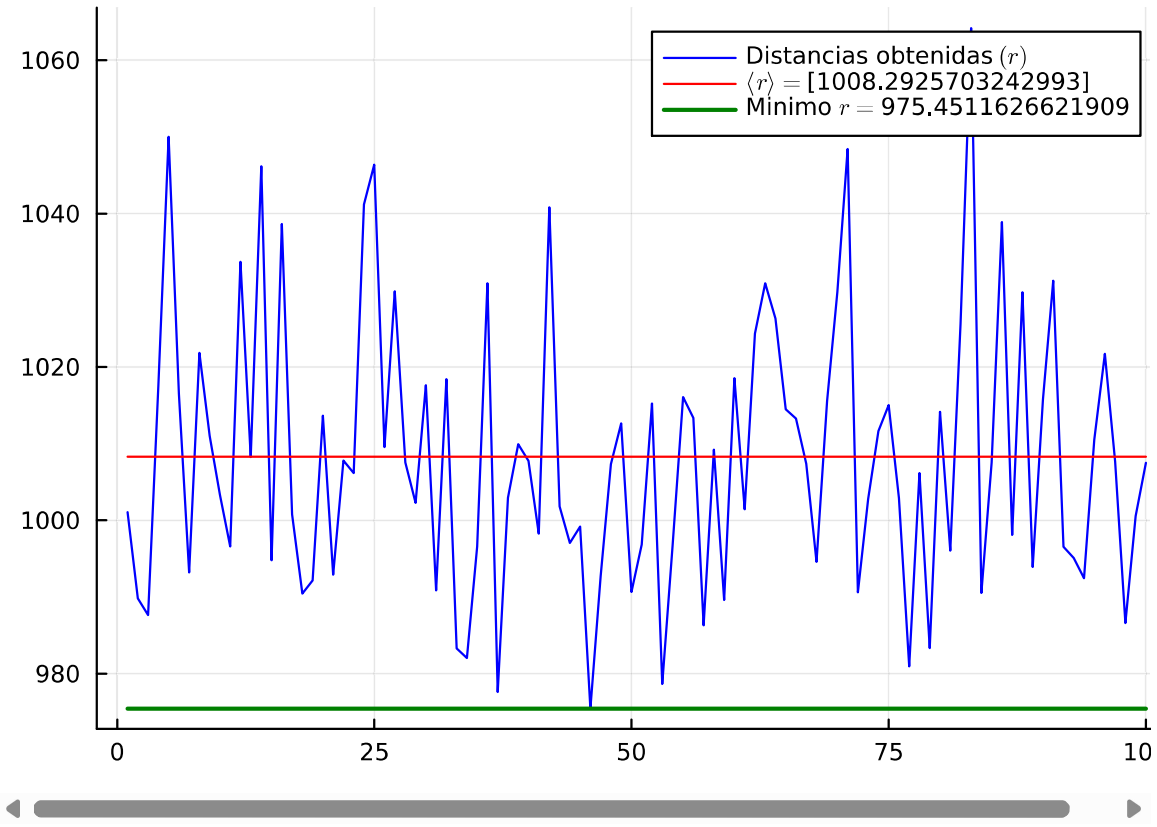
In [21]:
```
bp=Dict()
bp[1]=dista(rs)[1]
```

Out[21]:
```
7637.891502298925
```

In [22]:
```
initial_temperature = 100.0
cooling_factor = 0.995
max_iterations = 2000

for i in 1:NN
    best_distance, best_path ,time= simulated_annealing_optimized(rs, initial_te
    m[i,:]=[best_distance , time ]'
    if best_distance<bp[1]
        bp[1]=best_distance
        bp[2]=best_path
    end

end
#println("Shortest path:", best_path)
#println("Shortest distance:", best_distance)
#println("Time ", time)
```

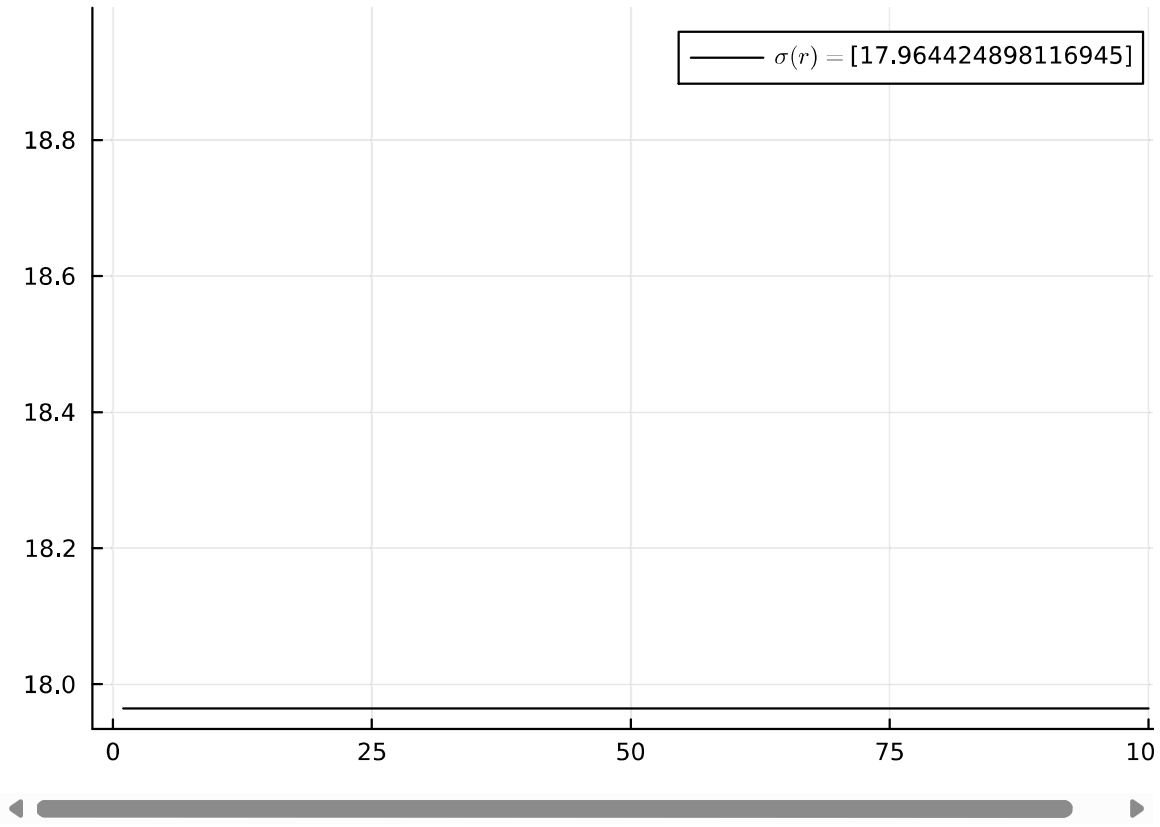
In [57]:
```
plot(m[:,1],color="blue",label="Distancias obtenidas "*L"(r)")
plot!(ones(NN).*mean(m[:,1],dims=1),color="red",label=L"\langle r \rangle="*stri
plot!(ones(NN).*bp[1],label="Minimo "*L" r="*string(bp[1]),color="green",lw=2)
```

Out[57]:



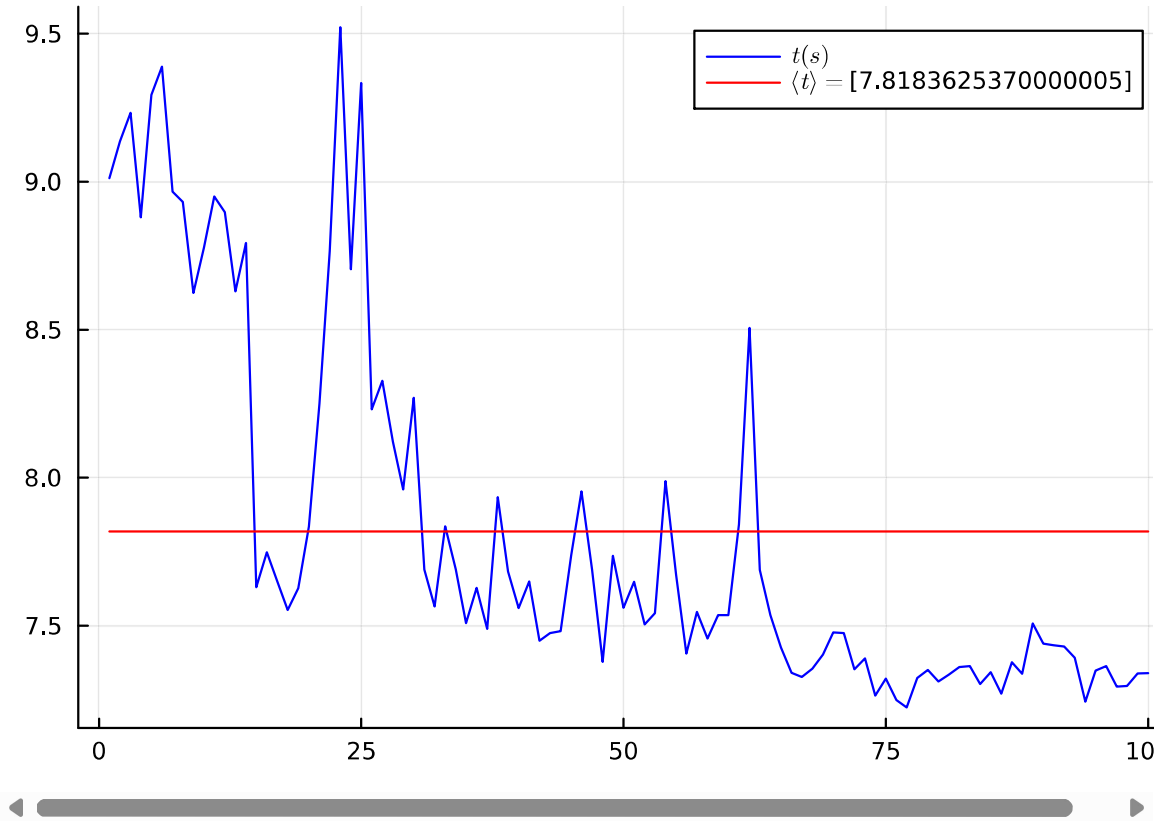
In [31]:
```
plot(ones(NN).*std(m[:,1],dims=1),color="black",label=L"\sigma(r)="*string(std(m
```

Out[31]:



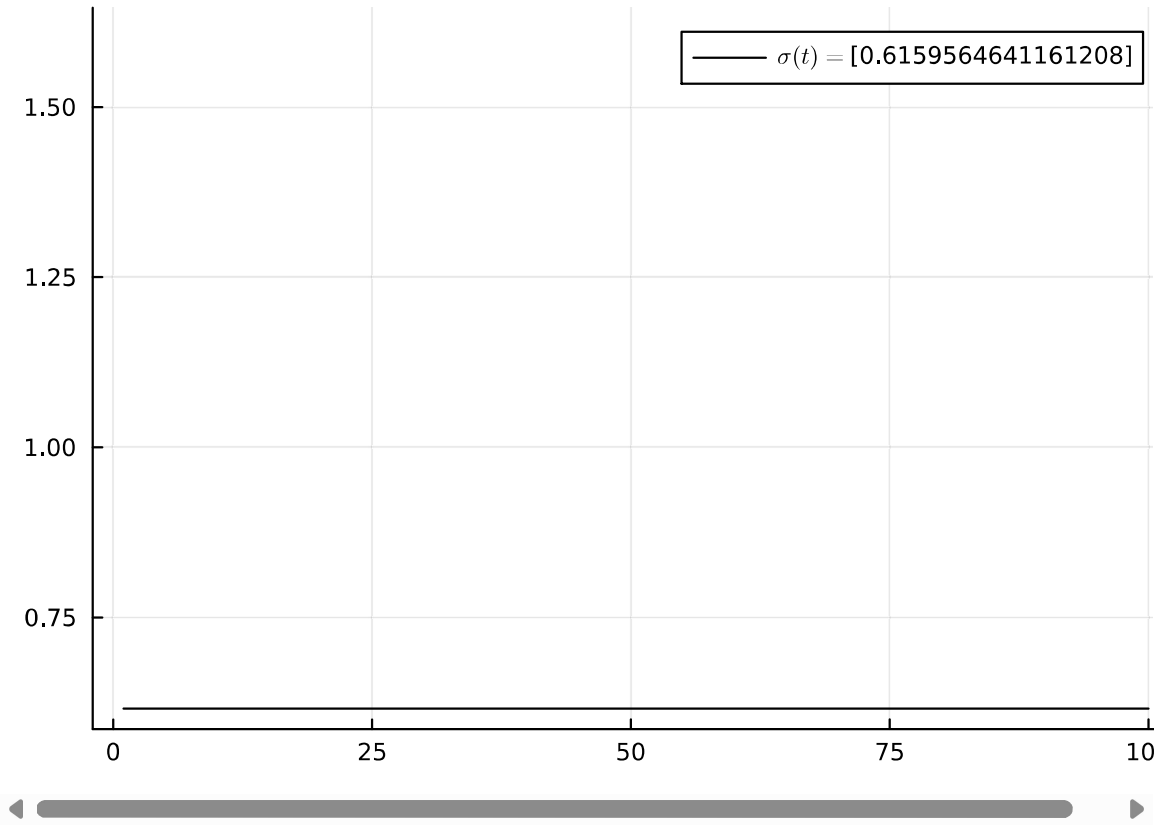
In [42]:
```
plot(m[:,2],color="blue",label=L" t(s)")
plot!(ones(NN).*mean(m[:,2],dims=1),color="red",label=L"\langle t \rangle="*stri
```

Out[42]:



In [34]:
```
plot(ones(NN).*std(m[:,2],dims=1),color="black",label=L"\sigma(t)="*string(std(m
```

Out[34]:



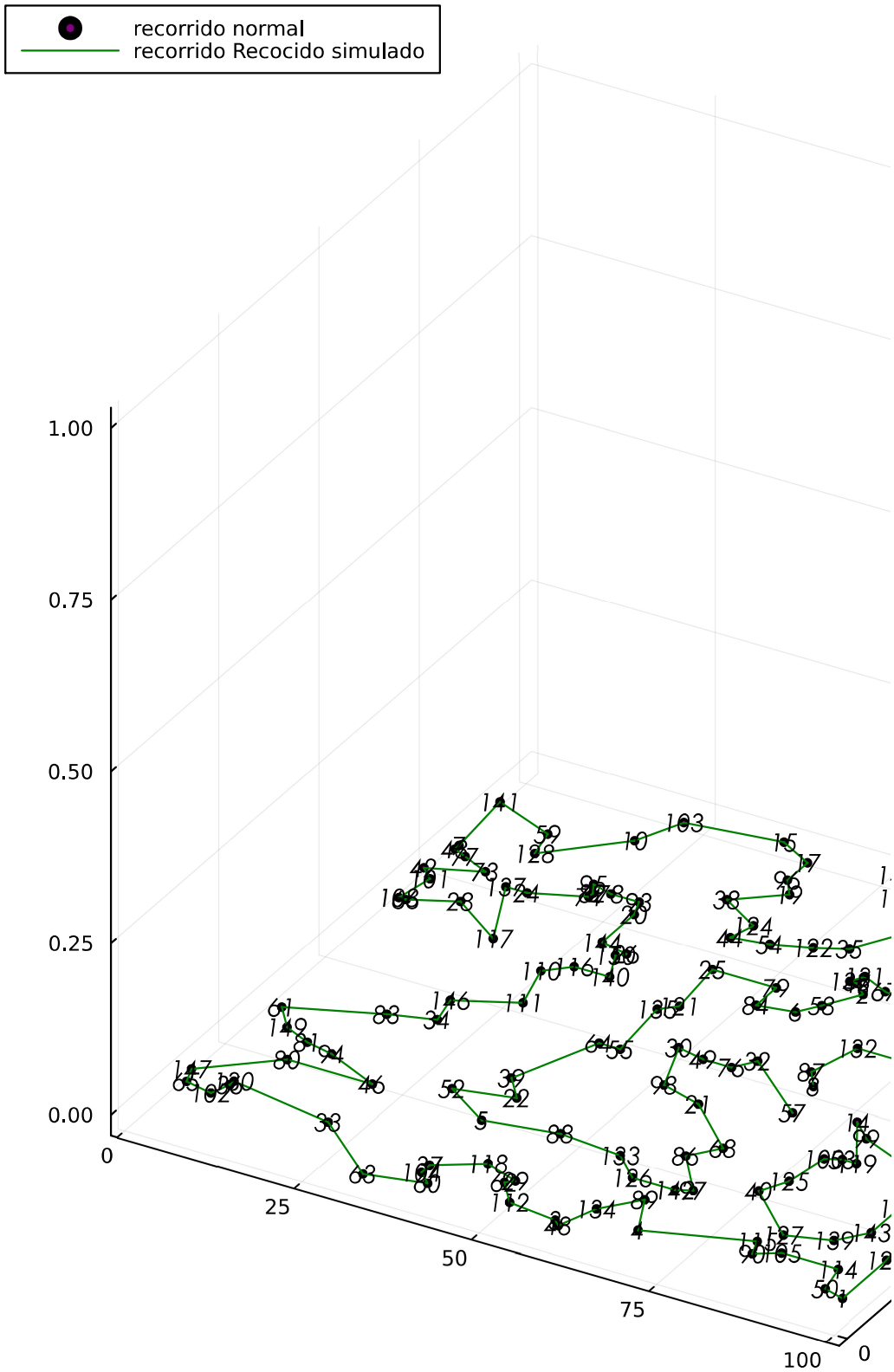
promedios y desviacion estandar

In [36]:
```
println(mean(m,dims=1))
println(std(m,dims=1))
```

```
[1008.2925703242993 7.8183625370000005]
[17.964424898116945 0.6159564641161208]
```

Recocido

In [96]:
```
pp=plot(size=(900, 900))
scatter!(rs[bp[2],1],rs[bp[2],2],rs[bp[2],3],color=:rainbow,label="recorrido nor

for i in 1:N
    annotate!([(rs[bp[2],1][i],rs[bp[2],2][i],rs[bp[2],3][i], text(bp[2][i], :up
end

plot!(rs[bp[2],1],rs[bp[2],2],rs[bp[2],3],color="green",label="recorrido Recocic
display(pp)
```



In [73]:
```
i=1
rs[bp[2], 1][i]
```

Out[73]: 70.0

In [60]:
```
rs[bp[2],1]
```

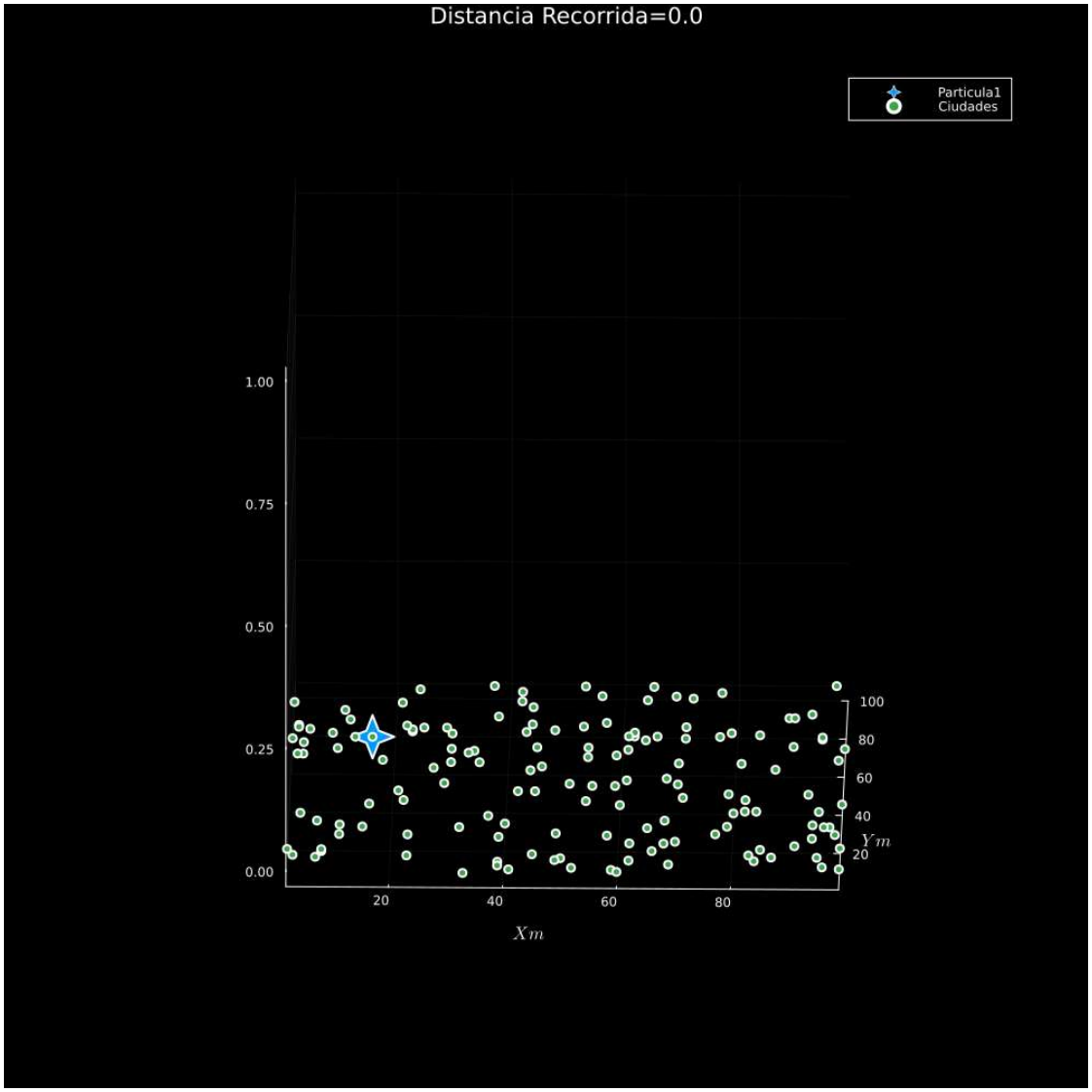
Out[60]:
```
150-element Vector{Float64}:
 70.0
 68.0
 70.0
 81.0
 87.0
 93.0
 99.0
 95.0
 82.0
 79.0
 84.0
 82.0
 80.0
  ⋮
 62.0
 68.0
 70.0
 65.0
 68.0
 60.0
 54.0
 51.0
 55.0
 59.0
 61.0
 71.0
```

Animación

In [150…
```
animac(rs[best_path,1]',rs[best_path,2]',rs[best_path,3]',1,"Optimizacion2d")
```

```
[ Info: Saved animation to C:\Users\Arif\OneDrive - Instituto Tecnologico y de Estudios Superiores de Monterrey\Estocasticos\Tarea\servando\Torneo\Optimizacion2d.gif
```

Out[150…



In [10]:

```
function animac(xs,ys,zs,N,titulo)
    n=length(xs)
    anim2d=@animate for i in 1:1:n;
        gr(size=(1000,1000))
        a=1
        j=1
        if i>1
            j=Int(ceil(i*0.3))
            a=range(0,1,length=(i-j+1))

        end
        for d in 1:N
            if d==1
                Plots.plot(xs[d,j:i],ys[d,j:i],zs[d,j:i],color=d,lw=2,label="",x
                Plots.scatter!([xs[d,i]],[ys[d,i]],[zs[d,i]], color = d,markersi
                title!("Distancia Recorrida="*string(round(dista(hcat(xs',ys',zs
                #Plots.scatter!([xs[d,i]],[ys[d,i]],[zs[d,i]], color = d,label="
            else
                Plots.plot!(xs[d,j:i],ys[d,j:i],zs[d,j:i],color=d,lw=2,label="",
                Plots.scatter!([xs[d,i]],[ys[d,i]],[zs[d,i]], color = d,markersi
                #title!("Distancia Recorrida="*string(dista(hcat(xs,ys,zs)[1:i,:
                #Plots.scatter!([xs[i]],[ys[i]],[zs[i]], color = 1,label="Distan
            end
        end
        scatter!(rs[:,1],rs[:,2],rs[:,3],label="Ciudades")
```

```
    end;
    gif(anim2d,titulo*".gif",fps=24);
    #anim = Animation(anim2d)
    #Animation.save(titulo * ".mp4", anim, fps = 24)


end
```

Out[10]: animac (generic function with 1 method)

In [5]:
```
function dista(rs)
    Δr = vcat(rs, rs[1, :]')  # Trayectoria
    D = diff(Δr, dims=1)
    Ds = sum(norm.(eachrow(D)))
    return Ds, Δr
end
```

Out[5]: dista (generic function with 1 method)

In [4]:
```
function simulated_annealing_optimized(rs, initial_temperature, cooling_factor,
    N = size(rs, 1)
    result = @timed begin
    # Define distance function
    function distance(city1, city2)
        return norm(city1 - city2)
    end

    # Define total distance function
    function total_distance(path)
        total = 0.0
        for i in 1:N-1
            total += distance(rs[path[i], :], rs[path[i+1], :])
        end
        total += distance(rs[path[N], :], rs[path[1], :])  # Return to the start
        return total
    end

    best_path = randperm(N)
    best_distance = total_distance(best_path)

    current_path = copy(best_path)
    current_distance = best_distance

    temperature = initial_temperature

    for iteration in 1:max_iterations
        for _ in 1:N
            # Generate a neighboring path by reversing a segment of the current
            i, j = sort(sample(1:N, 2, replace=false))
            neighbor_path = copy(current_path)
            neighbor_path[i:j] = reverse(neighbor_path[i:j])

            # Calculate distances
            neighbor_distance = total_distance(neighbor_path)
            delta_distance = neighbor_distance - current_distance

            # Decide whether to accept the new path
            if delta_distance < 0 || rand() < exp(-delta_distance / temperature)
                current_path = copy(neighbor_path)
```

```
                current_distance = neighbor_distance

                # Update the best path if necessary
                if current_distance < best_distance
                    best_path = copy(current_path)
                    best_distance = current_distance
                end
            end
        end

        # Cool down the temperature
        temperature *= cooling_factor
    end
end

    return best_distance, best_path,result.time
end
```

Out[4]: simulated_annealing_optimized (generic function with 1 method)

In [ ]:

```julia
using Plots
using Random
using Distributions
using Statistics
using LaTeXStrings
```

```julia
function Binomiali(N,p,nn) ### Numero de pruebas ##nn= numero de intentos
    options = [1, 0]
    probabilities = [p,1-p]  # Adjust the probabilities as needed

    # Create a random sample of indices using the probabilities
    sampled_indices = rand(Categorical(probabilities),nn,N)

    # Map the sampled indices to the options array
    dn = options[sampled_indices]

    n=zeros(nn,N)
    for i in 1:N
        n[:,i]=sum(dn[:,1:i],dims=2)
    end
    return n
end
```

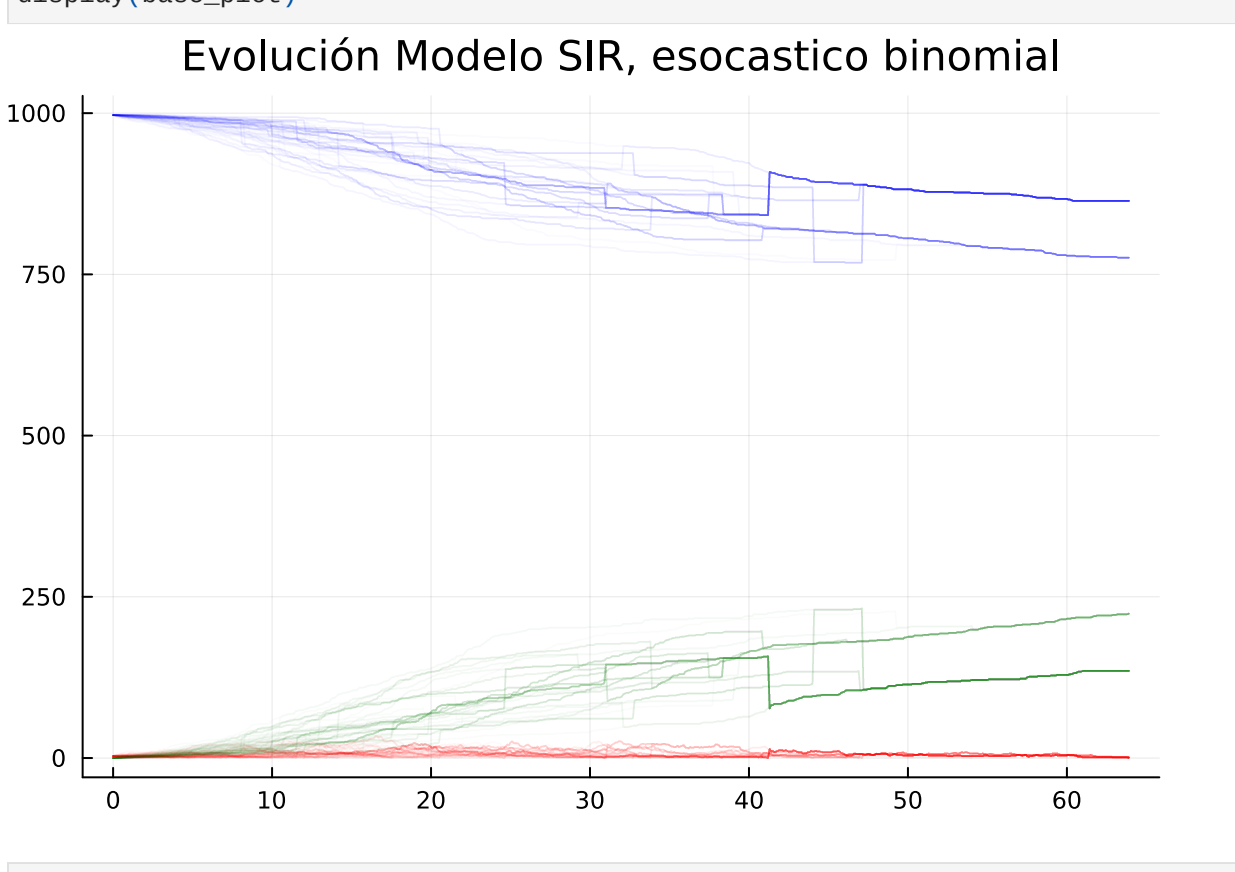
```
Binomiali (generic function with 1 method)
```

Modelo SIR Estocastico binomial

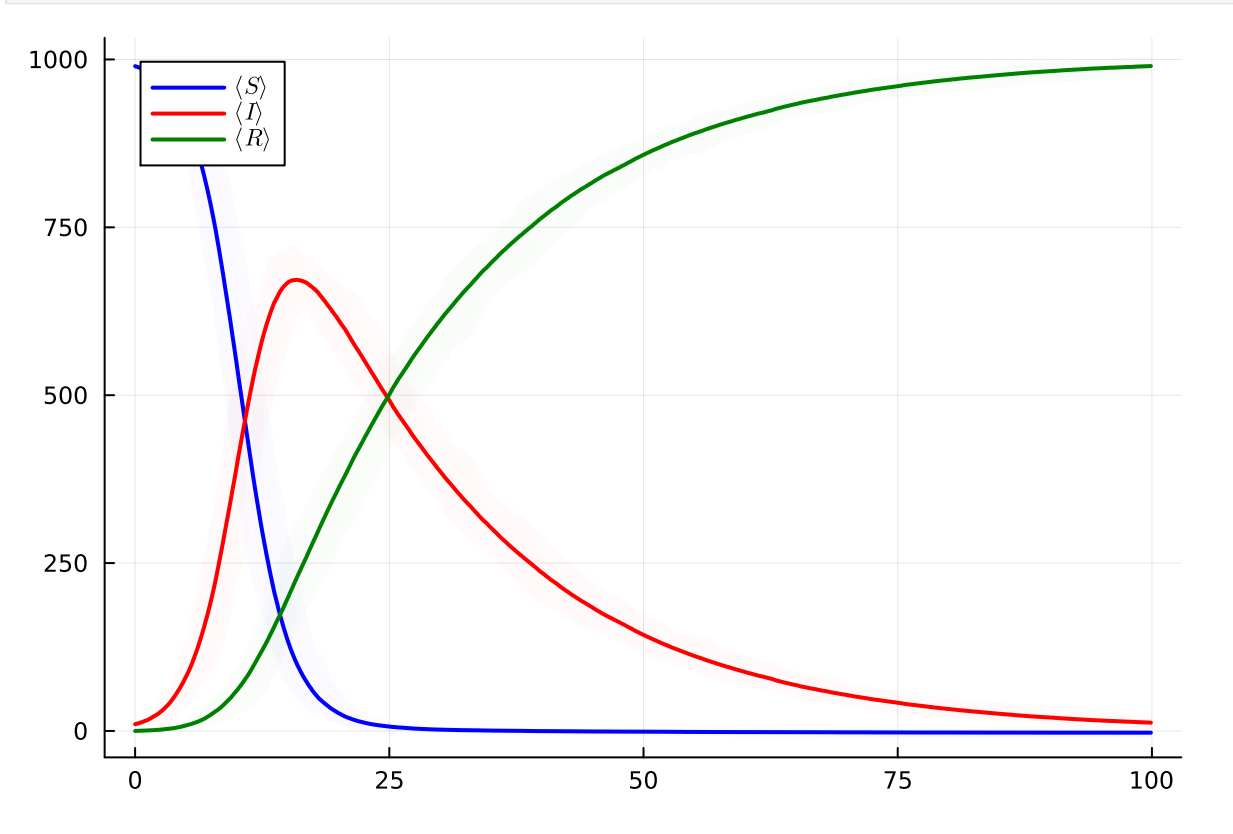
```julia
tf=100
NN=100
S,I,R,ts,tps=SIReb(997,3,100,1/2,1/2,1000,NN);

mt=mean(ts,dims=1)
ms=mean(S,dims=1)
sds=std(S,dims=1)

mi=mean(I,dims=1)
sdi=std(I,dims=1)

mr=mean(R,dims=1)
sdr=std(R,dims=1)

### tiempo
mean(tps)
std(tps)



#plot!(mt',ms', color="blue",label=L"\langle S \rangle",lw=2)
#plot!(mt',mi', color="red",label=L"\langle I \rangle",lw=2)
#plot!(mt',mr', color="green",label=L"\langle R \rangle",lw=2)
#plot!(mt',sds', label=L"\sigma(S)",lw=2)
```

```
0.16566999478569477
```

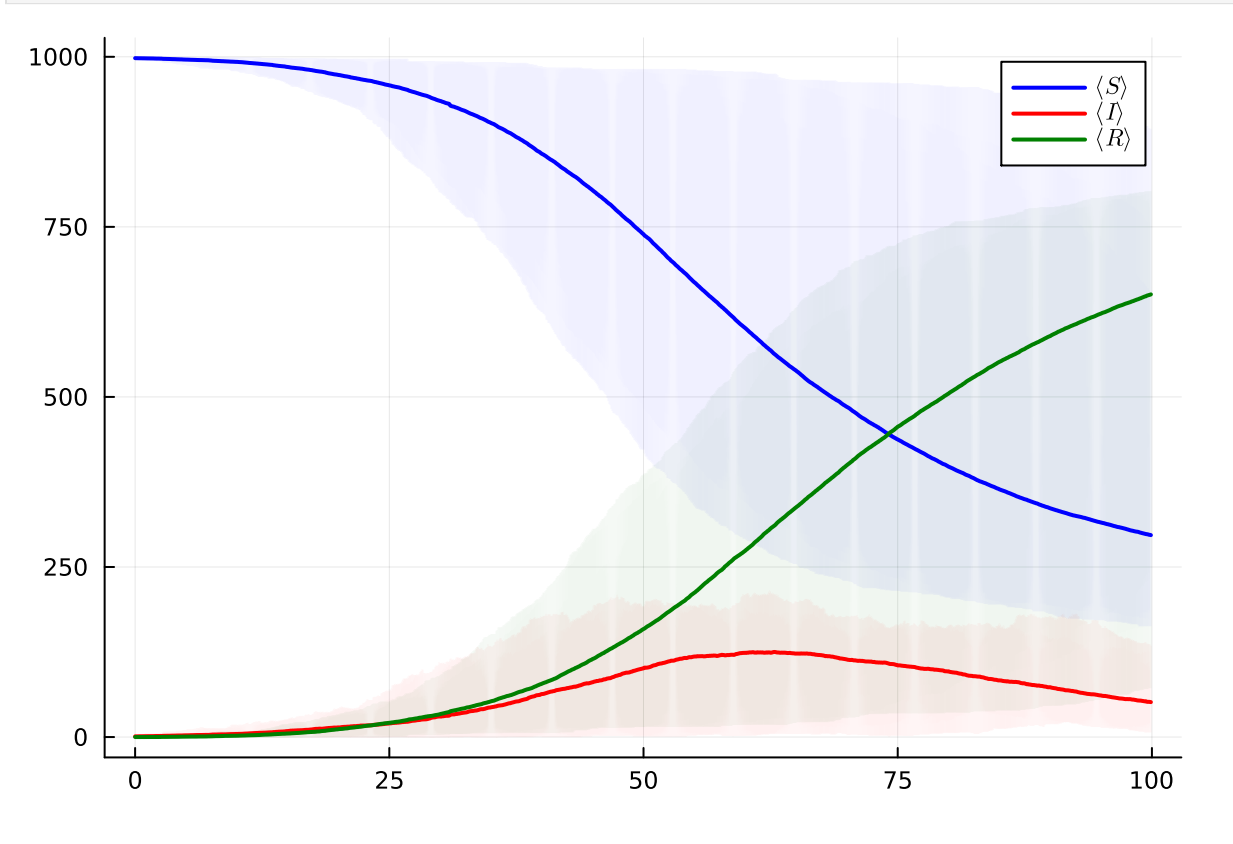
```julia
base_plot = plot(layout=(1, 1), legend=true)

for i in 1:NN

    non=vcat(1,findall(x -> x != 0, ts[1,:]))
    plot!(base_plot,ts[i,non],S[i,non],color="blue",label="",alpha=0.02, title="Evolución Modelo SIR, esocastico binomial")
    plot!(base_plot,ts[i,non],I[i,non],color="red",label="",alpha=0.02)
    plot!(base_plot,ts[i,non],R[i,non],color="green",label="",alpha=0.02)
end
display(base_plot)
```




```julia
tf=100

S,I,R,ts,tps=SIReb(990,10,100,1/20,1/2,1000,100);
plot(ts,S,color="blue",label="",alpha=0.01)
plot!(ts,I,color="red",label="",alpha=0.01)
plot!(ts,R,color="green",label="",alpha=0.01)
mt=mean(ts,dims=1)
ms=mean(S,dims=1)
sds=std(S,dims=1)

mi=mean(I,dims=1)
sdi=std(I,dims=1)

mr=mean(R,dims=1)
sdr=std(R,dims=1)

### tiempo
mean(tps)
std(tps)


plot!(mt',ms', color="blue",label=L"\langle S \rangle",lw=2)
plot!(mt',mi', color="red",label=L"\langle I \rangle",lw=2)
plot!(mt',mr', color="green",label=L"\langle R \rangle",lw=2)
#plot!(mt',sds', label=L"\sigma(S)",lw=2)
```



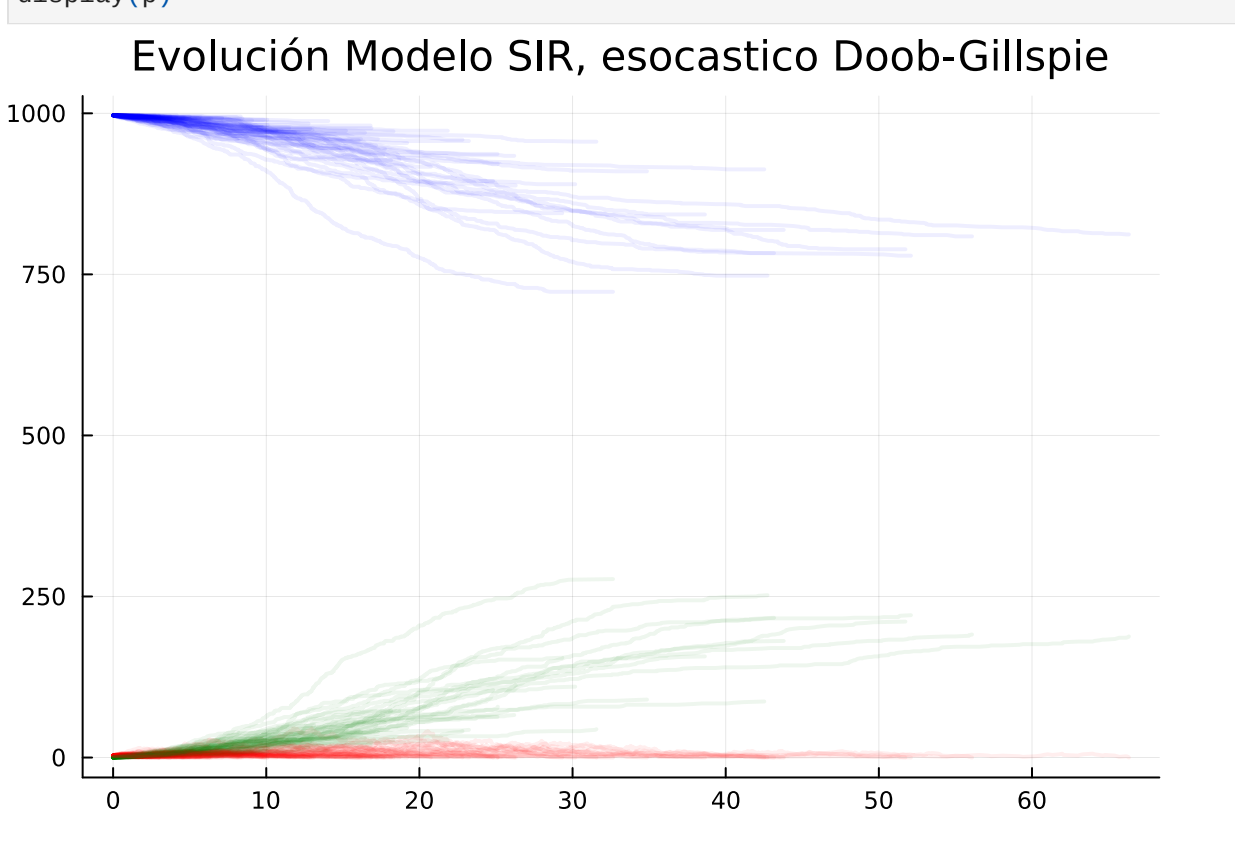

```julia
S,I,R,ts,tps=SIReb(998,1,100,0.1,0.2,1000,100);
plot(ts,S,color="blue",label="",alpha=0.02)
plot!(ts,I,color="red",label="",alpha=0.02)
plot!(ts,R,color="green",label="",alpha=0.02)
mt=mean(ts,dims=1)
ms=mean(S,dims=1)
sds=std(S,dims=1)

mi=mean(I,dims=1)
sdi=std(I,dims=1)

mr=mean(R,dims=1)
sdr=std(R,dims=1)

### tiempo
mean(tps)
std(tps)
plot!(mt',ms', color="blue",label=L"\langle S \rangle",lw=2)
plot!(mt',mi', color="red",label=L"\langle I \rangle",lw=2)
plot!(mt',mr', color="green",label=L"\langle R \rangle",lw=2)
#plot!(mt',sds', label=L"\sigma(S)",lw=2)
```



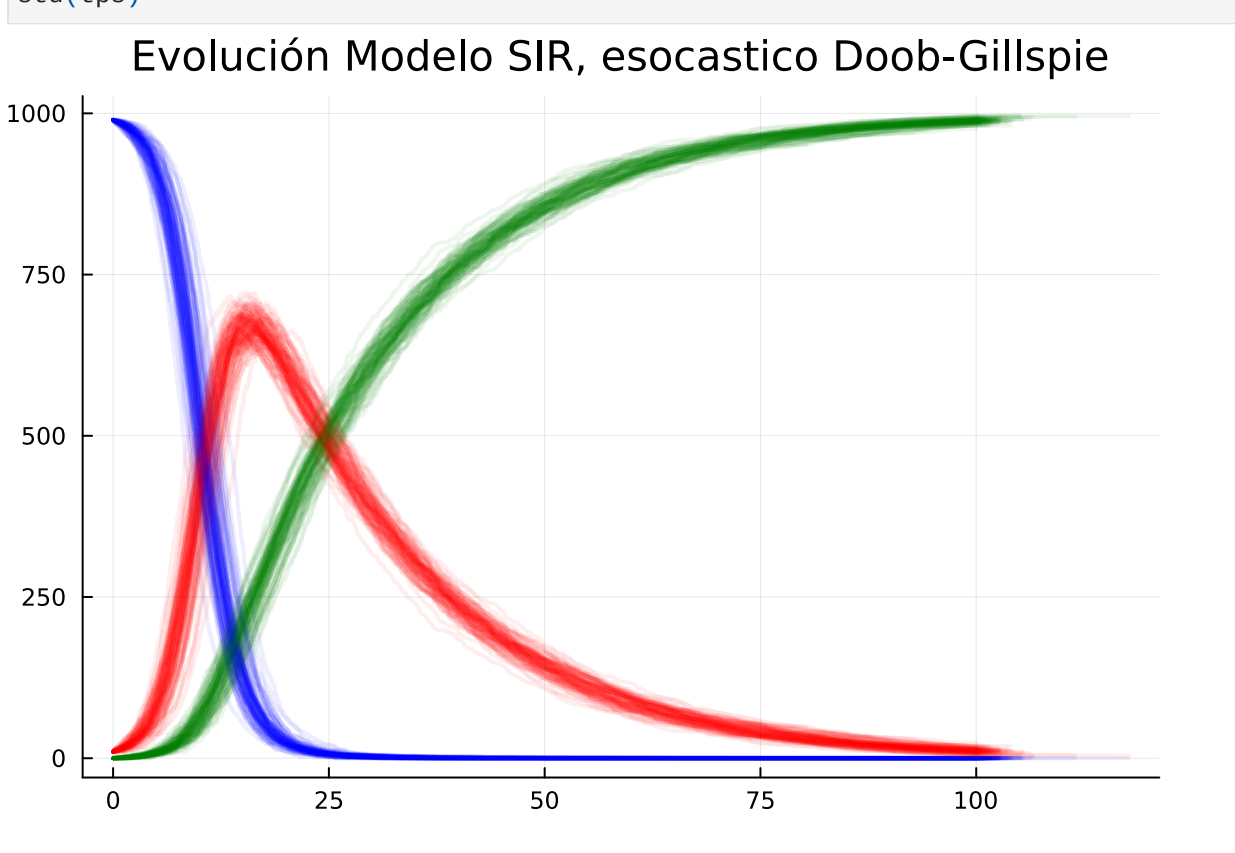
Modelo SIR Doob-Gillspie

$$S, I \to (S-1, I+1)$$
$$P = \beta\frac{SI}{N}$$
$$I, R \to (I-1, R+1)$$
$$P = \gamma I$$

```julia
S0=997
I0=3
tf=100
NN=100

β=1/2
γ=1/2

S1,I1,R1,ts1,tps1=SIRGB1(S0,I0,tf,γ,β,NN);
p=plot()
for j in 1:NN
    plot!(ts1[j]',S1[j]',label="",color="blue",lw=2,title="Evolución Modelo SIR, esocastico Doob-Gillspie",alpha=0.07)
    plot!(ts1[j]',I1[j]',label="",color="red",lw=2,alpha=0.07)
    plot!(ts1[j]',R1[j]',label="",color="green",lw=2,alpha=0.07)
end
display(p)
```




```julia
S0=990
I0=10
tf=100
NN=100

β=1/2
γ=1/20

S1,I1,R1,ts1,tps1 = SIRGB1(S0,I0,tf,γ,β,NN)
p=plot()
for j in 1:NN
    plot!(ts1[j]',S1[j]',label="",color="blue",lw=2,title="Evolución Modelo SIR, esocastico Doob-Gillspie",alpha=0.07)
    plot!(ts1[j]',I1[j]',label="",color="red",lw=2,alpha=0.07)
    plot!(ts1[j]',R1[j]',label="",color="green",lw=2,alpha=0.07)
end
display(p)
mean(tps)
std(tps)
```

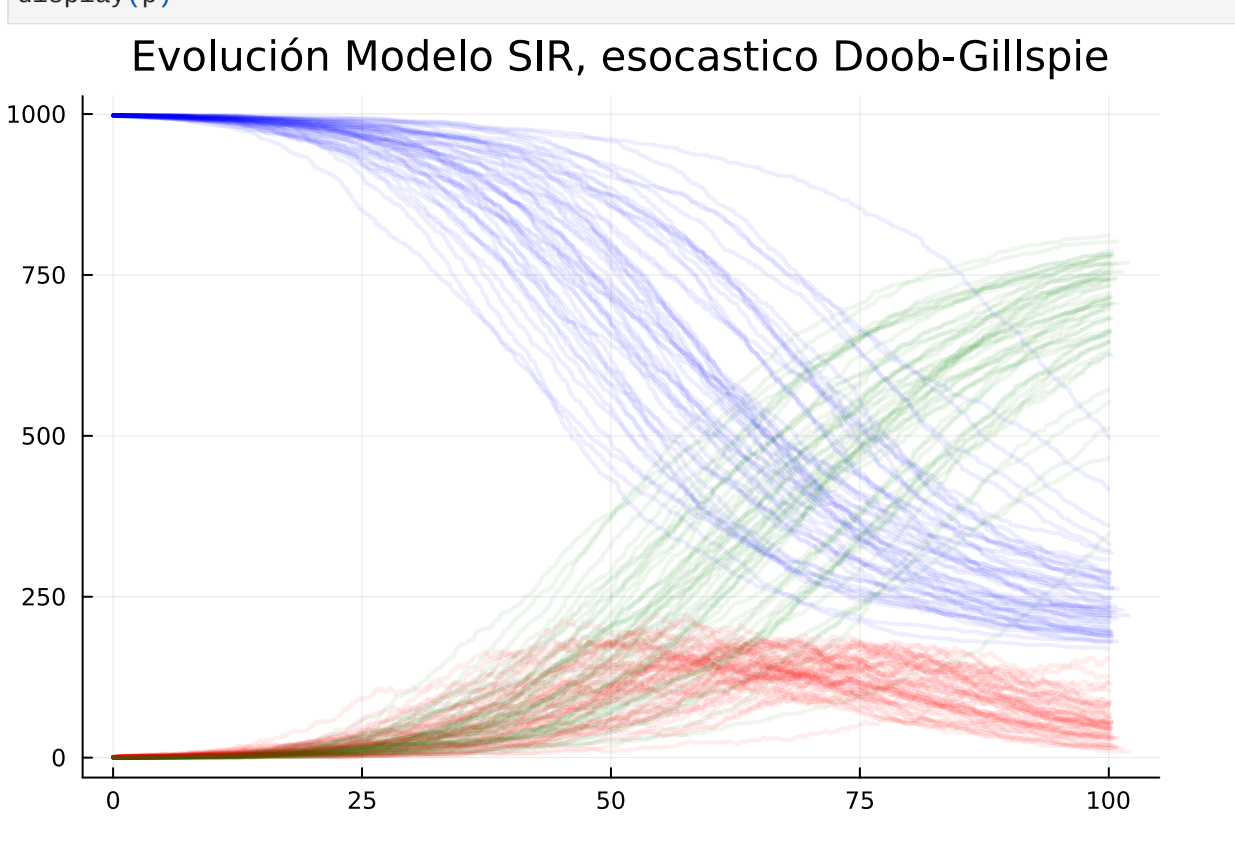



```
0.004360411585945468
```

```julia
mean(tps)
```

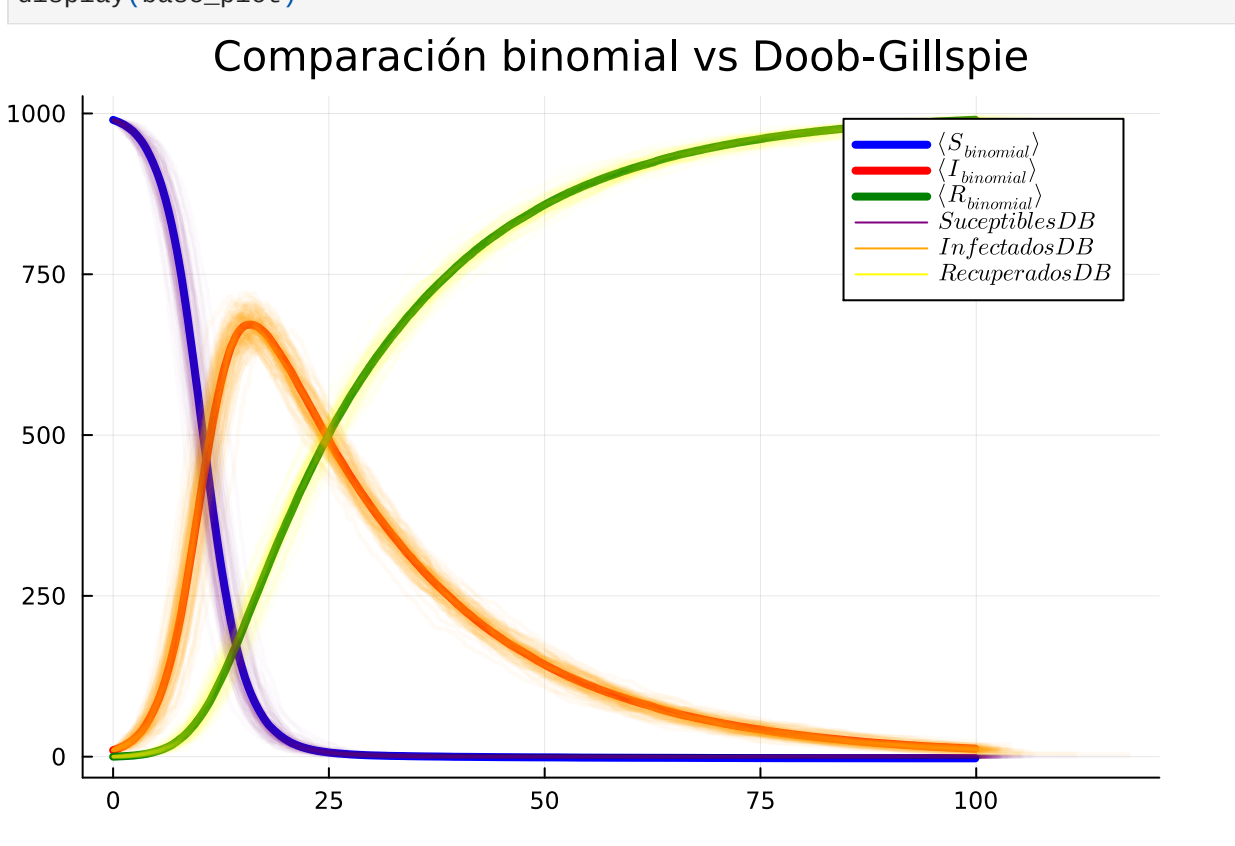
```
0.018770771
```

```julia
S0=998
I0=1
tf=100
NN=100

β=0.2
γ=0.1

S1,I1,R1,ts1,tps1 = SIRGB1(S0,I0,tf,γ,β,NN)
p=plot()
for j in 1:NN
    plot!(ts1[j]',S1[j]',label="",color="blue",lw=2,title="Evolución Modelo SIR, esocastico Doob-Gillspie",alpha=0.07)
    plot!(ts1[j]',I1[j]',label="",color="red",lw=2,alpha=0.07)
    plot!(ts1[j]',R1[j]',label="",color="green",lw=2,alpha=0.07)
end
display(p)
```



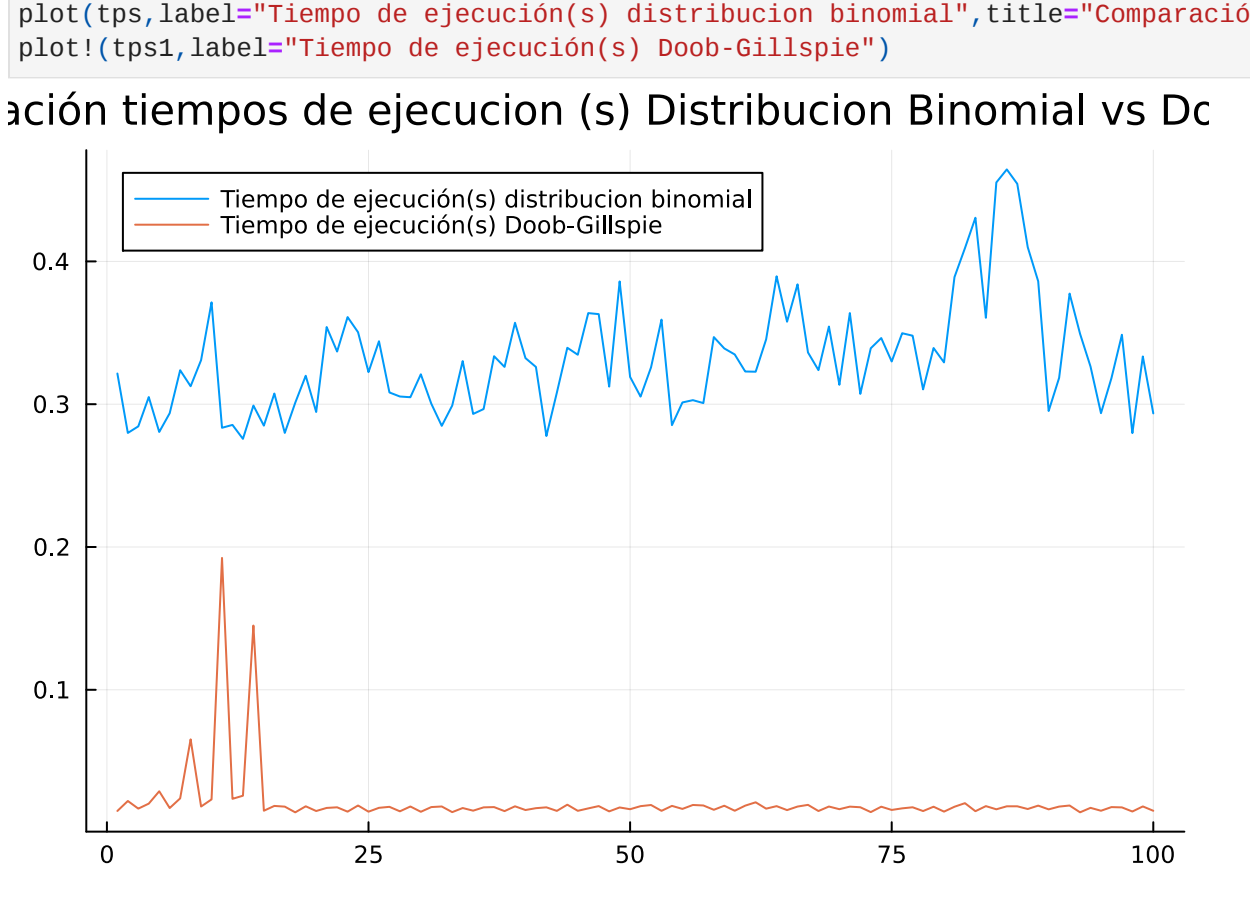
Comparacion binomial vs DB

```julia
# Create an empty plot with no legend, which will serve as the base plot
base_plot = plot(layout=(1, 1), legend=true)

plot!(base_plot, mt', ms', color="blue", label=L"\langle S_{binomial} \rangle", lw=4,title="Comparación binomial vs Doob-Gillspie")
plot!(base_plot, mt', mi', color="red", label=L"\langle I_{binomial} \rangle", lw=4)
plot!(base_plot, mt', mr', color="green", label=L"\langle R_{binomial} \rangle", lw=4)
plot!(base_plot,[NaN], [NaN],color="purple",label=L"Suceptibles DB")
plot!(base_plot,[NaN], [NaN],color="orange",label=L"Infectados DB")
plot!(base_plot,[NaN], [NaN],color="yellow",label=L"Recuperados DB")
for j in 1:NN
    # Create plots for the main data with labels and legend
# Create plots for the data specific to this iteration without labels or legend
    plot!(base_plot, ts1[j]', S1[j]', label="", color="Purple", lw=2, alpha=0.02)
    plot!(base_plot, ts1[j]', I1[j]', label="", color="orange", lw=2, alpha=0.05)
    plot!(base_plot, ts1[j]', R1[j]', label="", color="yellow", lw=2, alpha=0.02)
end

# Display the final plot with the legend
display(base_plot)
```



Comparación Tiempos

```julia
plot(tps,label="Tiempo de ejecución(s) distribucion binomial",title="Comparación tiempos de ejecucion (s) Distribucion Binomial vs Doob-Gillspie")
plot!(tps1,label="Tiempo de ejecución(s) Doob-Gillspie")
```




```julia
function SIReb(S0,I0,tf,γ,β,nn,NN)


    Δt=(tf)/nn
    ts=zeros(nn)
    Ts=zeros(NN,nn)
    tiempos=zeros(NN)
    #n=length(ts)
    ################
    SS=zeros(NN,nn)
    II=zeros(NN,nn)
    RR=zeros(NN,nn)
    S=zeros(nn);
    S[1]=S0
    I=zeros(nn);
    I[1]=I0
    N=S[1]+I[1]
    R=zeros(nn);

    function PSI(II,Δt)
         return (β*II*Δt/N)
    end
    function PIR(II,Δt)
        γ*Δt
    end



    for j in 1:NN
        i=1
        result = @timed begin
        while ts[i]<tf
            dt=Δt#rand(Exponential(1/i));
            ts[i+1]=ts[i]+dt
            if S[i]<=0
                SI=Binomiali(Int(1),PSI(I[i],dt),1)[end]
            else
                SI=Binomiali(Int(S[i]),PSI(I[i],dt),1)[end]
            end
            S[i+1]=S[i]-SI
            IR=Binomiali(Int(I[i]),PIR(I[i],dt),1)[end]

            I[i+1]=I[i]+SI-IR
            #println(IR)
            #println(SI)
            R[i+1]=R[i]+IR
            i+=1
            if i==nn || I[i]==0 #ts[i+1]>=tf
                #println("Finalizado")
                break
            end
            end
        end

        SS[j,:]=S'
        II[j,:]=I'
        RR[j,:]=R'
        Ts[j,:]=ts'
        tiempos[j]=result.time
    end
    repeat(range(0,10,100)',2)
    return SS,II,RR,Ts,tiempos#SS,II,RR
end
```

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

```julia
function SIRGB1(S0,I0,tf,γ,β,NN)
        SS=Dict()
        II=Dict()
        RR=Dict()
        TT=Dict()
        tiempos=zeros(NN)

    function PSIGP(S,I)
            N=S0+I0
            return (β*S*I/N)
        end

        function PIRGP(I)
            γ*I
        end

        infection_events = 0
        recovery_events = 0


    for j in 1:NN
        i=0
        S=[S0]
        I=[I0]
        R=[0]
        ts=[0]
        #for i in 1:3
        result = @timed begin
        while ts[end]<=tf
            i+=1
            ### Escoger reaccion
            SI=PSIGP(S[i],I[i])
            IR=PIRGP(I[i])
            total=SI+IR
            ####Escoger tiempo
            r1,r2=rand(2)
            Δt=(1 / total).* log(1 / r1)
            ts=hcat(ts,ts[i]+Δt)

            if r2 < (SI / total)
                    # Infection event
                    S =hcat(S,S[i]-1)
                    I =hcat(I,I[i]+1)
                    R = hcat(R,R[i])
                    infection_events += 1
                else
                    # Recovery event
                    S =hcat(S,S[i])
                    I =hcat(I,I[i]-1)
                    R = hcat(R,R[i]+1)
                    recovery_events += 1
            end
        end
        end
        SS[j]=copy(S)
        II[j]=copy(I)
        RR[j]=copy(R)
        TT[j]=copy(ts)
        tiempos[j]=result.time
    end
    return SS,II,RR,TT,tiempos
end
```

Out[ ]: `DIM001 (generic function with 1 method)`

In [ ]:

```python
import numpy as np
from scipy.integrate import odeint
import matplotlib.pyplot as plt
import tensorflow as tf
from sklearn.preprocessing import MinMaxScaler

# Definir tasas, condiciones iniciales y rango de tiempo
beta = 0.5
gamma = 0.05
S0 = 0.99
I0 = 0.01
R0 = 0.0
tspan = np.arange(0, 100, 0.1)

# Definir la función de ODE
def ode(y, t):
    S, I, R = y
    dSdt = -beta * S * I
    dIdt = beta * S * I - gamma * I
    dRdt = gamma * I
    return [dSdt, dIdt, dRdt]

# Condiciones iniciales
y0 = [S0, I0, R0]

# Resolver el sistema de ODE con scipy
y = odeint(ode, y0, tspan)

# Agregar ruido aleatorio
np.random.seed(0)
noise = 0.2  * np.random.randn(y.shape[0], y.shape[1])
y_with_noise = y + noise

# Crear una red neuronal en TensorFlow/Keras
def create_model():
    model = tf.keras.Sequential([
        tf.keras.layers.Dense(50, activation='relu'),
        tf.keras.layers.Dense(30, activation='relu'),
        tf.keras.layers.Dense(10, activation='relu'),
        tf.keras.layers.Dense(1)
    ])
    return model

# Normalizar los datos de entrada
scaler = MinMaxScaler()
t_normalized = scaler.fit_transform(tspan.reshape(-1, 1))

# Crear y entrenar la red neuronal para cada proporción
models = []
for i in range(3):
    model = create_model()
    model.compile(optimizer='adam', loss='mean_squared_error')
    model.fit(t_normalized, y_with_noise[:, i], epochs=1000, batch_size=120)
    models.append(model)

# Generar nuevos datos para suavizar
t_new = np.linspace(0, 100, 1000)
t_new_normalized = scaler.transform(t_new.reshape(-1, 1))
smoothed_data = []

for model in models:
    smoothed_normalized = model.predict(t_new_normalized)
    smoothed = smoothed_normalized * (max(y[:, 0]) - min(y[:, 0])) + min(y[:, 0])
    smoothed_data.append(smoothed)
```

```
65
66
67  #mse_susceptibles = np.abs(np.mean((y_with_noise[:, 0] - smoothed_data[0])))
68  #mse_infectados = np.abs(np.mean((y_with_noise[:, 1] - smoothed_data[1])))
69  #mse_recuperados = np.abs(np.mean((y_with_noise[:, 2] - smoothed_data[2])))
70
71  mse_susceptibles = np.mean(1-(np.abs(y_with_noise[:, 0] - smoothed_data[0])))
72  mse_infectados = np.mean(1-(np.abs(y_with_noise[:, 1] - smoothed_data[1])))
73  mse_recuperados = np.mean(1-(np.abs(y_with_noise[:, 2] - smoothed_data[2])))
74
75
76  # Graficar los resultados en una sola gráfica
77  plt.figure(figsize=(10, 6))
78  plt.plot(t_new, smoothed_data[0], label='Susceptibles Suavizada', linewidth=2)
79  plt.plot(t_new, smoothed_data[1], label='Infectados Suavizada', linewidth=2)
80  plt.plot(t_new, smoothed_data[2], label='Recuperados Suavizada', linewidth=2)
81  plt.plot(tspan, y_with_noise[:, 0], 'b--', label='Susceptibles Datos Originales', alpha=0.5)
82  plt.plot(tspan, y_with_noise[:, 1], 'g--', label='Infectados Datos Originales', alpha=0.5)
83  plt.plot(tspan, y_with_noise[:, 2], 'r--', label='Recuperados Datos Originales', alpha=0.5)
84
85  # Agregar los valores de MSE al gráfico
86  plt.text(35, -0.05, f'MSE Susceptibles: {mse_susceptibles:.6f}', fontsize=10)
87  plt.text(35, -0.1, f'MSE Infectados: {mse_infectados:.6f}', fontsize=10)
88  plt.text(35, -0.15, f'MSE Recuperados: {mse_recuperados:.6f}', fontsize=10)
89
90  plt.xlabel('Tiempo')
91  plt.ylabel('Proporción')
92  plt.legend()
93  plt.title('Simulación sir 20% de ruido')
94  plt.grid(True)
95  plt.show()
96
97
```