Theory

The paper discusses idea of balanced amplification using a linear firing rate model. The model is described through following equation:

$$T\frac{d\vec{\mathbf{r}}}{dt} = -\vec{\mathbf{r}} + \mathbf{W}\vec{\mathbf{r}} + \mathbf{I} = -(\mathbf{1} - \mathbf{W})\vec{\mathbf{r}} + \mathbf{I}$$

With $\vec{\mathbf{r}} \in \mathbb{R}^N$ representing the firing rates of N neurons, $\mathbf{W} \in \mathbb{R}^{N\times N}$ a connectivity matrix, and $\mathbf{I}$ being the feedforward input from 'external' neurons.

The paper proceeds by analysing the simples example: $\vec{\mathbf{r}} = \begin{bmatrix} r_E \\ r_I \end{bmatrix}$ and $\mathbf{W} = \begin{bmatrix} w & -k_I w \\ w & -k_I w \end{bmatrix}$ Here r_E is the excitatory and r_I the inhibitory input.

After clarifying these concepts the paper proceeds to plot the solutions of these differential equations for different values of k, l and the respective time constant, showing that for certain values the input gets amplified regardless of strong inhibition.

This notebook derives these parts of the results shown in the paper, to be precise, figures 1 and 2.

In the next cell the w for balanced amplification can be seen/used.

```
wb = 4.285714285714286
```

Parameters

Changing these parameters accordingly to the descriptions in the paper leads to the theoretical plots shown.

Some values mentioned in the paper are $w \in [0.75, 4\frac{2}{7}, 3/4, 0.9, 2.5, 90]$, $k = 1.1$.

The boundary conditions are the following: $\vec{\mathbf{r}}(0) = \begin{bmatrix} r_E(0) \\ r_I(0) \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix}$.

Synaptic weight (w) : 0.0
Inhibition factor (kᵢ) : 1.1
Time constant (τ) : 1.0

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

```
#sigmoid function
σ(x) = 1/(1+exp(-x))
```

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

```
#dirac delta
δ(t) = if t == 0 0. else 0. end
```

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

```julia
#theta
θ(t) = if t<0 0 else 1. end
```

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

```julia
#external inputs maybe this is useless
begin
    Iₑ(t) =  δ(t);
    Iᵢ(t) = 0.;
end
```

```
DiscreteCallback(condition (generic function with 1 method), affect! (generic function w
```

```julia
#Callbacks to set inputs. δ functions somehow get ignored if we add them manually
begin
    #a condition when the callsbacks gonna be called
    condition(u,t,integrator) = t==0
    #affect the inetgrator
    function affect!(integrator)
        integrator.u[1] += .5
        integrator.u[2] += .5
    end
    #callback
    cb = DiscreteCallback(condition,affect!)
end
```

```
balancedRNN! (generic function with 1 method)
```

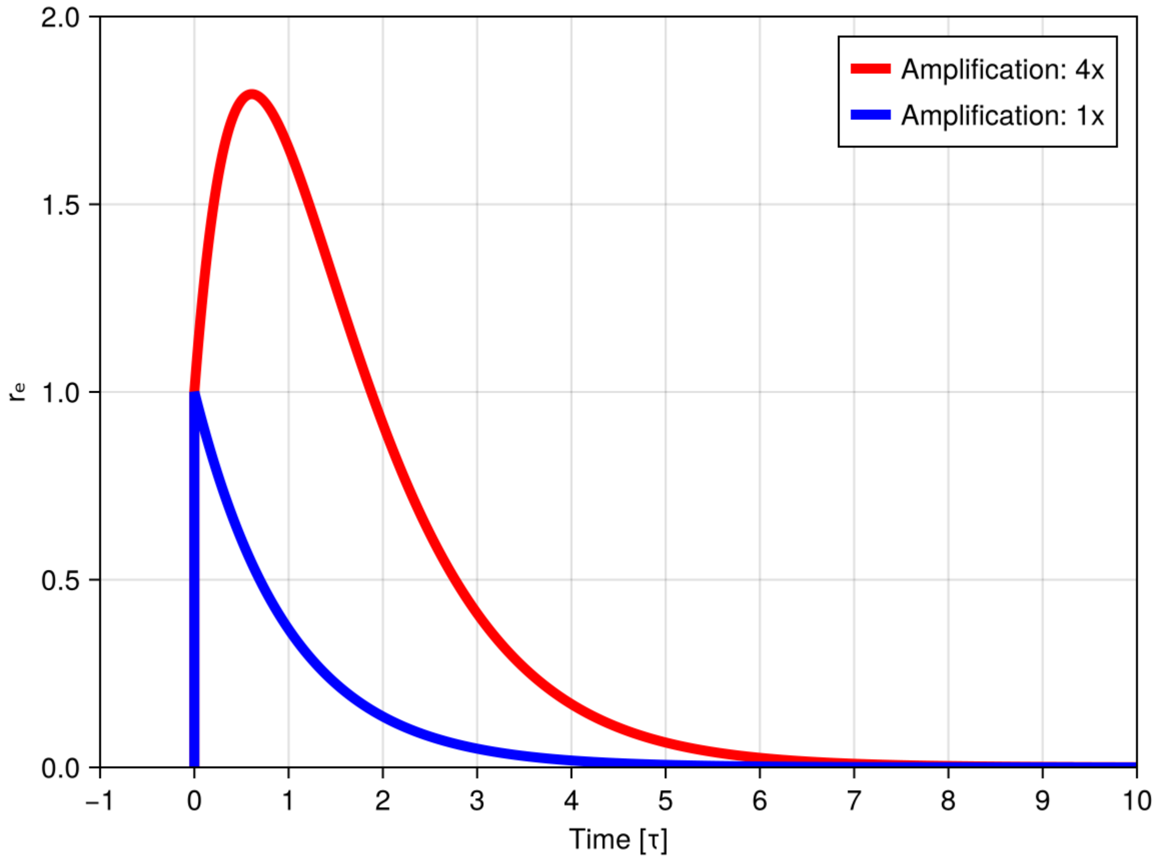
```julia
#ODE System for two populations of neurons
function balancedRNN!(du, u, p, t)
    du[1] = ((p[1] - p[2]*p[1] - 1)*u[1]+p[1]*(p[2]+1)*u[2]+(Iᵢ(t)+Iₑ(t))/2)/p[3]
    du[2] = (-u[2] + (Iₑ(t)-Iᵢ(t))/2)/p[3]
end
```

```julia
#solution for the ODE with two different parameters, using RK4
begin
    #meh
    u0 = [.0, .0]
    tspan =(-1., 10)

    #noice
    p1 = [0, kᵢ, τ]
    prob1 = ODEProblem(balancedRNN!, u0, tspan, p1)
    solvbal = solve(prob1, Tsit5(), dt=0.01, dtmax=0.001, callback=cb, tstops=[0.0])

    p2 = [wb, kᵢ, τ]
    prob2 = ODEProblem(balancedRNN!, u0, tspan, p2)
    solvamp = solve(prob2, Tsit5(), dtmax=0.001, callback=cb, tstops=[0.0])
end;
```

```
Makie.Legend()
```



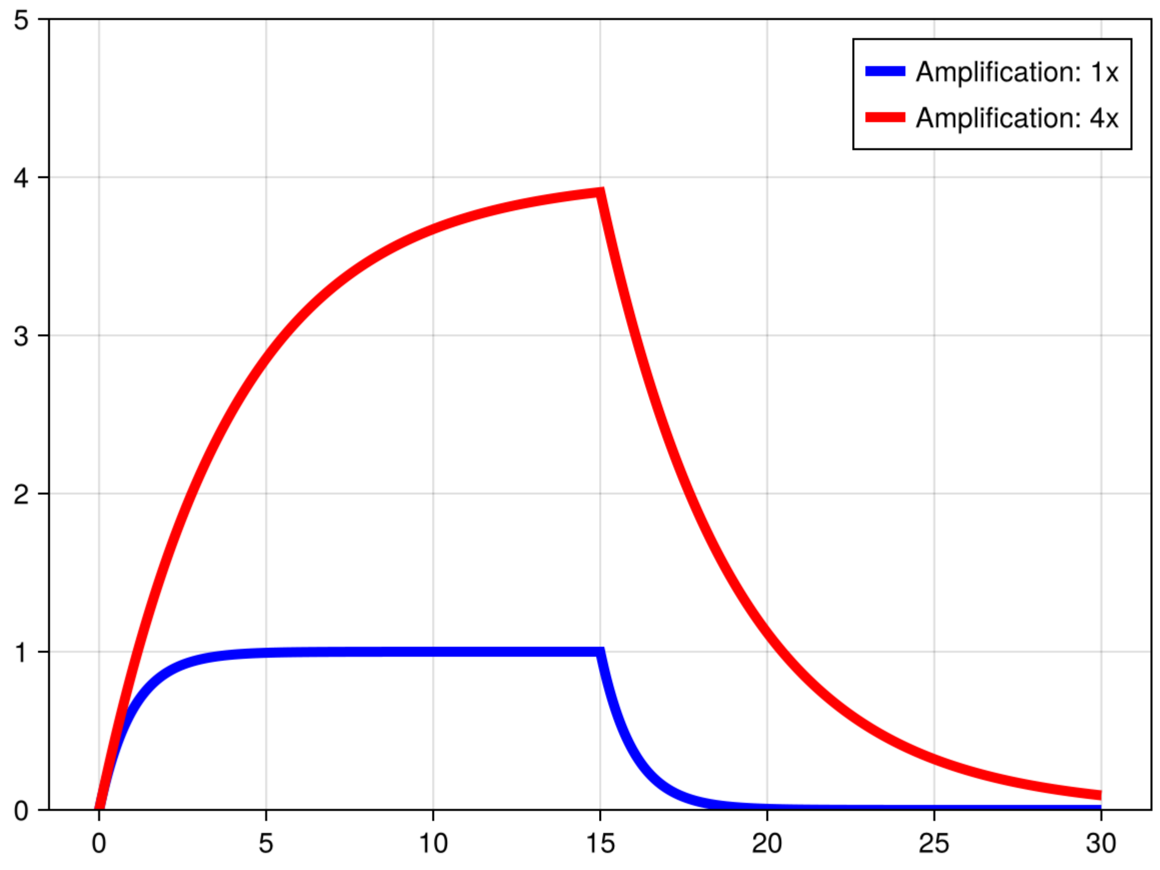
```julia
#save("fig2Bcairo.pdf", fig1)
```

```julia
#Integral of red plot should be 4x integral blue thing
begin
    s1 = sum(y1)*0.001;
    s2 = sum(y2)*0.001
    isit = isapprox(s1, 4*s2, rtol=0.01)
    print(if isit "Integrals are good, amplification OK" else "Integrals are not good, amplification WRONG" end)

end
```

```
Integrals are good, amplification OK
```

Hebbian case



```
Integrals OK, amplification OK
```

```
Enter cell code...
```