Hồi Quy Phi Tuyến (HD Code)

Name**Sử dụng ngôn ngữ R và Python (2 trường hợp dùng hàm thư viện và cài đặt thuật toán).**

Ví dụ minh họa cho bài toán hồi quy phi tuyến Y = C1X/(C2 + X)

$X = [0.038, 0.194, 0.425, 0.626, 1.253, 2.500, 3.740]$

$Y = [0.050, 0.127, 0.094, 0.2122, 0.2729, 0.2665, 0.3317]$

$$Y = \frac{C_1 X}{C_2 + X}$$

$$r = Y - \frac{C_1 X}{C_2 + X}$$

$$\frac{\partial r}{\partial C_1} = -\left(\frac{X}{C_2 + X}\right)$$

$$\frac{\partial r}{\partial C_2} = \left(\frac{C_1 X}{(C_2 + X)^2}\right)$$

- Cài đặt thuật toán Gauss-Newton trên ngôn ngữ Python

```
import numpy as np

def gauss_newton(X, y, B0, tol=1e-6, max_iter=1000):
    """
    Gauss-Newton optimization algorithm for nonlinear regression.

    Parameters:
    - X: array-like, shape (n_samples, n_features)
        Feature matrix.
    - y: array-like, shape (n_samples,)
        Target vector.
    - B0: array-like, shape (n_parameters,)
        Initial parameter values.
    - tol: float, optional (default=1e-6)
        Tolerance for stopping criteria.
    - max_iter: int, optional (default=1000)
        Maximum number of iterations.

    Returns:
    - B_opt: array-like, shape (n_parameters,)
        Optimal parameter values.
    - n_iter: int
        Number of iterations performed.
    """
```

```
    # Initialize Jacobian matrix
    J = np.zeros((len(X), len(B0)))

    # Initialize iteration counter
    n_iter = 0

    while True:
        n_iter += 1

        # Compute partial derivatives
        rC1 = -(X / (B0[1] + X))
        rC2 = (B0[0] * X / (B0[1] + X) ** 2)

        # Update Jacobian matrix
        J[:, 0] = rC1
        J[:, 1] = rC2

        # Compute SSE
        SSE = y - ((B0[0] * X) / (B0[1] + X))

        # Compute inverse of (J^T * J)
        t1 = np.linalg.inv(np.dot(J.T, J))

        # Compute t1 * J^T
        t2 = np.dot(t1, J.T)

        # Compute t2 * SSE
        t3 = np.dot(t2, SSE)

        # Update parameters
        B1 = B0 - t3

        # Check for convergence
        if np.max(np.abs(B1 - B0)) <= tol:
            break

        # Update parameters for next iteration
        B0 = B1

        # Check for maximum iterations
        if n_iter >= max_iter:
            break

    return B0, n_iter

# Example usage:
```

```
B_opt, n_iter = gauss_newton(X, y, B0)
C1, C2 = B_opt
print(f'Constant C1 = {C1:.4f}, Constant C2 = {C2:.4f}, Number of 
iterations = {n_iter}')
```

- Khai báo biến độc lập và biến phụ thuộc

```
#2. Khai báo biến phụ thuộc và biến độ lập

#2.1 Biến độc lập
X = np.array([0.038, 0.194, 0.425, 0.626, 1.253, 2.500, 3.740])

#2.2 Biến phụ thuộc
y = np.array([0.050, 0.127, 0.094, 0.2122, 0.2729, 0.2665, 0.33171])
```

- Khởi tạo giá trị

```
#3. Khởi tạo giá trị ban đầu
B0 = (1,1)

#4. Khởi tạo ma trận Jacobian
J = np.zeros((len(X),len(B0)))

#5. Khởi tạo giá trị Iter
i = 0
```

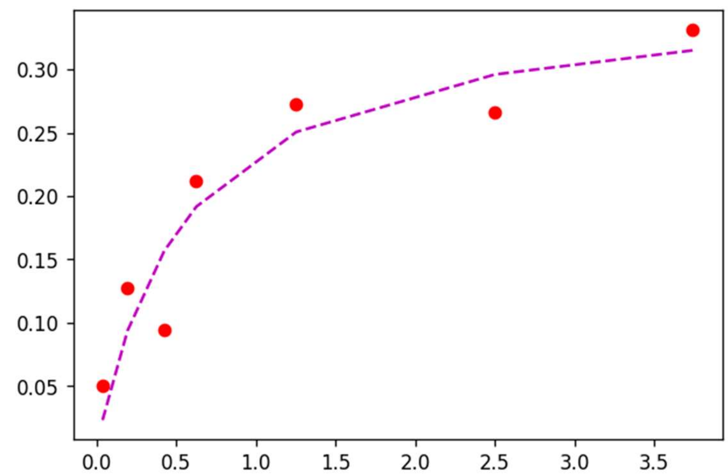
- Đưa ra độ chính xác và hình

```python
#6.1 Plotting

pred = C1*X/(C2+X)
plt.figure(1, figsize=(6,4),dpi=120)
plt.scatter(x=X, y = y, c = 'red', marker = 'o', label ='Actual')
plt.plot(X, pred, "--m", label='Predict')

#6.2 Đánh giá độ đo mô hình
SSR = np.sum((y-pred)**2)
SSE = np.sum((y-np.mean(y)**2))
r2 = 1 - (SSR/SSE)
print(r2)
```

```
0.9928184309771957
```



- Trường hợp sử dụng thư viện Curve_Fit để tối ưu hóa phương trình hồi quy tuyến tính

```python
import numpy as np
from scipy.optimize import curve_fit

# Define the nonlinear model function
def model_func(x, C1, C2):
    return C1 * x / (C2 + x)

# Define the independent and dependent variables
X = np.array([0.038, 0.194, 0.425, 0.626, 1.253, 2.500, 3.740])
y = np.array([0.050, 0.127, 0.094, 0.2122, 0.2729, 0.2665, 0.33171])

# Initial guess for the parameters
initial_guess = (1, 1)

# Fit the model to the data
params, covariance = curve_fit(model_func, X, y, p0=initial_guess)

# Extract the optimal parameters
C1_opt, C2_opt = params

print(f'Constant C1 = {C1_opt:.4f}, Constant C2 = {C2_opt:.4f}')
```

```
Constant C1 = 0.3618, Constant C2 = 0.5563
```

- Cài đặt thuật toán Gauss-Newton trên ngôn ngữ R

```r
# Import thư viện cần thiết
library(Matrix)

# Khai báo biến phụ thuộc và biến độ lập
# Biến độc lập
X <- c(0.038, 0.194, 0.425, 0.626, 1.253, 2.500, 3.740)
# Biến phụ thuộc
y <- c(0.050, 0.127, 0.094, 0.2122, 0.2729, 0.2665, 0.33171)

# Khởi tạo giá trị ban đầu
B0 <- c(1, 1)

# Khởi tạo ma trận Jacobian
J <- matrix(0, nrow = length(X), ncol = length(B0))

# Khởi tạo giá trị Iter
i <- 0

while (TRUE) {
  i <- i + 1
  # Đạo hàm riêng theo C1
  rC1 <- -(X / (B0[2] + X))
  # Đạo hàm riêng theo C2
  rC2 <- (B0[1] * X / (B0[2] + X)^2)
  J[, 1] <- rC1
  J[, 2] <- rC2
  # Tính SSE
  SSE <- y - ((B0[1] * X) / (B0[2] + X))

  # Tính Inverse (J^T*J)^-1
  t1 <- solve(t(J) %*% J)

  # Tính t1.J^T
  t2 <- t1 %*% t(J)

  # Tính t2.SSE
  t3 <- t2 %*% SSE

  # Update Bnew
  B1 <- B0 - t3

  # Kiểm tra điều kiện dừng
  t4 <- abs(B1 - B0)
  if (max(t4) <= 1e-6) {
    break
  }
```

```
  B0 <- B1
}

C1 <- round(B0[1], 4)
C2 <- round(B0[2], 4)

print(paste("Constant C1 =", C1, ", Constant C2 =", C2, ", i =", i))
```

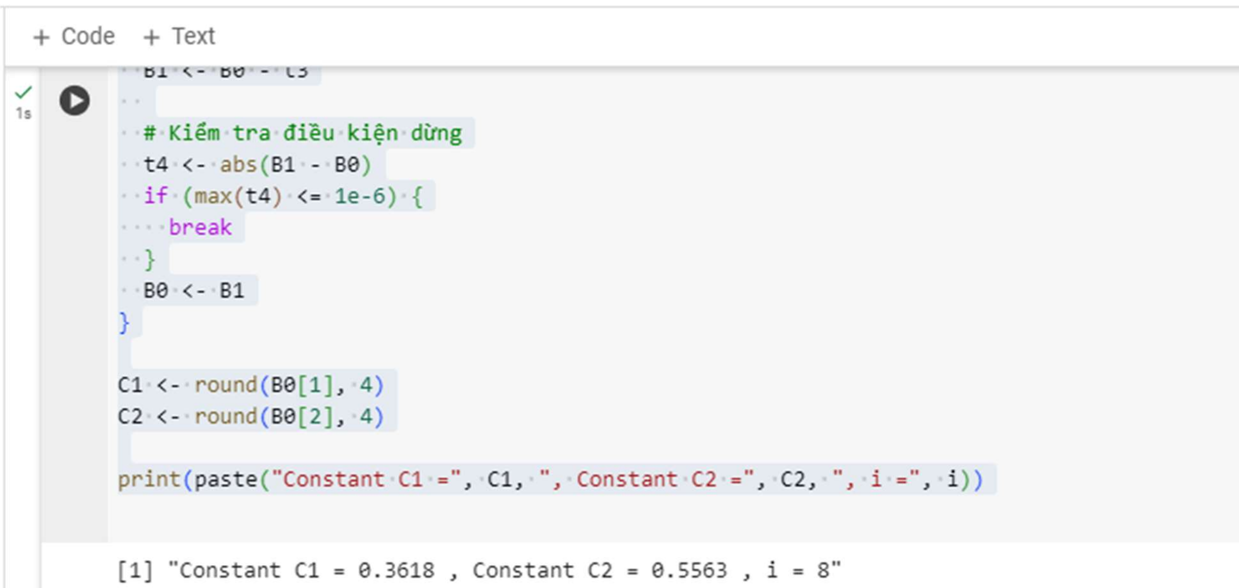


Figure 1 Kết quả tối ưu Gauss-Newton trên ngôn ngữ R

- Sử dụng thư viện có sẵn trong ngôn ngữ R, minpack.lm

```r
# Install and load the minpack.lm package
install.packages("minpack.lm")
library(minpack.lm)

# Define the model function
model <- function(X, C1, C2) {
  C1 * X / (C2 + X)
}

# Initial guess for parameters
B0 <- c(C1 = 1, C2 = 1)

# Fit the model using nlsLM
fit <- nlsLM(y ~ model(X, C1, C2), start = B0, control = nls.lm.control(maxiter = 100))

# Get the coefficients
coefficients <- coef(fit)
C1 <- round(coefficients["C1"], 4)
C2 <- round(coefficients["C2"], 4)

print(paste("Constant C1 =", C1, ", Constant C2 =", C2))
```

```
Installing package into '/usr/local/lib/R/site-library'
(as 'lib' is unspecified)

[1] "Constant C1 = 0.3618 , Constant C2 = 0.5563"
```

Nhận xét: Kết quả 4 cách làm là giống nhau, với dữ liệu đề xuất

Yêu cầu 2.2 Ứng dụng thuật toán Gauss-Newton giải bài toán hồi qui phi tuyến nhiều biến với dữ liệu về Việt Nam

Nguồn dữ liệu lấy từ:

https://databank.worldbank.org/reports.aspx?source=2&country=VNM&series&period&fbclid=IwAR05I8KKE1RIsADdIw3jRkPwYInibWou-4rrnC__dfPMT-eTeA_f5jM3RB0#

Bộ dữ liệu trích xuất hai cột **CO2 emissions (metric tons per capita)** và **Agriculture, forestry, and fishing, value added (% of GDP)** từ 1998 đến 2021

	A	B	C	D	E	F
1	CO2	Agriculture				
2	0.589587	25.78078				
3	0.594094	25.43444				
4	0.648189	24.53458				
5	0.710649	23.24105				
6	0.824577	23.02944				
7	0.870242	22.54244				
8	1.033295	21.8077				
9	1.110973	19.29998				
10	1.13005	18.72679				
11	1.24038	18.6551				
12	1.373702	20.41314				
13	1.529666	19.16846				
14	1.732202	15.37509				
15	1.76542	16.25915				
16	1.741551	16.19951				
17	1.820112	15.21561				
18	1.980575	14.88035				
19	2.185815	14.47473				
20	2.38416	13.81826				
21	2.444645	12.92988				
22	3.014711	12.30667				
23	3.567848	11.78453				
24	3.67644	12.6554				

Dùng thư viện curve_fit của python để tìm ra hàm tối ưu

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

# Dữ liệu
CO2 = np.array([0.589587, 0.594094, 0.648189, 0.710649, 0.824577,
0.870242, 1.033295, 1.110973, 1.130050, 1.240380,
       1.373702, 1.529666, 1.732202, 1.765420, 1.741551, 1.820112,
1.980575, 2.185815, 2.384160, 2.444645,
       3.014711, 3.567848, 3.676440])

Agriculture = np.array([25.780780, 25.434438, 24.534582, 23.241048,
23.029442, 22.542437, 21.807699, 19.299979, 18.726785,
               18.655100, 20.413144, 19.168460, 15.375091, 16.259148,
16.199508, 15.215612, 14.880349, 14.474726,
               13.818255, 12.929879, 12.306668, 11.784529, 12.655404])

# Hàm mũ
def exponential_function(x, a, b):
    return a * np.exp(b * x)

# Fit dữ liệu
from scipy.optimize import curve_fit
popt, pcov = curve_fit(exponential_function, CO2, Agriculture)

# Vẽ biểu đồ
```

```python
plt.scatter(CO2, Agriculture, label='Dữ liệu thực tế')
plt.plot(CO2, exponential_function(CO2, *popt), color='red', label='Hàm mũ tương quan')
plt.xlabel('CO2')
plt.ylabel('Agriculture')
plt.title('Mối quan hệ phi tuyến giữa CO2 và Agriculture')
plt.legend()
plt.show()
```

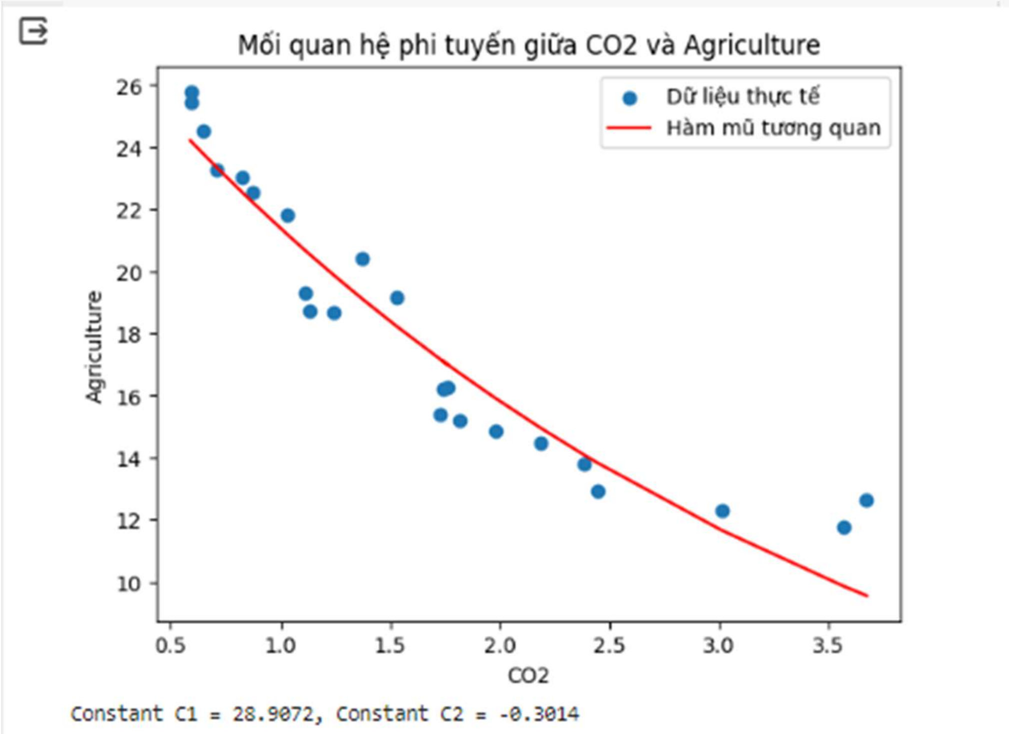


Figure 2 Kết quả của mô hình dự báo

Phương trình hồi quy phi tuyến đơn biến: Argiculture = $28.9072. e^{-0.314*CO2}$

Thực hiện hồi quy phi tuyến đa biến dữ liệu Việt Nam

A	B	C	D
CO2	Agricultur	PopulationGrowht	
0.589587	25.78078	1.396774	
0.594094	25.43444	1.282176	
0.648189	24.53458	1.116867	
0.710649	23.24105	1.028394	
0.824577	23.02944	1.027718	
0.870242	22.54244	1.028293	
1.033295	21.8077	1.020116	
1.110973	19.29998	1.004362	
1.13005	18.72679	0.969169	
1.24038	18.6551	0.960768	
1.373702	20.41314	0.980255	
1.529666	19.16846	1.029392	
1.732202	15.37509	1.06743	
1.76542	16.25915	1.067493	
1.741551	16.19951	1.072013	
1.820112	15.21561	1.07638	
1.980575	14.88035	1.066399	
2.185815	14.47473	1.042271	
2.38416	13.81826	1.009227	
2.444645	12.92988	0.96872	
3.014711	12.30667	0.93284	
3.567848	11.78453	0.904491	
3.67644	12.6554	0.906299	

Bộ dữ liệu trích xuất hai cột **CO2 emissions (metric tons per capita)** và **Agriculture, forestry, and fishing, value added (% of GDP)** từ 1998 đến 2021, thêm cột tỉ lệ gia tăng dân số **Population growth (annual %)**

Dùng thư viện curve_fit của python để tìm ra hàm tối ưu trên đa biến

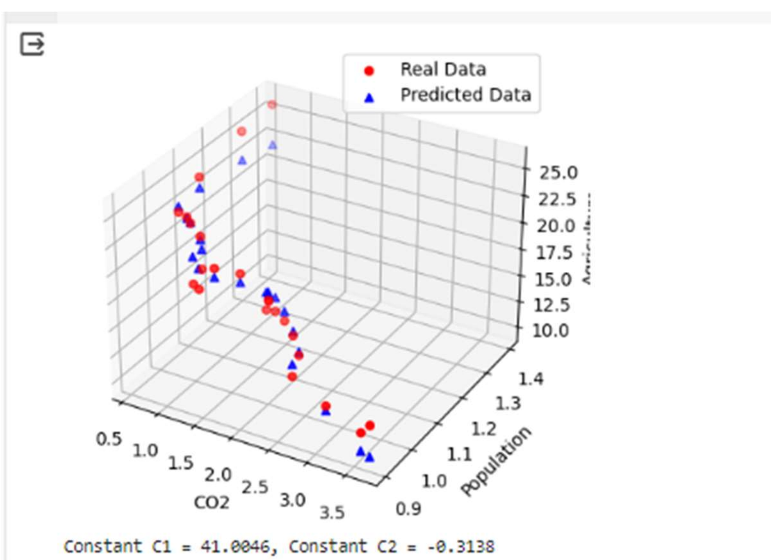
```
import numpy as np
from scipy.optimize import curve_fit
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
# Dữ liệu
CO2 = np.array([0.589587, 0.594094, 0.648189, 0.710649, 0.824577,
0.870242, 1.033295, 1.110973, 1.130050, 1.240380,
                1.373702, 1.529666, 1.732202, 1.765420, 1.741551,
1.820112, 1.980575, 2.185815, 2.384160, 2.444645,
                3.014711, 3.567848, 3.676440])

Agriculture = np.array([25.780780, 25.434438, 24.534582, 23.241048,
23.029442, 22.542437, 21.807699, 19.299979, 18.726785,
                        18.655100, 20.413144, 19.168460, 15.375091,
16.259148, 16.199508, 15.215612, 14.880349, 14.474726,
                        13.818255, 12.929879, 12.306668, 11.784529,
12.655404])
```

```
Population = np.array(df["PopulationGrowht"])  # Lưu ý sửa lại tên cột
# Định nghĩa một hàm mục tiêu (hàm mô hình)
def model_func(x, c1, c2):
    CO2, Population = x
    return c1 * np.exp((CO2 + Population) * c2)
# Phù hợp dữ liệu với mô hình
popt, pcov = curve_fit(model_func, (CO2, Population), Agriculture)
# Tính toán giá trị dự đoán từ mô hình đã phù hợp
predicted_Agriculture = model_func((CO2, Population), *popt)
# Vẽ đồ thị 3D
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Vẽ dữ liệu thực tế
ax.scatter(CO2, Population, Agriculture, c='r', marker='o', label='Real
Data')
# Vẽ dữ liệu dự đoán từ mô hình
ax.scatter(CO2, Population, predicted_Agriculture, c='b', marker='^',
label='Predicted Data')
ax.set_xlabel('CO2')
ax.set_ylabel('Population')
ax.set_zlabel('Agriculture')
plt.legend()
plt.show()
```

Kết quả



Phương trình hồi quy phi tuyến đa biến: Argiculture = $41{,}0046. e^{-0.314*(CO2+Population)}$

Tài liệu tham khảo

[1] https://en.citizendium.org/wiki/Newton%27s_method#Computational_complexity

[2] https://en.wikipedia.org/wiki/Gauss%E2%80%93Newton_algorithm

[3] Tham khảo code tại: https://www.youtube.com/watch?v=weuKzGFVQV0