Hill Equation

The following code describes the stability analysis for cooperative inhibition using a given set of parameters for the Hill equation:

$$\frac{dS_1}{dt} = \frac{k_1}{1 + (\frac{S_2}{M_2})^{n_1}} - k_3 S_1$$

$$\frac{dS_2}{dt} = \frac{k_2}{1 + (\frac{S_1}{M_1})^{n_2}} - k_4 S_2$$

Here, the parameters that are given are $n_1 = n_2 = 2, k_1 = k_2 = 20, M_1 = M_2 = 1, k_3 = k_4 = 5$, with initial conditions $S_1(0) = 3$ and $S_2(0) = 1$.

In order to do this in a code, we need to first load the required libraries:

```
library(deSolve)
library(rootSolve)
```

The Hill equation needs to be defined in a function. Variables have to be created to store the initial values, parameters, and time range for the solution.

```
# Define Hill equation function with t, initial values, parameter
hill = function(t, init, para){
  S1 <- init[1]
  S2 <- init[2]
  n1 <- para[1]
  n2<- para[2]
  k1 <- para[3]
  k2 <- para[4]
  M1 <- para[5]
  M2 <- para[6]
  k3 <- para[7]
  k4 <- para[8]
  dS1 = (k1 / (1 + ((S2/M2)^n1))) - (k3 * S1) # dS1/dt
  dS2 = (k2 / (1 + ((S1/M1)^n2))) - (k4 * S2) # dS2/dt
  list(c(dS1, dS2)) # function automatically returns dS1, dS2 as a list
}
# Give initial values - S1(t=0), S2(t=0)
init <- c(3, 1)
# Give parameter value -  n1, n2, k1, k2, M1, M2, k3, k4
para <- c(2, 2, 20, 20, 1, 1, 5, 5)
# Time range for solution - till 6.7 - found out when calculated till 100
t <- seq(0, 6.7, 0.01)
```

Numerical Solution

The first step in stability analysis is to plot the numerical solution of each equation once you have substituted the parameters.

Substituting the parameters in the Hill equation gives us the following equations:

$$\frac{dS_1}{dt} = \frac{20}{1 + S_2^2} - 5S_1$$

$$\frac{dS_2}{dt} = \frac{20}{1 + S_1^2} - 5S_2$$

The R package *deSolve* contains a function that will calculate the numerical solutions of both the differential equations, given the initial conditions, parameters, and time range of the solution.

```
# Calling ODE function
out <- ode(y = init, times = t, func = hill, parms = para)
colnames(out) <- c("Time", "S1", "S2")
```

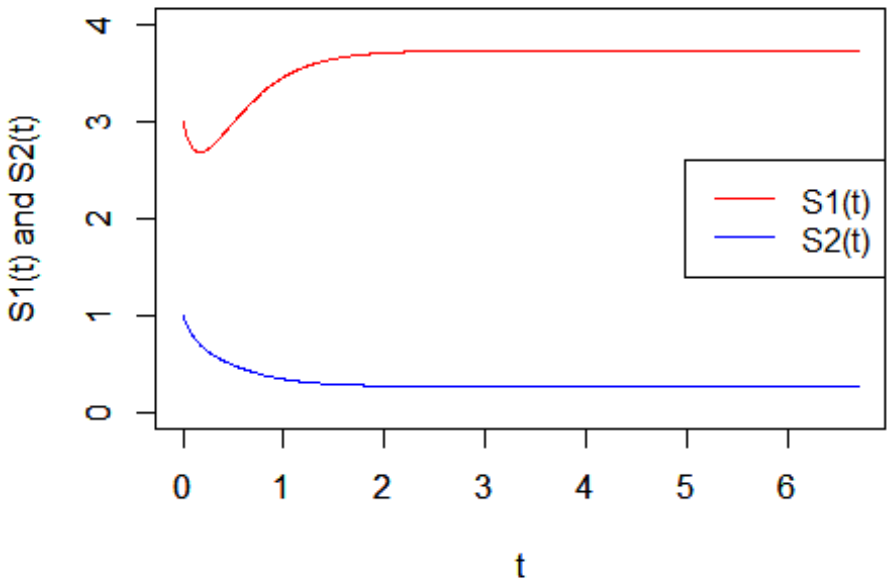
A matrix named *out* is created which contains the numerical solutions of both the differential equations. You can view it using the following command:

```
head(out)
```

```
##      Time       S1        S2
## [1,] 0.00 3.000000 1.0000000
## [2,] 0.01 2.952681 0.9710246
## [3,] 0.02 2.910486 0.9440109
## [4,] 0.03 2.873030 0.9188158
## [5,] 0.04 2.839964 0.8953101
## [6,] 0.05 2.810953 0.8733684
```

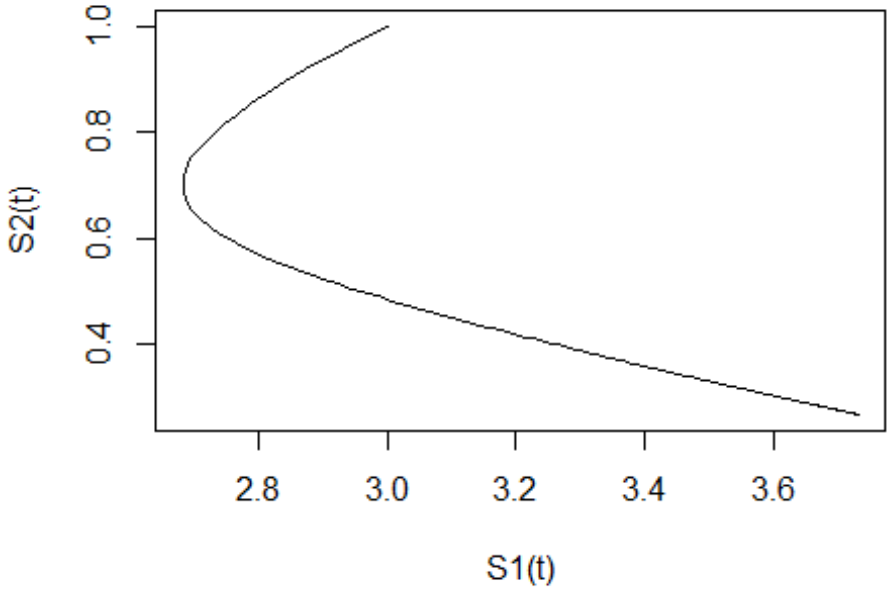
We can plot the solutions using the following code:

```
# Plotting solution for S1(t)
plot(out[,1], out[,2], type = "l", xlab = "t", ylab = "S1(t) and S2(t)", col
= "red", ylim = c(0,4))
par(new=TRUE)
# Plotting solution for S2(t)
plot(out[,1], out[,3], type = "l", xlab = "t", ylab = "S1(t) and S2(t)", col
= "blue", ylim = c(0,4))
legend("right", col = c("red", "blue"), legend = c("S1(t)", "S2(t)"), lty =
1)
```



The **phase plot** will tell us if the system is stable or not at the points of equilibrium. It consists of plotting all the values of $S_1(t)$ against $S_2(t)$. If the curve is closed, the system is stable at all points of equilibrium. If the curve is open, the system is unstable at some points of equilibrium. (See section on **vector field**).

```
# Plotting S1(t) vs S2(t)
plot(out[,2], out[,3], type = "l", xlab = "S1(t)", ylab = "S2(t)")
```



As we can see, this system is unstable for certain points at equilibrium. We will go into more detail in the next section.

Stability Analysis

We will now perform stability analysis for each of the equilibrium points that we will find, as well as for the given initial conditions of S_1 and S_2.

Visually, we can plot the nullclines to identify equilibrium points.

Nullclines

Consider the autonomous system:

$$\frac{dx}{dt} = f(x, y)$$

$$\frac{dy}{dt} = g(x, y)$$

The **x-nullcline** is the set of points where $\frac{dx}{dt} = 0$ and the **y-nullcline** is the set of points where $\frac{dy}{dt} = 0$.

Thus, **the points of intersection between the x-nullcline and the y-nullcline are the equilibrium points**.

Nullclines do not necessarily have to be straight lines. They can be curves, as we will see in our solution of the Hill equation below.

Coming back to our question,

$$\frac{20}{1+S_2^2} - 5S_1 = 0$$

$$\frac{20}{1+S_1^2} - 5S_2 = 0$$

This can be written as,

$$S_1 = \frac{4}{1+S_2^2}$$

$$S_2 = \frac{4}{1+S_1^2}$$

In order to plot them in R, I will replace S_1 with y and S_2 with x. Rewriting the above equations,

$$y = \frac{4}{1+x^2}$$

is the equation for S_2.

$y = \sqrt{\frac{4}{x} - 1}$ and

$$y = -\sqrt{\frac{4}{x} - 1}$$

are the equations that collectively describe S_1.

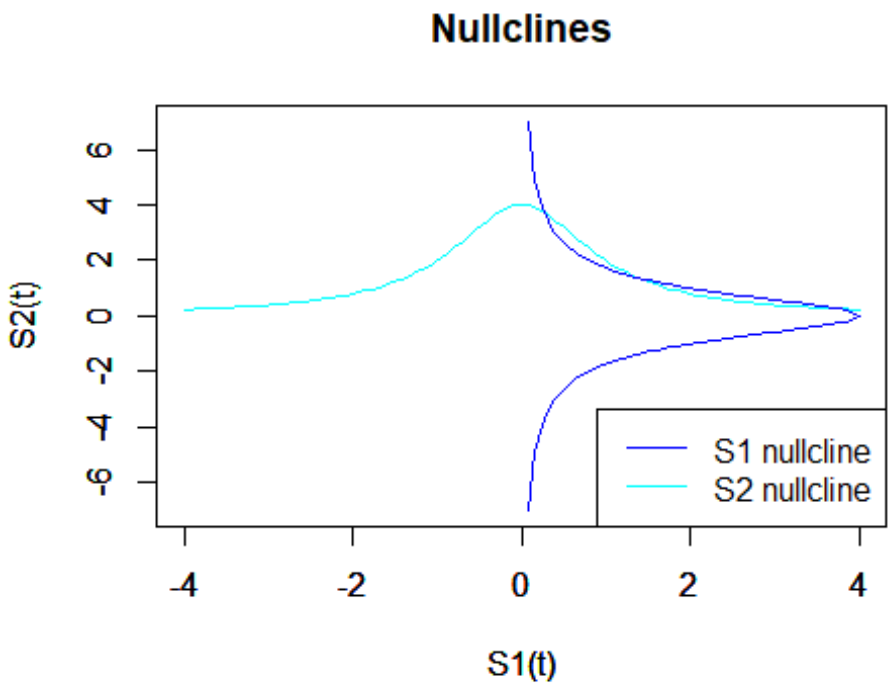
```
# Plotting dS2/dt = 0
eqn1 <- function(x){4/(1 + (x^2))}

# Plotting dS1/dt = 0; positive half
eqn2 <- function(x){sqrt((4/x)-1)}

# Plotting dS1/dt = 0; negative half
eqn3 <- function(x){-1*sqrt((4/x)-1)}

curve(eqn1(x), -4, 4, ylim=c(-7,7), xlab = "S1(t)", ylab = "S2(t)",
main="Nullclines", col = "cyan")
par(new=TRUE)
curve(eqn2(x), -4, 4, ylim=c(-7,7), col = "blue", xlab = "S1(t)", ylab =
"S2(t)")
par(new=TRUE)
```

```
curve(eqn3(x), -4, 4, ylim=c(-7,7), col = "blue", xlab = "S1(t)", ylab =
"S2(t)")
legend("bottomright", col = c("blue", "cyan"), legend = c("S1 nullcline", "S2
nullcline"), lty = 1)
```



By looking at the nullclines, we can see that there are 3 points of intersection. This means there are 3 equilibrium points.

Finding Equilibrium Points

To find the points S_1 and S_2 at which the system is in equilibrium, we have to equate the differential equations to 0 and solve them.

$$\frac{20}{1+S_2^2} - 5S_1 = 0$$

$$\frac{20}{1+S_1^2} - 5S_2 = 0$$

We can use functions in the R package *rootSolve* to find out the points of equilibrium. We know approximately where the two nullclines intersect. In the *multiroot* function, we have to enter the approximate values of the points of intersection in order to find out the actual points.

```
# Defining the equations in a function named "model"
model = function(x){
  F1 <- (20/(1+x[2]^2)) - (5*x[1])
```

```r
  F2 <- (20/(1+x[1]^2)) - (5*x[2])
  c(F1 = F1, F2 = F2)
}
# Finding roots
ss1 = multiroot(f=model, start=c(0,0))
print(ss1)
```

```
## $root
## [1] 1.378797 1.378797
##
## $f.root
##           F1           F2
## 3.606493e-10 3.606493e-10
##
## $iter
## [1] 7
##
## $estim.precis
## [1] 3.606493e-10
```

```r
ss2 = multiroot(f=model, start=c(1,3))
print(ss2)
```

```
## $root
## [1] 0.2679492 3.7320509
##
## $f.root
##            F1            F2
##  1.017053e-07 -2.037010e-07
##
## $iter
## [1] 5
##
## $estim.precis
## [1] 1.527031e-07
```

```r
ss3 = multiroot(f=model, start=c(3,1))
print(ss3)
```

```
## $root
## [1] 3.7320509 0.2679492
##
## $f.root
##            F1            F2
## -2.036568e-07  1.017014e-07
##
## $iter
## [1] 5
##
## $estim.precis
## [1] 1.526791e-07
```

The roots (values of S_1 and S_2 respectively) are displayed under *'roots'*. *'f.root'* shows the values of the functions when the roots are substituted in them. Verify that the values are zero or close to zero.

Eigen Values

The way we find out the stability of the system at initial conditions or at equilibrium is by finding the Eigen values of the Jacobian matrix.

Consider the autonomous system:

$$\frac{dx}{dt} = f(x, y)$$

$$\frac{dy}{dt} = g(x, y)$$

If f and g are linear functions of time t (or if they are not, linearize them),

$$\frac{dx}{dt} = a_{11}x + a_{12}y + u_1$$

$$\frac{dy}{dt} = a_{21}x + a_{22}y + u_2$$

where u_1 and u_2 are constants.

In matrix form,

$$\begin{bmatrix} \frac{dx}{dt} \\ \frac{dy}{dt} \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

This can be written with the derivative function outside,

$$\frac{d}{dt}\begin{bmatrix} x \\ y \end{bmatrix} = \begin{bmatrix} a_{11} & a_{12} \\ a_{21} & a_{22} \end{bmatrix} \begin{bmatrix} x \\ y \end{bmatrix} + \begin{bmatrix} u_1 \\ u_2 \end{bmatrix}$$

This can be written in the form of a rotation vector,

$$\frac{d\overline{X}}{dt} = \overline{AX} + \overline{U}$$

Here, $\overline{A}$ is called the **Jacobian matrix**.

Under steady state,

$$\frac{d\overline{X}}{dt} = \overline{AX} + \overline{U} = 0$$

$$\overline{AX} + \overline{U} = 0$$

$\overline{A}^{-1}\overline{AX} + \overline{A}^{-1}\overline{U} = 0$

$\overline{X} = \overline{A}^{-1}\overline{U}$ is the solution.

For our example of the autonomous system, a **Jacobian matrix is constructed in the following way**:

$$a_{11} = \frac{\partial f}{\partial x}$$

$$a_{12} = \frac{\partial f}{\partial y}$$

$$a_{21} = \frac{\partial g}{\partial x}$$

$$a_{22} = \frac{\partial g}{\partial y}$$

For the stability analysis, we have to find the Eigen values of the Jacobian matrix, which are given by:

$$\lambda_1, \lambda_2 = \frac{Tr(\overline{A})}{2} \pm \sqrt{\frac{[Tr(\overline{A})]^2}{4} - Det(\overline{A})}$$

Interpretation

1. If the Eigen values are both real and negative, the system is stable for the given conditions of $x(t)$ and $y(t)$. The trajectory is called a stable node.
2. If the Eigen values are both real and positive, the system is unstable for the given conditions of $x(t)$ and $y(t)$. The trajectory is called an unstable node.
3. If the Eigen values are both real but one is positive and one is negative, the system is unstable for the given conditions of $x(t)$ and $y(t)$. The trajectory is called a saddle point.
4. If the Eigen values are both complex and the real parts are positive, the system is oscillatory and unstable for the given conditions of $x(t)$ and $y(t)$. The trajectory is called an unstable focus.
5. If the Eigen values are both complex and the real parts are negative, the system is oscillatory and stable for the given conditions of $x(t)$ and $y(t)$. The trajectory is called a stable focus.

Continuing with the Hill equation:

```
# Printing Eigen values for the initial conditions
lambdas <- eigen(jacobian.full(y = init, func = hill,
                               parms = para))$values
print(lambdas)
```

```
## [1] -8.464102 -1.535898

if(is.complex(lambdas) == FALSE && lambdas[1] < 0 && lambdas[2] < 0){
  print("Stable node - system is stable for given conditions.")
  } else if(is.complex(lambdas) == FALSE && lambdas[1] > 0 && lambdas[2] >
0){
  print("Unstable node - system is unstable for given conditions.")
  } else if(is.complex(lambdas) == FALSE && lambdas[1] > 0 && lambdas[2] <
0){
    print("Unstable node - saddle point - system is unstable for given
conditions.")
  } else if(is.complex(lambdas) == FALSE && lambdas[1] < 0 && lambdas[2] >
0){
    print("Unstable node - saddle point - system is unstable for given
conditions.")
  } else if(is.complex(lambdas) == TRUE && Re(lambdas[1]) > 0 &&
Re(lambdas[2]) > 0){
    print("Unstable focus - system is oscillatory and unstable.")
  } else if(is.complex(lambdas) == TRUE && Re(lambdas[1]) < 0 &&
Re(lambdas[2]) < 0){
    print("Stable focus - system is oscillatory and stable.")
  }

## [1] "Stable node - system is stable for given conditions."

# Printing Eigen values for equilibrium point 1: (1.378,1.378)
lambdas <- eigen(jacobian.full(y = c(1.378,1.378), func = hill,
                          parms = para))$values
print(lambdas)

## [1]   1.55915 -11.55915

if(is.complex(lambdas) == FALSE && lambdas[1] < 0 && lambdas[2] < 0){
  print("Stable node - system is stable for given conditions.")
  } else if(is.complex(lambdas) == FALSE && lambdas[1] > 0 && lambdas[2] >
0){
  print("Unstable node - system is unstable for given conditions.")
  } else if(is.complex(lambdas) == FALSE && lambdas[1] > 0 && lambdas[2] <
0){
    print("Unstable node - saddle point - system is unstable for given
conditions.")
  } else if(is.complex(lambdas) == FALSE && lambdas[1] < 0 && lambdas[2] >
0){
    print("Unstable node - saddle point - system is unstable for given
conditions.")
  } else if(is.complex(lambdas) == TRUE && Re(lambdas[1]) > 0 &&
Re(lambdas[2]) > 0){
    print("Unstable focus - system is oscillatory and unstable.")
  } else if(is.complex(lambdas) == TRUE && Re(lambdas[1]) < 0 &&
Re(lambdas[2]) < 0){
```

```
    print("Stable focus - system is oscillatory and stable.")
  }

## [1] "Unstable node - saddle point - system is unstable for given
conditions."

# Printing Eigen values for equilibrium point 2: (0.267,3.732)
lambdas <- eigen(jacobian.full(y = c(0.267,3.732), func = hill,
                          parms = para))$values
print(lambdas)

## [1] -7.496797 -2.503203

if(is.complex(lambdas) == FALSE && lambdas[1] < 0 && lambdas[2] < 0){
  print("Stable node - system is stable for given conditions.")
  } else if(is.complex(lambdas) == FALSE && lambdas[1] > 0 && lambdas[2] >
0){
  print("Unstable node - system is unstable for given conditions.")
  } else if(is.complex(lambdas) == FALSE && lambdas[1] > 0 && lambdas[2] <
0){
    print("Unstable node - saddle point - system is unstable for given
conditions.")
  } else if(is.complex(lambdas) == FALSE && lambdas[1] < 0 && lambdas[2] >
0){
    print("Unstable node - saddle point - system is unstable for given
conditions.")
  } else if(is.complex(lambdas) == TRUE && Re(lambdas[1]) > 0 &&
Re(lambdas[2]) > 0){
    print("Unstable focus - system is oscillatory and unstable.")
  } else if(is.complex(lambdas) == TRUE && Re(lambdas[1]) < 0 &&
Re(lambdas[2]) < 0){
    print("Stable focus - system is oscillatory and stable.")
  }

## [1] "Stable node - system is stable for given conditions."

# Printing Eigen values for equilibrium point 3: (3.732, 0.267)
lambdas <- eigen(jacobian.full(y = c(3.732, 0.267), func = hill,
                          parms = para))$values
print(lambdas)

## [1] -7.496797 -2.503203

if(is.complex(lambdas) == FALSE && lambdas[1] < 0 && lambdas[2] < 0){
  print("Stable node - system is stable for given conditions.")
  } else if(is.complex(lambdas) == FALSE && lambdas[1] > 0 && lambdas[2] >
0){
  print("Unstable node - system is unstable for given conditions.")
  } else if(is.complex(lambdas) == FALSE && lambdas[1] > 0 && lambdas[2] <
0){
    print("Unstable node - saddle point - system is unstable for given
conditions.")
```

```
  } else if(is.complex(lambdas) == FALSE && lambdas[1] < 0 && lambdas[2] >
0){
    print("Unstable node - saddle point - system is unstable for given
conditions.")
  } else if(is.complex(lambdas) == TRUE && Re(lambdas[1]) > 0 &&
Re(lambdas[2]) > 0){
    print("Unstable focus - system is oscillatory and unstable.")
  } else if(is.complex(lambdas) == TRUE && Re(lambdas[1]) < 0 &&
Re(lambdas[2]) < 0){
    print("Stable focus - system is oscillatory and stable.")
  }

## [1] "Stable node - system is stable for given conditions."
```

Thus, to summarise, the system is **stable** at initial conditions **(3, 1)** and for the equilibrium points **(0.267, 3.732) and (3.732, 0.267)**. The system is **unstable** for the equilibrium point **(1.378, 1.378)** because that point is a saddle point.