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)
library(phaseR)
```

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")
```

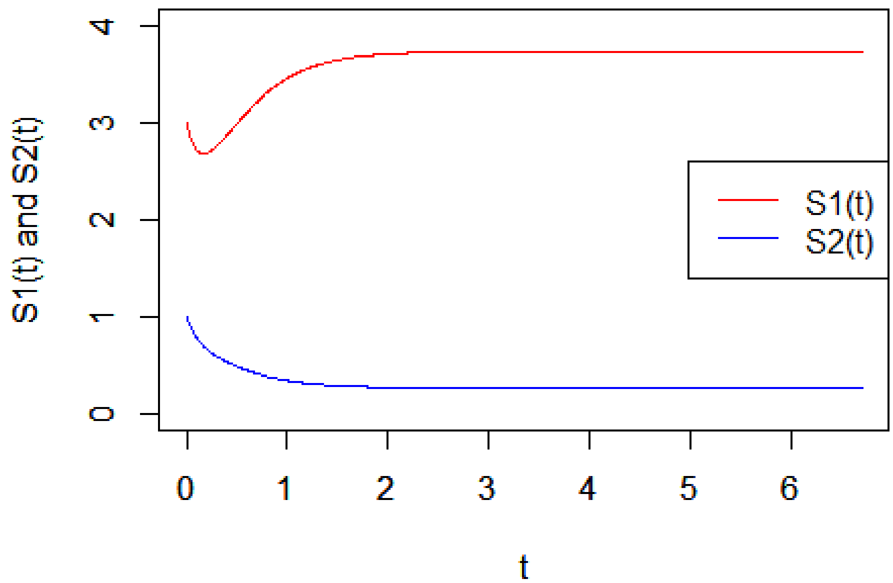
A matrix named *out* is created which contains the numerical solutions of both the differential equations. You can view it using the following command:

```
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
```

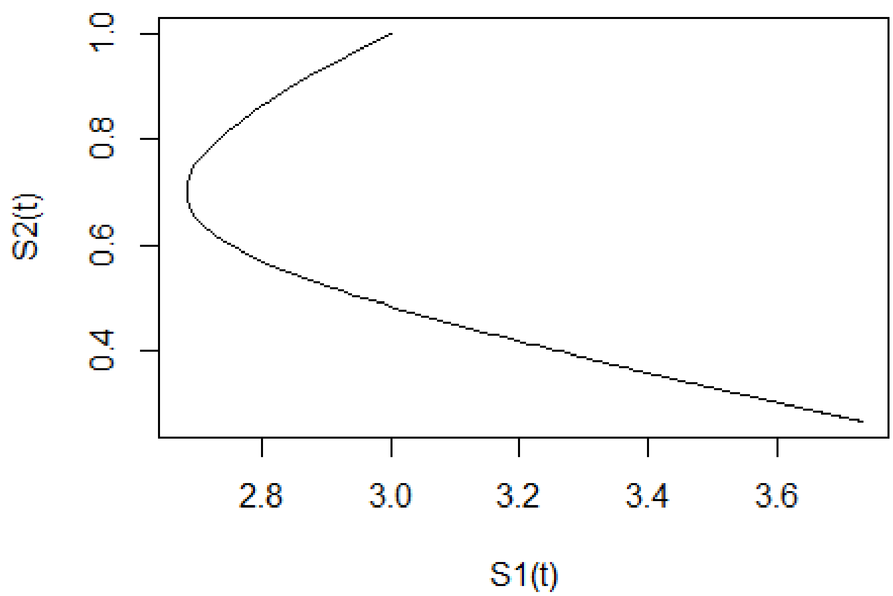
We can plot the solutions using the following code:

```
# 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 - Visual

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

Visually, we can plot a vector field, nullclines, and a trajectory in the phase plane to identify if the system is stable. This section has been adapted from S.O.S. Math and more information can be found there.

Vector Field

Consider a system of 2 differential equations.

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

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

In this system, the rates of change of the variables x and y are interdependent. This means any change in x will affect y and vice versa.

In our example of the Hill equation, the variable t is missing. Such a system is called an **autonomous system** and is written as:

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

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

Coming back to the general case, the solution to this system is a group of 2 functions, $x(t)$ and $y(t)$, which satisfy both the differential equations of the system. When we change the time t, we get a set of points on the X-Y plane for each t called a **trajectory**. The moving object has the coordinates $(x(t), y(t))$ at time t.

The velocity to the trajectory at time t is given by:

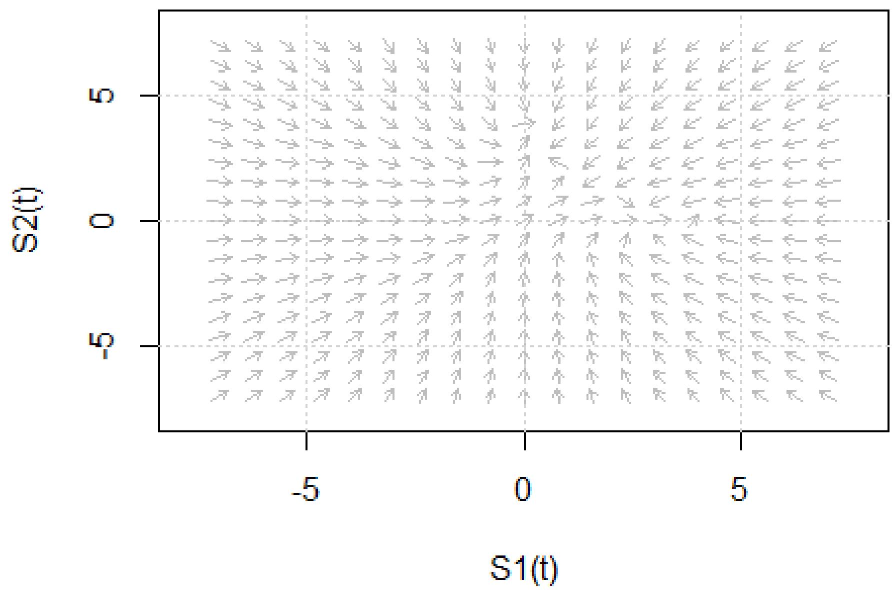
$$\overline{V} = (\frac{dx}{dt}, \frac{dy}{dt})$$

We do not need to know the solution of $x(t)$ and $y(t)$ because we already have:

$$\overline{V} = (f(t, x, y), g(t, x, y))$$

Thus, we can draw all the velocity vectors on the phase plane. This is called the **vector field**.

```r
## Plotting vector field
hill.flowField <- flowField(hill, xlim = c(-7, 7), ylim = c(-7, 7),
parameters = c(2, 2, 20, 20, 1, 1, 5, 5), points = 19, add = FALSE, xlab =
"S1(t)", ylab = "S2(t)")
grid()
```



The length of the vector depicts the speed of the moving object. **Each arrow (vector) represents one point of a trajectory.**

Nullclines

In the above vector field, we can see that the direction of motion is changing at certain regions. The boundaries of these regions are very important in determining the direction of the motion along the trajectories. These boundaries are called **nullclines**.

Consider the autonomous system:

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

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

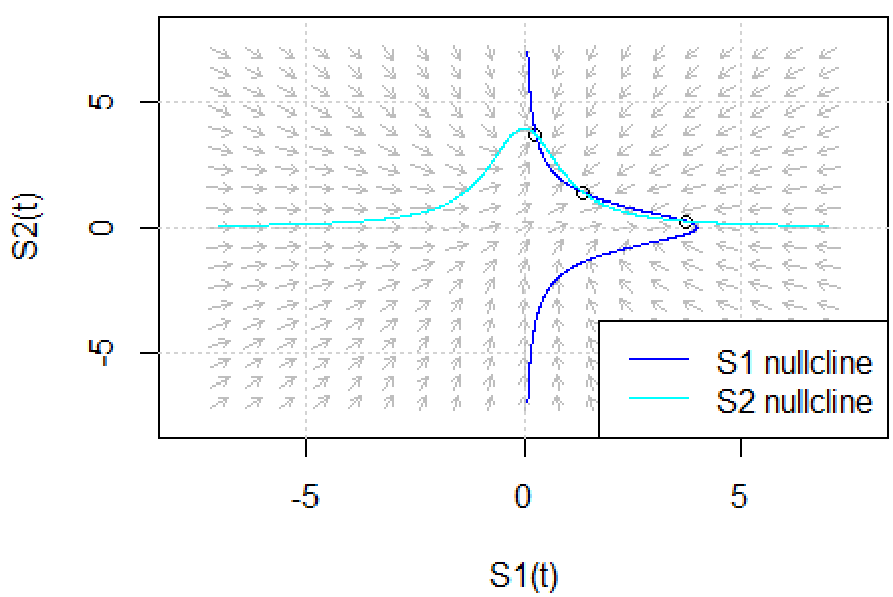
The **x-nullcline** is the set of points where $\frac{dx}{dt} = 0$ and the **y-nullcline** is the set of points where $\frac{dy}{dt} = 0$.

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

Along the x-nullcline, the velocity vectors are vertical while along the y-nullcline, the velocity vectors are horizontal. As long as we are travelling along a nullcline without crossing an equilibrium point, the direction of the velocity vector must be the same. Once we cross an equilibrium point, then we may have a change in the direction (from up to down, or right to left, and vice-versa).

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:

```
hill.flowField <- flowField(hill, xlim = c(-7, 7), ylim = c(-7, 7),
parameters = c(2, 2, 20, 20, 1, 1, 5, 5), points = 19, add = FALSE, xlab =
"S1(t)", ylab = "S2(t)")
grid()
# Refer the next section to find out the exact equilibrium points:
hill.traj1 <- trajectory(hill, y0 = c(1.378,1.378), tlim = c(-7,7),
parameters = c(2, 2, 20, 20, 1, 1, 5, 5), system = "two.dim")
hill.traj2 <- trajectory(hill, y0 = c(0.267,3.732), tlim = c(-7,7),
parameters = c(2, 2, 20, 20, 1, 1, 5, 5), system = "two.dim")
hill.traj3 <- trajectory(hill, y0 = c(3.732,0.267), tlim = c(-7,7),
parameters = c(2, 2, 20, 20, 1, 1, 5, 5), system = "two.dim")
# Plotting nullclines
hill.nullclines <- nullclines(hill, xlim = c(-7, 7), ylim = c(-7, 7),
parameters = c(2, 2, 20, 20, 1, 1, 5, 5), points = 500, add.legend = FALSE)
legend("bottomright", col = c("blue", "cyan"), legend = c("S1 nullcline", "S2
nullcline"), lty = 1)
```



By looking at the vector field, the nullclines, and the trajectories, we can say that the point (3.732,0.268) is a stable node, (0.268, 3.732) is a stable node, and (1.378,1.378) is an unstable saddle point. To know more about how to visually classify equilibrium points, refer to *page 2* of MCB Berkeley or go to Paul's Online Notes

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.

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

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

(See section on **nullclines**). We can use functions in the R package *rootSolve* to find out the points of equilibrium. We will verify this in a future section by plotting both of these functions and showing that they intersect at 3 different points, which are the solutions to this set of equations.

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

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
```

```
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.

Equilibrium Solutions of Autonomous Systems

An **equilibrium solution** (or **critical solution**) of an autonomous system is the trajectory of a moving-object with no motion, that is the object is not moving. In this case, the velocity vector is equal to 0. Consider the autonomous system:

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

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

The equilibrium solutions are the algebraic solutions of the system:

$$f(x, y) = 0$$

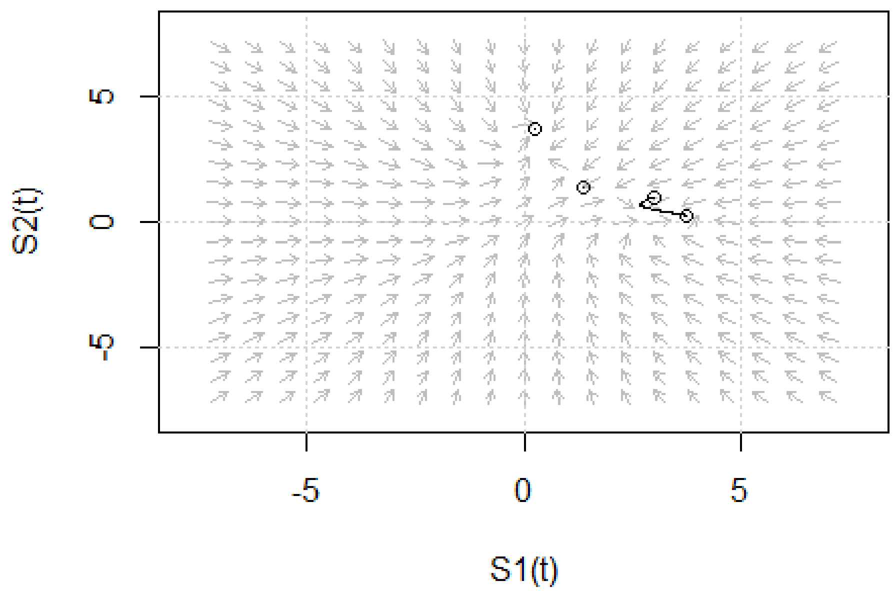
$$g(x, y) = 0$$

The equilibrium solutions are also called **equilibrium points** since the associated *trajectories* are **exactly points** on the phase plane.

Plotting the trajectories of the Hill equation system:

```
hill.flowField <- flowField(hill, xlim = c(-7, 7), ylim = c(-7, 7),
parameters = c(2, 2, 20, 20, 1, 1, 5, 5), points = 19, add = FALSE, xlab =
"S1(t)", ylab = "S2(t)")
grid()
```

```
## Plotting trajectories of equilibrium points
hill.traj1 <- trajectory(hill, y0 = c(1.378,1.378), tlim = c(-7,7),
parameters = c(2, 2, 20, 20, 1, 1, 5, 5), system = "two.dim")
hill.traj2 <- trajectory(hill, y0 = c(0.267,3.732), tlim = c(-7,7),
parameters = c(2, 2, 20, 20, 1, 1, 5, 5), system = "two.dim")
hill.traj3 <- trajectory(hill, y0 = c(3.732,0.267), tlim = c(-7,7),
parameters = c(2, 2, 20, 20, 1, 1, 5, 5), system = "two.dim")
## Plotting trajectory of initial conditions
hill.traj4 <- trajectory(hill, y0 = init, tlim = c(-7,7), parameters = c(2,
2, 20, 20, 1, 1, 5, 5), system = "two.dim")
```



Note that the trajectory of the initial conditions is exactly the phase plot we obtained in the previous section.

Stability Analysis - Eigen Values

Another way to verify the results of the visual analysis 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,

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

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

Under steady state,

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

$$\overline{A}\overline{X} + \overline{U} = 0$$

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

$\overline{X} = \overline{A}^{-1}\overline{U}$ 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:

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

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){
```

```r
    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."
```

```r
# 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
```

```r
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."
```

```r
# 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
```

```r
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, we have verified the results obtained from the visual analysis.