Project: Estimating transition rates from stochastic trajectories of a two-level system

Flavio Samu Kindler
Supervisor: Prof. Dr. Patrick Potts

May 7, 2025

Abstract

In fluorescence experiments, the transition rates between different states of a molecule can be estimated by observing transitions in real time. This project theoretically investigates the estimation of transition rates from real-time trajectories of a two-level system.
The maximum likelihood estimator was derived and real-time trajectories were simulated in Julia using the Gillespie algorithm. The error of the derived likelihood estimator was quantified regarding the measurement time and different rate combinations.
The results can serve as a basis for investigating the possibility of estimating time-dependent transition rates.

1 Introduction

In fluorescence experiments, the transition rates between different states of a molecule can be estimated by observing transitions in real time. This project investigates the accuracy of estimated transition rates of a basic two-level system.
There are many two-level systems in physics, such as a spin-$\frac{1}{2}$ particle used e.g. in quantum computing using semi-conductors, the description of interfering quantum states and the ground and first excited state of an atom.
Fluorescence resonance energy transfer (FRET) is applied to probe biomolecular conformational changes or interactions. By measuring the change in fluorescence of donor and acceptor (called FRET-pair) of an energy transfer, one can use such a FRET-pair as a spectroscopic ruler.[1]
Accurately estimating the state change rates in such systems can be useful. This project serves as a basis to simulate and estimate more complex systems.

2 Two-level system

We are considering a two-level system with ground state 0 and excited state 1. The change of the system will be described by the two rates Γ_{10} and Γ_{01}. With Γ_{10} denoting the rate of the transition $0 \to 1$ and Γ_{01} denoting $1 \to 0$.

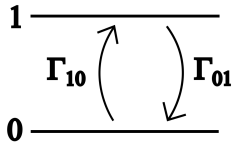

Figure 1: Sketch of a two-level systems with levels 0 and 1 and transition rates Γ_{10} and Γ_{01}.

Such a two-level system is described by the following rate equation

$$\partial \begin{pmatrix} p_0(t) \\ p_1(t) \end{pmatrix} = \begin{pmatrix} -\Gamma_{10} & \Gamma_{01} \\ \Gamma_{10} & -\Gamma_{01} \end{pmatrix} \begin{pmatrix} p_0(t) \\ p_1(t) \end{pmatrix}, \tag{1}$$

with $p_0(t)$ describing the probability, that the system is in state 0 at time t and $p_1(t)$ describing the probability, that the system is in state 1 at time t.
Using $p_0(t) = 1 - p_1(t)$ one can solve Eq.(1). The probability that the system is in state 1 is given by

$$p_1(t) = e^{-(\Gamma_{10}+\Gamma_{01})t} \left[p_1(0) - \frac{\Gamma_{10}}{\Gamma_{10}+\Gamma_{01}} \right] + \frac{\Gamma_{10}}{\Gamma_{10}+\Gamma_{01}}. \tag{2}$$

When observing a two-level-system over a time τ, the measured changes over time can be described as a trajectory

$$\nu_\tau = \{ n(t) \,|\, t \in [0, \tau] \}. \tag{3}$$

The trajectory describes the current state $n(t)$ at any time t. Illustrations of trajectories plotted as a function of time are shown in Fig.2.

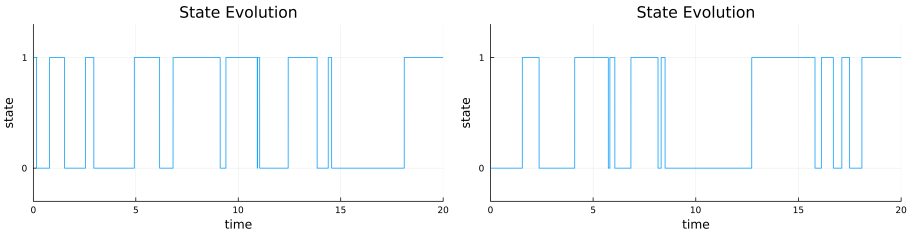

Figure 2: Two trajectories of the state evolution. The chosen transition rates here are $\Gamma_{10} = \Gamma_{01}$.

As illustrated in Fig.2, the system jumps between states 0 and 1 at random times. N is the number of total jumps and $n_i \in \{0, 1\}$ denotes the i-th state and n_0 therefore describing the starting state. The time intervals between jumps are called θ_i. These are distributed exponentially by the following waiting time distribution (WTD):

$$W_{n_i}(\theta_i) = \Gamma_{n_i} e^{-\theta_i \Gamma_{n_i}}, \tag{4}$$

with $\Gamma_1 = \Gamma_{10}$ and $\Gamma_0 = \Gamma_{01}$ denoting the i-th jump into the i-th state n_i. The probability to get a trajectory ν_τ over a fixed time τ is

$$p(\nu_\tau) = p_{n_0} \cdot e^{-\Gamma_{n_N}\left(\tau - \sum_{i=1}^{N} \theta_j\right)} \cdot \left(\prod_{j=1}^{N} W_{n_j}(\theta_j)\right). \tag{5}$$

Eq.(5) describes the probability of a trajectory ν_τ by multiplying the probability of the start state p_{n_0} by the distribution of each time interval between jumps (second term) then multiplying with the probability, that no jump happens for the remaining time for the last state n_N (third term).
If we choose k as the number of jumps up, l as the number of jumps down, and τ_1 as the total time the system is in state 1, we can rewrite Eq.(5) into a simpler form as a probability density to observe a specific trajectory ν_τ that derives directly from Eq.(5)

$$p(\nu_\tau | \Gamma_{10}, \Gamma_{01}) = p_{n_0} \Gamma_{10}^k \Gamma_{01}^l e^{-\Gamma_{10}(\tau - \tau_1) - \Gamma_{01}\tau_1} \tag{6}$$

Again, start with the steady state p_{n_0}, but this time multiply with the corresponding rates as many times as there are jumps: Multiply by Γ_{10}^k and the exponential density for the total time $\tau_0 = \tau - \tau_1$ for when the state is 0: $e^{-\Gamma_{10}(\tau - \tau_1)}$.
Analogously multiply by Γ_{01}^l and the exponential density for the total time τ_1 for when the state is 1: $e^{-\Gamma_{01}\tau_1}$. [2]

3 Estimation of Rates

With the maximum likelihood function, we can determine the estimators $\hat{\Gamma}_{10}$ and $\hat{\Gamma}_{01}$ for the transition rates:

$$\left(\hat{\Gamma}_{10}, \hat{\Gamma}_{01}\right) = \underset{\Gamma_{10},\Gamma_{01}}{\arg\max} \log P(\nu|\Gamma_{10}, \Gamma_{01}) \tag{7}$$

To maximize this, we need to solve

$$\vec{\nabla}_{\Gamma} \log P(\nu|\Gamma_{10}, \Gamma_{01})\Big|_{\substack{\Gamma_{10}=\hat{\Gamma}_{10} \\ \Gamma_{01}=\hat{\Gamma}_{01}}} = 0 \tag{8}$$

Plugging the probability of Eq.(6) into Eq.(8) and considering a set starting state of $n_0 = 0$ or 1 we get the following estimators

$$\begin{aligned} \hat{\Gamma}_{10} &= \frac{k}{\tau - \tau_1} \\ \hat{\Gamma}_{01} &= \frac{l}{\tau_1} \end{aligned} \tag{9}$$

4 Simulation

The simulation was performed in the Julia programming language.

To sample the time intervals between jumps, the Gillespie Algorithm was used to get the exponential distribution of θ_i described by Eq.(4). The process is explained in chapter 3.1.

The trajectory ν_τ is realized as two saved arrays chronologically listing the state n_i (0 or 1) in the first array and the time passed θ_i in that state in the second array.

The simulation will be run several times for a fixed total time τ. For each simulation, the number of jumps k, l and the time in state one τ_1 will be read out. With this, the estimators $\hat{\Gamma}_{10}$ and $\hat{\Gamma}_{01}$ are calculated with Eq.(9) for each simulated trajectory. These calculated estimators are then visualized in a $\hat{\Gamma}_{10}$-$\hat{\Gamma}_{01}$-scatterplot.

The covariance matrix of the two estimators is calculated and converted to confidence ellipses for 50% and 95%. These are also drawn over the scatter-plot to visualize the accuracy and correlation of the two estimators $\hat{\Gamma}_{10}$ and $\hat{\Gamma}_{01}$. The calculation and plotting is explained in chapter 3.2.

4.1 Gillespie Algorithm

At the start, a given total time τ and the rates Γ_{01} and Γ_{10} are determined for the simulation.
Then the initial state is sampled at an equal chance to be 0 or 1.

To sample the different time intervals θ_i we first sample $r_i \in U[0,1)$ from a uniform distribution integrated in *Julia*. The generated r_i are saved in an array and converted to the exponential distribution required given in eq.(4) as follows:

$$\theta_i = -\frac{1}{\Gamma_{n_i}} \ln(r_i) \tag{10}$$

with $\Gamma_1 = \Gamma_{10}$ and $\Gamma_0 = \Gamma_{01}$ denoting the i-th jump into the i-th state n_i.
The distribution of sampled θ_i has been plotted as a histogram in *Julia* to visualize and verify it is exponentially distributed as expected.

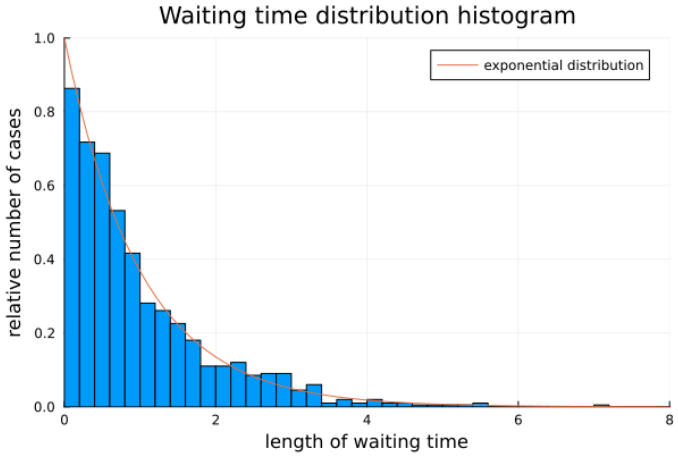

Figure 3: Normalized histogram of the sampled waiting times θ_i for the rates $\Gamma_{10} = \Gamma_{01}$ in comparison with the corresponding waiting time exponential distribution $W(\theta) = \Gamma_{01} e^{-\theta \cdot \Gamma_{01}}$. Here simulated for $\Gamma_{10} = \Gamma_{01}$ and time $\tau \cdot \Gamma_{01} = 1000$.

Before the sampling of every θ_i, it is checked if the total time τ is already exceeded. The sampling stops as soon as $\sum_{j=1}^{i} \theta_j > \tau$. The last θ_i is then adjusted, such that in the end $\sum_{j=1}^{i} \theta_j = \tau$. [3]

4.2 Confidence Ellipse

To visualize the confidence ellipses for the estimators $\hat{\Gamma}_{10}$ and $\hat{\Gamma}_{01}$ the covariance matrix was used. For two variables x and y and N observations, the entries of the covariance matrix are calculated as follows

$$C = \frac{1}{N-1} \sum_{i=1}^{N} \begin{pmatrix} (x_i - \overline{x})^2 & (x_i - \overline{x})(y_i - \overline{y}) \\ (y_i - \overline{y})(x_i - \overline{x}) & (y_i - \overline{y})^2 \end{pmatrix}. \tag{11}$$

This covariance matrix is always real and symmetric. This implies that it has real eigenvalues $\lambda_{1,2}$ to two orthogonal eigenvectors $v_{1,2}$. These are calculated with the implemented `LinearAlgebra` package in Julia.
The center point of the ellipse is the point defined by the mean of both variables. The eigenvectors act as the two symmetric axes of the confidence ellipse. To draw it, the angle between the x-axis and v_1 is calculated.
The eigenvalues are converted to the two axes of the ellipse with $\sqrt{c \cdot \lambda_1}$ being the radius of the major axis and $\sqrt{c \cdot \lambda_2}$ the radius of the minor axis.
The factor c depends on the desired confidence and is determined using the χ^2 distribution implemented in Julia to read out the value for the corresponding quantile.
This is done by assuming the estimators to be normally distributed in both directions. The χ^2 distribution with two degrees of freedom is the distribution of the sum of the squares of two normal distributed variables. Therefore reading out the value of the χ^2 for two dimensions at the points 0.95 (go to Appendix A1 in line `141` to see how), gives the factor for our two confidence intervals together creating the 95% confidence ellipse.
The confidence ellipse can then be drawn with the following points:

$$\begin{pmatrix} x(t) \\ y(t) \end{pmatrix} = \begin{pmatrix} v_1 & v_2 \end{pmatrix} \cdot \begin{pmatrix} \sqrt{c \cdot \lambda_1} \cdot \cos(t) \\ \sqrt{c \cdot \lambda_2} \cdot \sin(t) \end{pmatrix} \tag{12}$$

The index of dispersion

$$D_j = \frac{\lambda_j}{\mu_j} \tag{13}$$

has also been calculated for each simulation where λ_j represents one of the two eigenvalues of the covariance matrix and μ_j the mean of the corresponding estimator $\hat{\Gamma}_j$ with $j \in \{10, 01\}$.

5 Results

Some examples of the simulation for different rates Γ_{10} and Γ_{01} and total time τ are shown below.

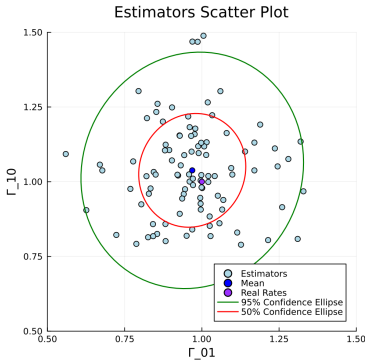

Figure 4: Scatter plot of 100 simulations for rates $\Gamma_{01} = \Gamma_{10}$ and total time $\tau \cdot \Gamma_{01} = 100$ with confidence ellipses of 50% and 95%. The indices of dispersion are $D_{10} = 2.4 \cdot 10^{-2}$ and $D_{01} = 2.1 \cdot 10^{-2}$.

One can see in Fig.4, that the calculated estimators $\hat{\Gamma}_{10}$ and $\hat{\Gamma}_{01}$ are not accurate for small total times τ. Therefore the simulations are done for bigger $\tau \cdot \Gamma_{01} = 200$ and 500 for three different combination of rates Γ_{10} and Γ_{01}.

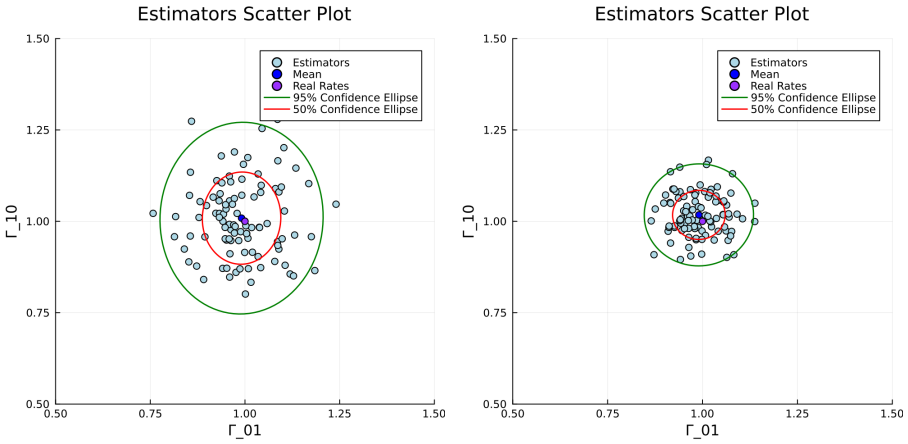

Figure 5: Scatter plot of 100 simulations for rates $\Gamma_{01} = \Gamma_{10}$ and total time $\tau \cdot \Gamma_{01} = 200$ (left) and 500 (right) with confidence ellipses of 50% and 95%. The indices of dispersion are $D_{10} = 1.2 \cdot 10^{-2}$ and $D_{01} = 0.9 \cdot 10^{-2}$ for $\tau \cdot \Gamma_{01} = 200$ and $D_{10} = 4.2 \cdot 10^{-3}$ and $D_{01} = 4.8 \cdot 10^{-3}$ for $\tau \cdot \Gamma_{01} = 500$.

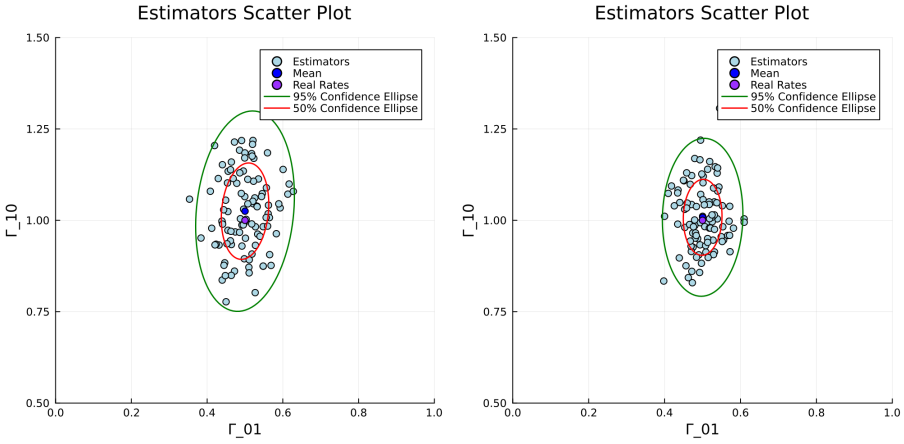

Figure 6: Scatter plot of 100 simulations for rates $\Gamma_{01} = \frac{1}{2}\Gamma_{10}$ and total time $\tau \cdot \Gamma_{01} = 200$ (left) and 500 (right) with confidence ellipses of 50% and 95%. The indices of dispersion are $D_{10} = 1.6 \cdot 10^{-2}$ and $D_{01} = 1.0 \cdot 10^{-2}$ for $\tau \cdot \Gamma_{01} = 200$ and $D_{10} = 7.3 \cdot 10^{-3}$ and $D_{01} = 2.8 \cdot 10^{-3}$ for $\tau \cdot \Gamma_{01} = 500$.

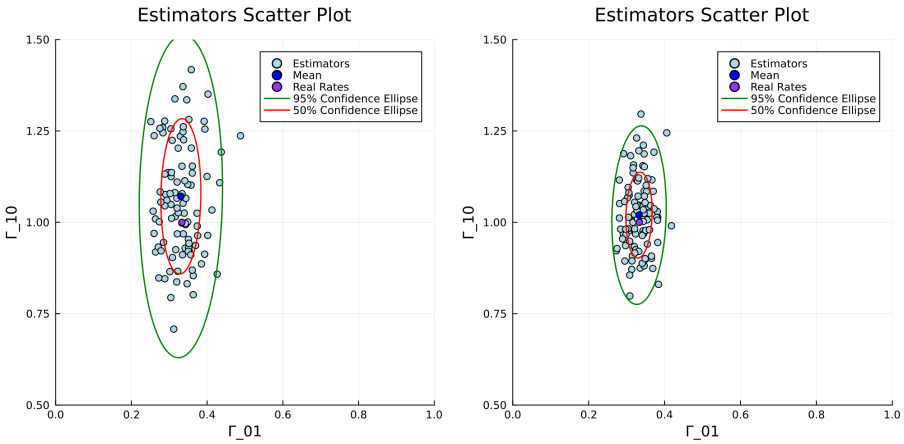

Figure 7: Scatter plot of 100 simulations for rates $\Gamma_{01} = \frac{1}{3}\Gamma_{10}$ and total time $\tau \cdot \Gamma_{01} = 200$ (left) and 500 (right) with confidence ellipses of 50% and 95%. The indices of dispersion are $D_{10} = 3.0 \cdot 10^{-2}$ and $D_{01} = 0.6 \cdot 10^{-2}$ for $\tau \cdot \Gamma_{01} = 200$ and $D_{10} = 9.2 \cdot 10^{-3}$ and $D_{01} = 2.4 \cdot 10^{-3}$ for $\tau \cdot \Gamma_{01} = 500$.

We can see in Fig.5 that the estimation gets a lot better for longer total times τ. Also there is no significant covariance and the accuracy of the estimators is similar for both Γ_{10} and Γ_{01}.

Looking at Fig.6 a drastic difference is seen as the confidence ellipse is way broader in Γ_{10}-direction (the bigger rate). Considering that in this simulation, the system takes up more time in state 1 than 0, this is expected. One can also observe a slight shift up in Γ_{10}-direction, meaning the estimator $\hat{\Gamma}_{10}$ will slightly overshoot the real rate Γ_{10} on average.

This is also more prevalent in Fig.7, where Γ_{10} is three times larger than Γ_{01}. Here the drift upwards in Γ_{10}-direction is clearly visible, even for long simulations with $\tau \cdot \Gamma_{01} = 500$ (corresponding to approximately 250 jumps).

6 Conclusions and Outlook

Basic two-level systems with similar rates can be estimated with measurements over a rather short time with a relative uncertainty of around $D = 1 \cdot 10^{-2}$ for both estimators.
When looking at two-level systems with very different rates (such as $\Gamma_{01} = \frac{1}{3}\Gamma_{10}$) there is almost unchanged uncertainty observed in the estimator for the larger rate, still around $D = 1 \cdot 10^{-2}$. The smaller rate can be estimated more accurately around $D = 5 \cdot 10^{-3}$. This is very likely due to the fact, that the system will spend more time e.g. in the excited state 1 when the smaller rate is Γ_{01} (the rate corresponding to the state change $1 \to 0$).
The overall estimation gets more precise when longer measurements of the system can be taken.
Future steps could include looking at time-dependent rates, expanding the system to three levels or including a coupling of two systems.
One could also look at fluorescence interactions, where the amount of energy transferred changes over time, resulting in distinct FRET states for different energy values and their corresponding transition rates. [1]

Appendix

A1 Julia Code

```
using Random, Plots, Statistics, LinearAlgebra, Distributions

#simulation for one trajectory with estimator calculation
function sim(cutofftime, gammafactor)
    r = AbstractFloat[]   #uniform dist to sample theta
    theta = AbstractFloat[]   #individual waiting times
    t = AbstractFloat[]   #individual total time (sum theta)
    step = 1
    tau = 0           #total time
    tau_1 = 0         #total time in state 1

    #helpvariables to plot state evolution
    plotstate = AbstractFloat[]
    plottime = AbstractFloat[]
    epsilon = 0.001

    #Predetermined Variables
    tau_tot = cutofftime      #cutoff time
    Gamma_10 = 1                #rate to jump 0->1
    Gamma_01 = gammafactor * Gamma_10 #rate to jump 1->0

    #Start State
    startprob = 0.5         #prob that start state is 1 (0.5=50%)
    startrand = rand()      #rand num to sample start state

    if startprob >= startrand
        state = 0
        k = 0      #number of times 0->1
        l = -1     #number of times 1->0

    elseif startprob < startrand
        state = 1
        k = -1
        l = 0
    end

    push!(plotstate, state)
    push!(plottime, 0)

    while tau < tau_tot
        #sample uniform dist number in [0,1)
        push!(r, rand())
        #sample theta based on current state
        if state == 0
            #convert to exp dist number -> WTD
```

```
            push!(theta, -1/Gamma_10*log(r[step]))
            l = l + 1
        elseif state == 1
            push!(theta, -1/Gamma_01*log(r[step]))
            tau_1 = tau_1 + theta[step]
            k = k + 1
        end
        #total time
        if tau + theta[step] < tau_tot
            tau = tau + theta[step]
            #save total time
            push!(t, tau)

            #adjust arrays for state evolution plot
            push!(plottime, tau)
            push!(plottime, tau + epsilon)
            push!(plotstate, state)

            #increase step
            step = step + 1

            #change state
            state = mod(state + 1, 2)

            #adjust arrays for state evolution plot
            push!(plotstate, state)

        elseif tau + theta[step] >= tau_tot
            #adjust last theta to end when total time ends
            theta_temp = theta[step]
            if state == 1
                tau_1 = tau_1 - theta[step] + theta_temp
            end
            #save final theta
            theta[step] = tau_tot - tau
            #save total time
            tau = tau_tot

            #adjust arrays for state evolution plot
            push!(t, tau)
            push!(plottime, tau)
            push!(plotstate, state)
        end
    end

    #Calculate estimators
    Gamma_10_est = k/(tau-tau_1)
    Gamma_01_est = l/tau_1

    #output estimators
```

```
    return (Gamma_01_est, Gamma_10_est)

    #output for when sim is only run once to see state evolution plot
    println("number of jumps: ", length(t)-1)
    println("jumps up/down: (k, l)= (", k, ", ", l, ")")
    println("time in state 1: tau_1=", tau_1)
    println("Estimators:")
    println("Gamma_10 = ", Gamma_10_est)
    println("Gamma_01 = ", Gamma_01_est)
    plot(plottime, plotstate)
    ylims!(-0.3, 1.3)
    xlims!(0,tau_tot)
    title!("State Evolution")
    xlabel!("time")
    ylabel!("state")
end

#simulation of multiple trajectories with scatter plot
function multisim(cutofftime, gammafactor, num_of_sims)
    Gamma_01_estimators = AbstractFloat[]
    Gamma_10_estimators = AbstractFloat[]

    for j = 1:num_of_sims
        (Gamma_01_est, Gamma_10_est) = sim(cutofftime, gammafactor)
        push!(Gamma_01_estimators, Gamma_01_est)
        push!(Gamma_10_estimators, Gamma_10_est)
    end
    #println(Gamma_01_estimators)
    #println(Gamma_10_estimators)

    mean01 = mean(Gamma_01_estimators)
    mean10 = mean(Gamma_10_estimators)

    # Calculate Covarince-Matrix
    C = cov_mat(Gamma_01_estimators, Gamma_10_estimators)

    # Calculate Eigenvalues and Eigenvectors for estimation ellipse
    (lambda_1, lambda_2) = eigvals(C);
    v_1 = sqrt(lambda_1)*eigvecs(C)[:,1];
    v_2 = sqrt(lambda_2)*eigvecs(C)[:,2];

    # Calculate Angle for estimation ellipse
    angle = acos(dot([1,0],v_1)/norm(v_1));

    #getting confidence factors from Chi-Squared distribution
    c_95 = quantile(Chisq(2), 0.95)
    c_50 = quantile(Chisq(2), 0.50)

    #println(dot(v_1,v_2)) #check that eigenvectors are normal to each
        other
```

```
    println(mean01," / ", mean10)
    println("Cov-Matrix: ", C)
    #plot(quiver([mean01,mean10], [mean01,mean10], quiver=(v_1, v_2)))
    scatter(Gamma_01_estimators, Gamma_10_estimators, mc=:lightblue,
        label="Estimators", size=(500,500), dpi=1000)
    scatter!([gammafactor], [1.0], mc=:blue, label="Real Rates")
    plot!(getellipsepoints(mean01, mean10, sqrt(c_95*lambda_1),
        sqrt(c_95*lambda_2), angle), linewidth=1.5, linecolor=:green,
        label="95% Confidence Ellipse")
    plot!(getellipsepoints(mean01, mean10, sqrt(c_50*lambda_1),
        sqrt(c_50*lambda_2), angle), linewidth=1.5, linecolor=:red,
        label="50% Confidence Ellipse")
    ylims!(0.5, 1.5)
    xlims!(0, 1)
    title!("Estimators Scatter Plot")
    xlabel!("Gamma_01")
    ylabel!("Gamma_10")

    savefig("scatter_500_13.png")
end

#covariance matrix calculations
function cov_mat_entry(a,b)
    N = length(a);
    sum = 0;
    for i = 1:N
        sum = sum + (a[i]-mean(a))*(b[i]-mean(b));
    end
    sum = 1/(N-1)*sum;
    return sum
end
function cov_mat(x,y)
    cov_11 = cov_mat_entry(x,x);
    cov_12 = cov_mat_entry(x,y);
    cov_21 = cov_mat_entry(y,x);
    cov_22 = cov_mat_entry(y,y);
    return [[cov_11 cov_12]; [cov_21 cov_22]]
end

#tilted ellipse points calculation
# cx: x-position of the center
# cy: y-position of the center
# rx: major radius
# ry: minor radius
# ang: angle to x-axis
function getellipsepoints(cx, cy, rx, ry, ang)
   t = range(0, 2*pi, length=100)
   ellipse_x_r = @. rx * cos(t)
   ellipse_y_r = @. ry * sin(t)
   R = [cos(ang) sin(ang); -sin(ang) cos(ang)]
```

```
    r_ellipse = [ellipse_x_r ellipse_y_r] * R
    x = @. cx + r_ellipse[:,1]
    y = @. cy + r_ellipse[:,2]
    (x,y)
end


#run simulation with given parameters
multisim(500, 1/3, 100)
#inputs:
#cutofftime, gammafactor, number_of_sims
```

References

[1] Taekjip Ha et al. 2024 Fluorescence resonance energy transfer at the single-molecule level *Nat Rev Methods Primers* 4, 21
https://doi.org/10.1038/s43586-024-00298-3

[2] Julia Boeyens et al. 2023 Probe thermometry with continuous measurements *New J. Phys.* 25 123009

[3] Dr. Luca Donati "Stochastic and Diffusive Processes" Lecture 4b: method of generating functions and Gillespie algorithm WISE2324 Berlin

[4] function "getellipsepoints" for calculating ellipse points
from user @filchristou 2022 on Julia Forum
https://discourse.julialang.org/t/plot-ellipse-in-makie/82814
(accessed 13.04.2025)