Wir wollen das folgende nichtlineare Gleichungssystem lösen:

$$\begin{aligned} 1-\alpha &= e^{-t_1} - e^{-t_2}, \quad \alpha \in (0,1) \text{ gegeben} \\ t_1 e^{-t_1} &= t_2 e^{-t_2} \end{aligned}$$

Wir schreiben dieses Gleichungssystem in der Form

$$\boldsymbol{F}(\boldsymbol{x}) = \boldsymbol{0},$$

mit

$$\boldsymbol{F} : \mathbb{R}^2 \to \mathbb{R}^2, \quad \boldsymbol{F}(\boldsymbol{x}) := \begin{pmatrix} 1-\alpha-e^{-x_1}+e^{-x_2} \\ x_1 e^{-x_1} - x_2 e^{-x_2} \end{pmatrix}.$$

Solche nichtlinearen Gleichungssysteme löst man üblicherweise numerisch mit dem Newton-Verfahren. Die Studierenden kennen dieses Verfahren aus Numerik I (2. Semester). Das iterative Verfahren funktioniert wie folgt:

$$\begin{aligned} \boldsymbol{x}_0 &\quad \text{gegeben,} \\ \boldsymbol{x}_{n+1} &= \boldsymbol{x}_n - \left(\underline{\boldsymbol{dF}}(\boldsymbol{x}_n)\right)^{-1} \boldsymbol{F}(\boldsymbol{x}_n), \quad n \geq 0. \end{aligned}$$

Die Vektoren $\boldsymbol{x}_n$ konvergieren sehr schnell (quadratisch) gegen die exakte Lösung $\boldsymbol{x}$ für $n \to \infty$, falls der Startvektor $\boldsymbol{x}_0$ genügend nahe bei der Lösung $\boldsymbol{x}$ liegt. Die Jacobi-Matrix $\underline{\boldsymbol{dF}}$ ist hier gegeben durch

$$\underline{\boldsymbol{dF}}(\boldsymbol{x}) = \begin{pmatrix} e^{-x_1} & -e^{-x_2} \\ (1-x_1)e^{-x_1} & (x_2-1)e^{-x_2} \end{pmatrix}.$$

Wie in der Numerik üblich, berechnen wir den Ausdruck $\boldsymbol{y}_n := \left(\underline{\boldsymbol{dF}}(\boldsymbol{x}_n)\right)^{-1} \boldsymbol{F}(\boldsymbol{x}_n)$ nicht durch explizite Inversion der Jacobi-Matrix $\underline{\boldsymbol{dF}}$, sondern als Lösung des linearen Gleichungssystems

$$\underline{\boldsymbol{dF}}(\boldsymbol{x}_n)\boldsymbol{y}_n = \boldsymbol{F}(\boldsymbol{x}_n).$$

Jetzt zur Implementierung: Weil wir wissen, dass $x_2 > x_1$, wählen wir $\boldsymbol{x}_0 = (0,1)^\top$ und rechnen bis $n = 20$. In **R** können wir das Verfahren wie folgt implementieren (hier für $\alpha = 0.05$):

```
> alpha<-0.05
> N<-20
> x<-c(0,1)
> for (i in 1:N) {
+ F<-c(1-alpha-exp(-x[1])+exp(-x[2]),x[1]*exp(-x[1])-x[2]*exp(-x[2]))
+ dF<-matrix(c(exp(-x[1]),-exp(-x[2]),(1-x[1])*exp(-x[1]),
+ (x[2]-1)*exp(-x[2])),2,2,byrow=TRUE)
+ x<-x-solve(dF,F)
+ }
> x
```

In **Matlab** sieht das Verfahren so aus (wieder für $\alpha = 0.05$):

```
alpha=0.05;
N=20;
x=[0;1];
for i=1:N
    F=[1-alpha-exp(-x(1))+exp(-x(2));x(1)*exp(-x(1))-x(2)*exp(-x(2))];
    dF=[exp(-x(1)) -exp(-x(2));(1-x(1))*exp(-x(1)) (x(2)-1)*exp(-x(2))];
    x=x-dF\F;
end
x
```

In **Maple** können wir das Problem wie folgt direkter lösen:

```
alpha=0.05:
solve({1-alpha-exp(-x)+exp(-y),x*exp(-x)-y*exp(-y)},{x,y});
```

Auch Maple verwendet wahrscheinlich ein Newton-Verfahren zur Lösung. Für $\alpha = 0.1$ erhält Maple die unsinnige Lösung $t_1 = t_2 \simeq -262$, was wohl auf einen ungünstig gewählten Startvektor zurück zu führen ist.
Wir erhalten die folgenden Lösungen:

α	t_1	t_2
0.001	0.000903	9.23
0.01	0.00873	6.64
0.05	0.0424	4.77
0.1	0.0838	3.93
0.2	0.167	3.08