

Automated measurement of water infiltration: implications for sustainable water management in different soil types

Medición automatizada de la infiltración de agua: implicaciones para la
gestión sostenible del agua en diferentes tipos de suelo

Luis Fernando Fantti ¹
Willian José Ferreira ²
Marcelo dos Santos Targa ³
Julio Cesar Raposo de Almeida ⁴
Rafael Barbosa Rodrigues ⁵
Marcos Cleve da Silva ⁶

¹ Forest Engineer, Master in Environmental Science.
University of Taubaté, Brazil. lferfantti@yahoo.com.br

² Physics, Ph.D. in Space Geophysics.
University of Taubaté, Brazil. willian.jferreira@unitau.br

³ Agronomist, Ph.D. in Agronomy.
University of Taubaté, Brazil. mtarga@unitau.br

⁴ Agronomist, Ph.D. in Forest Resources.
University of Taubaté, Brazil. julio.cralmeida@unitau.br

⁵ Systems Analyst, Master's student in Environmental Sciences.
University of Taubaté, Brazil. rafael.brodrigues@unitau.br

⁶ Civil Engineer, Master in Environmental Sciences
University of Taubaté, Brazil. marcoscleve@msn.com

HOW TO CITE: Fantti, L., Ferreira, W., Targa, M., de Almeida, J., Rodrigues, R. & da Silva, M. (2025). Automated measurement of water infiltration: Implications for sustainable water management in different soil types. Annexes. *Revista de Investigación Agraria y Ambiental*, 16(2).

ANNEXES**APPENDIX 1 - Interacting with Arduino using Python via Serial Communication**

#To store the data received from the Arduino in a file on your notebook, you can modify the Python code to write the received data to a text file.

pip install pyserial

#Python Code

```
import serial
import time
```

Establishing connection with Arduino through serial port

```
arduino = serial.Serial('COM3', 9600, timeout=.1) # Replace 'COM3' with your Arduino's
port
```

```
time.sleep(2) # Allowing some time for the connection to be established
```

```
while True:
```

```
    # Sending a command to Arduino to request data
```

```
    arduino.write(b'r') # 'r' can be any character indicating a request in the Arduino code
```

```
    time.sleep(1) # Waiting for the Arduino to process and send data back
```

```
    # Reading data sent from Arduino
```

```
    data = arduino.readline().decode().strip() # Decoding the received bytes
```

```
    if data: # If data is received
```

```
        print(f"Data received from Arduino: {data}")
```

```
arduino.close() # Closing the serial connection when done
```

#Arduino Code

```
void setup() {
```

```
    Serial.begin(9600); // Starting serial communication at 9600 baud rate
```

```
}
```

```
void loop() {
```

```
    if (Serial.available() > 0) {
```

```
        char command = Serial.read(); // Reading incoming data from Python
```

```
        if (command == 'r') { // If command received is 'r'
```

```
            int sensorValue = analogRead(A0); // Reading analog pin A0 (example)
```

```

    Serial.println(sensorValue); // Sending the sensor value back to Python
    delay(1000); // Delay for stability
  }
}
}

#Option

with open('arduino_data.txt', 'w') as file: # Opening a file to store data
    while True:
        arduino.write(b'r') # Sending a command to Arduino to request data
        time.sleep(1) # Waiting for the Arduino to process and send data back

        data = arduino.readline().decode().strip() # Reading data sent from Arduino
        if data: # If data is received
            print(f"Data received from Arduino: {data}")
            file.write(data + '\n') # Writing data to the file

arduino.close() # Closing the serial connection when done

```

APPENDIX 2 - Curve fitting with Horton Equation for hydrological analysis: parameter estimation and R-squared calculation

```

import numpy as np
from scipy.optimize import curve_fit

data = df_select

# Horton Equation
def horton_eq(t, fc, f0, k):
    return fc + (f0 - fc) * np.exp(-k * t)

#f(t) is the infiltration rate over time; fc is the constant infiltration rate (final value for large t); f0 is the initial infiltration rate; k is the decay rate.

# Function to fit Horton parameters and calculate R^2 with initial values and more evaluations
def horton_fit(data):
    data = data.dropna() # Remove rows with NaN values
    p0 = (1, 1, 1) # Initial parameter values
    try:
        popt, _ = curve_fit( Horton_eq, data['hora_acumulada'], data['inf_acumulada'],
        p0=p0, maxfev=10000)

```

```

        fitted_curve = horton_eq(data['hora_acumulada'], *popt)
        residuals = data['inf_acumulada'] - fitted_curve
        ss_res = np.sum(residuals ** 2)
        ss_tot = np.sum((data['inf_acumulada'] - np.mean(data['inf_acumulada'])) ** 2)
        r_squared = 1 - (ss_res / ss_tot)
        return popt, r_squared
    except RuntimeError as e:
        return np.nan, np.nan # In case of failure, return NaN for parameters and R^2

# Group by 'Point' and apply the fitting function
results = df_select.groupby('Point').apply( Horton_fit)

# Create lists for the results
points = []
fc_values = []
f0_values = []
k_values = []
r_squared_values = []

# Iterate through the results and extract the values
for point, (params, r_squared) in results.items():
    points.append(point)
    fc_values.append(params[0])
    f0_values.append(params[1])
    k_values.append(params[2])
    r_squared_values.append(r_squared)

# Create a new DataFrame with the results
results_df = pd.DataFrame({
    'Point': points,
    'Parameter_fc': fc_values,
    'Parameter_f0': f0_values,
    'Parameter_k': k_values,
    'R_squared_coefficient': r_squared_values
})

```

Conflicto de intereses

*Los autores declaran no tener
ningún conflicto de intereses.*

**Licencia de Creative Commons**

Revista de Investigación Agraria y Ambiental is licensed under a Creative Commons
Reconocimiento - No Comercial - Compartir Igual 4.0 Internacional License.