Advanced Modelling and Simulation PS 8

Reviewed	☐
Files & media	
Multi-select	AMAS
Status	Not started
Type	

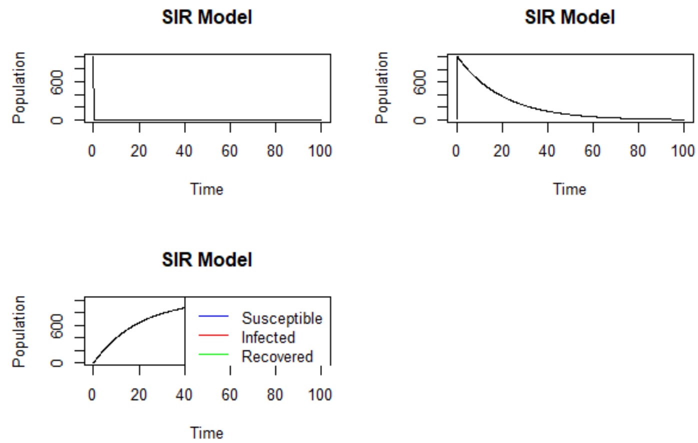
Group 2:

- Endri Lohja 29251
- Ajkana Hima 31033
- Orlando Rexhaj 31034
- Artem Khoripiakov 31024

Exercise 1: The SIR model (R and Python)

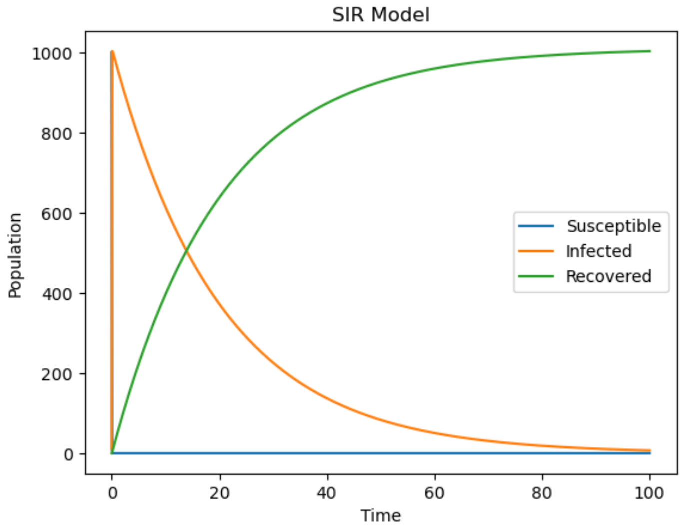
R

```
library(deSolve)

# Function defining the SIR model
sir_model <- function(time, state, parameters) {
  with(as.list(c(state, parameters)), {
    dS <- -alpha * S * I
    dI <- alpha * S * I - beta * I
    dR <- beta * I

    return(list(c(dS, dI, dR)))
  })
}

# Initial conditions
S0 <- 1000  # Initial number of susceptible individuals
I0 <- 10    # Initial number of infected individuals
R0 <- 0     # Initial number of recovered individuals

# Parameters
alpha <- 0.1  # Infection rate
beta <- 0.05  # Recovery rate

# Time points
times <- seq(0, 100, by = 0.1)

# Solve the system of equations
initial_state <- c(S = S0, I = I0, R = R0)
parameters <- c(alpha = alpha, beta = beta)
solution <- ode(y = initial_state, times = times, func = sir_model, parms = parameters)

# Plot the results
plot(solution, xlab = "Time", ylab = "Population", main = "SIR Model")
legend("topright", legend = c("Susceptible", "Infected", "Recovered"), col = c("blue", "red", "green"), lty = 1)
```



Python

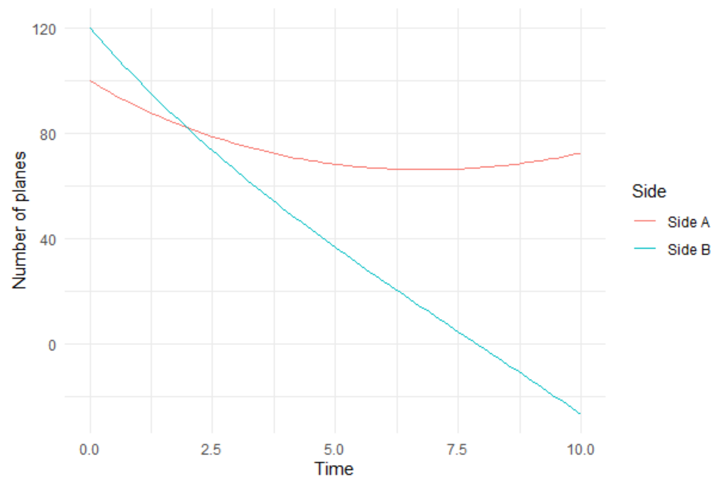
```
import numpy as np
import matplotlib.pyplot as plt
from scipy.integrate import odeint

# Function defining the SIR model
def sir_model(state, time, alpha, beta):
    S, I, R = state
    dS = -alpha * S * I
    dI = alpha * S * I - beta * I
    dR = beta * I
    return [dS, dI, dR]

# Initial conditions
S0 = 1000  # Initial number of susceptible individuals
I0 = 10    # Initial number of infected individuals
R0 = 0     # Initial number of recovered individuals

# Parameters
alpha = 0.1  # Infection rate
beta = 0.05  # Recovery rate

# Time points
times = np.linspace(0, 100, 1000)

# Solve the system of equations
initial_state = [S0, I0, R0]
args = (alpha, beta)
solution = odeint(sir_model, initial_state, times, args)

# Plot the results
plt.plot(times, solution[:, 0], label='Susceptible')
plt.plot(times, solution[:, 1], label='Infected')
plt.plot(times, solution[:, 2], label='Recovered')
plt.xlabel('Time')
plt.ylabel('Population')
plt.title('SIR Model')
plt.legend()
plt.show()
```



Exercise 2: The Lanchester combat (R and Python)

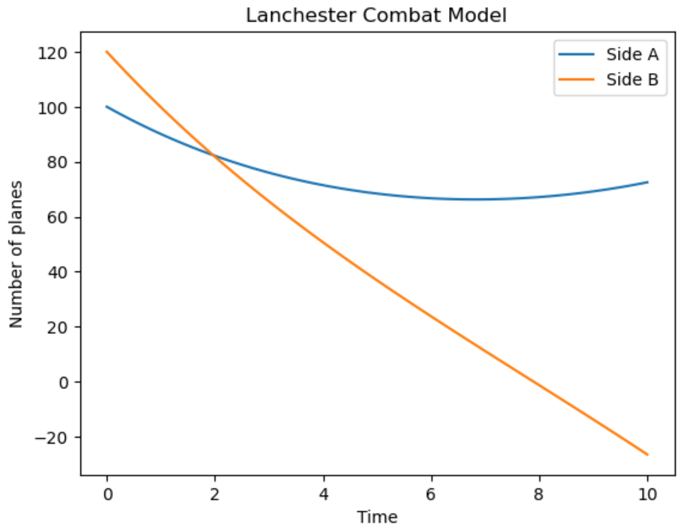
R

```
library(deSolve)
library(ggplot2)

# Function defining the Lanchester combat model
lanchester_model <- function(time, state, parameters) {
  with(as.list(c(state, parameters)), {
    dA <- r1 - k1 * B - b1 * A
    dB <- r2 - k2 * A - b2 * B

    return(list(c(dA, dB)))
  })
}

# Initial conditions
A0 <- 100  # Initial number of planes on side A
B0 <- 120  # Initial number of planes on side B

# Parameters
k1 <- 0.1  # Combat effectiveness rate for side A
k2 <- 0.2  # Combat effectiveness rate for side B
b1 <- 0.01  # Breakdown rate for side A
b2 <- 0.02  # Breakdown rate for side B
r1 <- 2  # Reinforcements rate for side A
r2 <- 1  # Reinforcements rate for side B

# Time points
times <- seq(0, 10, by = 0.1)

# Solve the system of equations
initial_state <- c(A = A0, B = B0)
parameters <- c(k1 = k1, k2 = k2, b1 = b1, b2 = b2, r1 = r1, r2 = r2)
solution <- ode(y = initial_state, times = times, func = lanchester_model, parms = parameters)

# Plot the results
df <- as.data.frame(solution)
ggplot(df, aes(x = time)) +
  geom_line(aes(y = A, color = "Side A")) +
  geom_line(aes(y = B, color = "Side B")) +
  labs(x = "Time", y = "Number of planes", color = "Side") +
  theme_minimal()
```



Python

```
import numpy as np
import matplotlib.pyplot as plt

# Function defining the Lanchester combat model
def lanchester_model(state, time, k1, k2, b1, b2, r1, r2):
    A, B = state
    dA = r1 - k1 * B - b1 * A
    dB = r2 - k2 * A - b2 * B
    return [dA, dB]

# Initial conditions
A0 = 100  # Initial number of planes on side A
B0 = 120  # Initial number of planes on side B

# Parameters
k1 = 0.1  # Combat effectiveness rate for side A
k2 = 0.2  # Combat effectiveness rate for side B
b1 = 0.01  # Breakdown rate for side A
b2 = 0.02  # Breakdown rate for side B
r1 = 2  # Reinforcements rate for side A
r2 = 1  # Reinforcements rate for side B

# Time points
times = np.linspace(0, 10, 100)

# Solve the system of equations
initial_state = [A0, B0]
args = (k1, k2, b1, b2, r1, r2)
solution = odeint(lanchester_model, initial_state, times, args)

# Plot the results
plt.plot(times, solution[:, 0], label='Side A')
plt.plot(times, solution[:, 1], label='Side B')
plt.xlabel('Time')
plt.ylabel('Number of planes')
plt.title('Lanchester Combat Model')
plt.legend()
plt.show()
```



Exercise 3: Monte Carlo Estimation of Pi in Python

R

```
estimate_pi <- function(num_points) {
  points <- matrix(runif(num_points * 2), ncol = 2)  # Generate random points in a 2D space

  # Count the number of points that lie within the unit circle
  inside_circle <- sum(sqrt(points[, 1]^2 + points[, 2]^2) < 1)

  # Calculate the proportion of points inside the circle
  proportion <- inside_circle / num_points

  # Estimate the value of Pi
  pi_estimate <- 4 * proportion

  return(pi_estimate)
}

# Number of random points
num_points <- 1000000

# Estimate Pi using the Monte Carlo method
pi_estimate <- estimate_pi(num_points)

print(paste("Estimated value of Pi:", pi_estimate))
```

Python

```
import numpy as np
def estimate_pi(num_points):
    points = np.random.rand(num_points, 2)  # Generate random points in a 2

    # Count the number of points that lie within the unit circle
    inside_circle = np.sum(np.linalg.norm(points, axis=1) < 1)

    # Calculate the proportion of points inside the circle
    proportion = inside_circle / num_points

    # Estimate the value of Pi
    pi_estimate = 4 * proportion

    return pi_estimate

# Number of random points
num_points = 1000000

# Estimate Pi using the Monte Carlo method
pi_estimate = estimate_pi(num_points)

print("Estimated value of Pi:", pi_estimate)
```

Exercise 4: Simulating a Lottery Draw

R

```
lottery_draw <- function() {
  numbers <- 1:49  # Create a vector of numbers from 1 to 49
  draw <- sample(numbers, size = 6, replace = FALSE)  # Select 6 unique numbers

  return(sort(draw))
}

# Simulate a lottery draw
lottery_numbers <- lottery_draw()

# Print the lottery numbers
cat("Lottery numbers:", lottery_numbers, "\n")
```

Python

```
import numpy as np

def lottery_draw():
    numbers = np.arange(1, 50)  # Create an array of numbers from 1 to 49
    draw = np.random.choice(numbers, size=6, replace=False)  # Select 6 uni

    return sorted(draw)

# Simulate a lottery draw
lottery_numbers = lottery_draw()

# Print the lottery numbers
print("Lottery numbers:", lottery_numbers)
```

Exercise 5: Distribution for a random variable

R

```r
# Simulate 10^5 numbers from a standard normal distribution
numbers <- rnorm(10^5)

# Calculate the number of points lying more than 5 standard deviations away from 0
count <- sum(abs(numbers) > 5)

# Calculate the proportion
proportion <- count / length(numbers) * 100

print(paste("Proportion of points more than 5 standard deviations away:", proportion, "%"))
```

Python

```python
import numpy as np

# Simulate 10^5 numbers from a standard normal distribution
numbers = np.random.normal(0, 1, size=10**5)

# Calculate the number of points lying more than 5 standard deviations away
count = np.sum(np.abs(numbers) > 5)

# Calculate the proportion
proportion = count / len(numbers) * 100

print("Proportion of points more than 5 standard deviations away:", proport
```