# Sprawozdanie: Analiza liczby błędów krytycznych w systemie operacyjnym

**1. Wstęp**

Celem niniejszego sprawozdania jest analiza liczby błędów krytycznych wykrytych w trakcie eksploatacji systemu operacyjnego. Badanie obejmuje modelowanie danych za pomocą modelu liniowego oraz funkcji logistycznej, co pozwala zidentyfikować tendencje oraz przewidzieć dalszy rozwój liczby błędów w czasie.

Treść zadania:  
Obraz zawierający tekst, linia, zrzut ekranu, Równolegle

Opis wygenerowany automatycznie

**2 . Metodologia**

**2.1. Model liniowy**

Regresja liniowa opisuje zależność między zmienną zależną a zmienną niezależną za pomocą równania:

Gdzie:

* : przewidywana wartość skumulowanej liczby błędów,
* : numer miesiąca,
* : wyraz wolny,
* : współczynnik nachylenia (wpływ numeru miesiąca na liczbę błędów),
* : błąd modelu.

**2.2. Funkcja logistyczna**

Funkcja logistyczna modeluje proces, w którym liczba błędów rośnie w sposób nasycony:

Gdzie:

* : maksymalna liczba błędów (poziom nasycenia),
* : parametr przesunięcia w poziomie,
* : tempo wzrostu.

**3. Wyniki analizy**

Analizę problemu zaczynamy od przyjrzenia się danym.

Poniżej znajduje się wykres zależności liczby błędów w okresie użytkowania programu.

**Tendencja**:

* Nie widać jednoznacznej tendencji wzrostowej lub spadkowej – skoki są nieregularne, co może wskazywać na różnorodne czynniki zewnętrzne lub wewnętrzne wpływające na stabilność systemu.

Obraz zawierający tekst, zrzut ekranu, Czcionka, numer

Opis wygenerowany automatycznie

Korelacja tych dwóch parametrów to: 

Wnioski: liczba błędów od miesiąca jest słabo skorelowana.

Aby lepiej sprecyzować zjawisko zamiast liczby błędów w danych miesiącu używamy sumy wystąpień błędów do danego momentu(miesiąca)

**3.1. Model liniowy**

Zmienna zależna: Wystapienia\_lacznie (skumulowana liczba błędów).

Zmienna niezależna: Nr\_miesiaca (numer miesiąca od początku eksploatacji).

Obraz zawierający zrzut ekranu, tekst, linia, Wykres

Opis wygenerowany automatycznie

**Wyniki regresji liniowej:**

Obraz zawierający tekst, zrzut ekranu, oprogramowanie, numer

Opis wygenerowany automatycznie

Analiza parametrów:

# OLS Regression Results

|  |  |
| --- | --- |
| Dep. Variable: | *Wystąpienia łącznie* |
| R-squared: | 0.938 |
| Model: | OLS (Metoda Najmniejszych Kwadratów) |
| Adj. R-squared: | 0.937 |
| F-statistic: | 1370 |
| Prob(F-statistic): | 1*.*19 · 10−56 |
| Log-Likelihood: | -421.63 |
| AIC: | 847.3 |
| BIC: | 852.3 |
| No. Observations: | 93 |
| Df Residuals: | 91 |
| Df Model: | 1 |

Obraz zawierający tekst, zrzut ekranu, Czcionka, dokument

Opis wygenerowany automatycznie

# Testy reszt

1. Test Omnibus: *Prob(Omnibus)* = 0.010
2. Test Jarque-Bera: *Prob(JB)* = 0.114
3. Sko´sno´s´c (*Skew*): 0.287
4. Kurtoza (*Kurtosis*): 2.111

Wnioski: Model liniowy bardzo dobrze wyjaśnia zmienność danych, o czym świadczy wysoki współczynnik . Parametry modelu są istotne statystycznie, a interpretacja współczynnika nachylenia (Nr\_miesiaca) wskazuje na systematyczny wzrost liczby błędów w kolejnych miesiącach.

**3.2. Funkcja logistyczna**

Przechodząc do kolejnego etapu analizy, wykorzystamy funkcję logistyczną do zbadania zależności między zmiennymi. Funkcja logistyczna jest narzędziem statystycznym, które umożliwia modelowanie zmiennych zależnych w postaci proporcji (w naszym przypadku: sumaryczna liczba błędów do danego miesiąca). Zamiast przewidywać wartości bezpośrednio, model ten estymuje krzywą, która opisuje tempo wzrostu i nasycenia liczby błędów w czasie. Dzięki temu możemy oszacować, jak liczba błędów narasta w poszczególnych miesiącach oraz przewidzieć maksymalną liczbę błędów w całym okresie analizy.

Obraz zawierający tekst, zrzut ekranu, Czcionka

Opis wygenerowany automatycznie

**Interpretacja:**

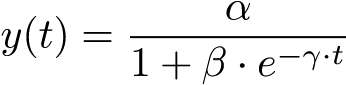
1. **Koszt początkowy (Iteration 0)**: Koszt na starcie wynosił 1.2687×. Jest to suma kwadratów różnic między rzeczywistymi danymi a początkową estymacją.
2. **Zmniejszanie kosztu**:
   * Koszt szybko zmniejsza się w początkowych iteracjach, co wskazuje na znaczną poprawę dopasowania modelu.
   * Po 6. iteracji tempo redukcji kosztu zaczyna drastycznie spadać.
3. **Norm kroku (Step Norm)**:
   * Oznacza wielkość kroku w przestrzeni parametrów. Początkowo kroki są duże (np. 48.3 w iteracji 4), a później coraz mniejsze, co wskazuje na zbliżanie się do optymalnych wartości.
4. **Optymalność (First-order optimality)**:
   * Maleje stopniowo. W ostatnich iteracjach osiąga bardzo niską wartość (0.197), co sugeruje spełnienie warunku zatrzymania.
5. **Kryterium zakończenia**:
   * Kod zatrzymuje się, ponieważ został spełniony warunek tolerancji na różnicę kosztów (fto).

Obraz zawierający tekst, zrzut ekranu, linia, Wykres

Opis wygenerowany automatycznie

# Funkcja logistyczna

Funkcja logistyczna opisana jest wzorem:



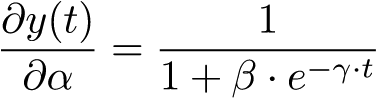
gdzie:

* *y*(*t*) – wartość funkcji w czasie *t*,
* *α* – maksymalna wartość funkcji (poziom nasycenia),
* *β* – parametr przesunięcia w poziomie,
* *γ* – parametr kontrolujący tempo wzrostu.

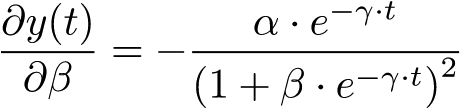
# Pochodne cząstkowe funkcji logistycznej

Macierz Jacobiego zawiera pochodne cząstkowe funkcji logistycznej względem parametrów *α*, *β*, *γ*:

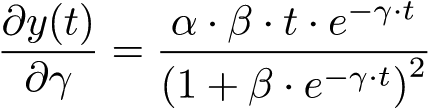
**Pochodna po** *α***:**



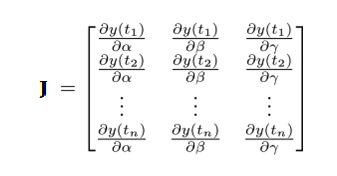
**Pochodna po** *β***:**



**Pochodna po** *γ***:**



# Macierz Jacobiego

Dla *n* punktów danych *t*1*,t*2*,...,tn*, macierz Jacobiego **J** ma postać:

# Równanie Newtona-Gaussa

Zmiana parametrow w *k*-tej iteracji obliczana jest wedlug wzoru:

**r**

gdzie:

* ∆*θ* = [∆*α,*∆*β,*∆*γ*]*T* – wektor zmian parametr´ow,
* **J** – macierz Jacobiego,
* **r** = [*r*1*,r*2*,...,rn*]*T*, gdzie *ri* = *yi* − *y*ˆ*i*, to wektor reszt,
* **J***T***J** – aproksymacja macierzy Hessego,
* **J***T***r** – gradient funkcji błędu

# Aktualizacja parametrów

Nowe wartości parametrów po iteracji obliczamy jako:

nowe = stare + ∆

gdzie: = [*α,β,γ*]*T*

**Wnioski:**

* Model dobrze dopasował funkcję logistyczną do danych.
* Początkowa szybka redukcja kosztów świadczy o skuteczności metody w znajdowaniu przybliżonych wartości parametrów.
* Ostateczne parametry funkcji logistycznej: α=310.38, β=91.34, γ=0.0687
* Liczba iteracji została ograniczona do 15, ale algorytm zakończył pracę po 11 iteracjach, co wskazuje na szybkie zbieżne rozwiązanie.

**4. Podsumowanie: Analiza liczby błędów krytycznych w systemie operacyjnym**

**Cel i kontekst**

Celem analizy było zbadanie liczby błędów krytycznych w systemie operacyjnym w trakcie jego eksploatacji. Wykorzystano dwa podejścia modelowania: regresję liniową oraz funkcję logistyczną. Badanie miało na celu określenie trendów oraz prognozowanie dalszego przebiegu liczby błędów.

**Metodologia**

1. **Model liniowy:**
   * Regresja liniowa wykazała wysoką zgodność danych z modelem, z wartością R² równą 0,938.
   * Parametry modelu były istotne statystycznie, wskazując na systematyczny wzrost skumulowanej liczby błędów w kolejnych miesiącach.
2. **Funkcja logistyczna:**
   * Model ten pozwolił na analizę nasycenia liczby błędów w czasie.
   * Ostateczne parametry funkcji logistycznej wynosiły:
     + α = 310,38 (maksymalna liczba błędów),
     + β = 91,34 (przesunięcie w czasie),
     + γ = 0,0687 (tempo wzrostu).
   * Algorytm zbiegał do rozwiązania po 11 iteracjach, osiągając niskie wartości błędu.

**Wyniki**

* Dane z analizy regresji liniowej wykazały, że liczba błędów rośnie systematycznie w czasie, mimo niskiej korelacji dla miesięcznych zmian liczby błędów.
* Funkcja logistyczna umożliwiła oszacowanie maksymalnej liczby błędów oraz wizualizację procesu ich narastania.
* Oba modele potwierdziły skuteczność w opisie dynamiki błędów, przy czym funkcja logistyczna lepiej oddawała charakter nasycenia procesu.

**Wnioski**

1. **Dane wskazują na systematyczny wzrost skumulowanej liczby błędów, co sugeruje potencjalną potrzebę optymalizacji systemu operacyjnego.**
2. **Funkcja logistyczna pozwala na długoterminowe prognozy, dzięki czemu może być narzędziem wspierającym zarządzanie stabilnością systemu.**
3. **Wysoka zgodność obu modeli z danymi świadczy o poprawności przyjętej metodologii.**

Kod źródłowy oprogramowania:

* from sklearn.model\_selection import train\_test\_split  
  import pandas as pd  
  import numpy as np  
  import statsmodels.api as sm  
  import matplotlib.pyplot as plt  
  import plotly.graph\_objects as go  
  from scipy.stats import pearsonr  
  import numpy as np  
  from scipy.stats import chi2  
  from numpy.linalg import inv   
    
  # Read the data from the CSV file  
  data = pd.read\_csv('Dane\_awarii.csv', sep=';', header=0)  
    
  # Plot the number of errors (`Liczba\_bledow`) for each month (`Nr\_miesiaca`)  
  plt.plot(data['Nr\_miesiaca'], data['Liczba\_bledow'])  
  plt.title("Zaleznosc liczba bledow od numeru miesiaca")  
  plt.xlabel("Numer miesiaca")  
  plt.ylabel("Liczba bledow")  
  plt.grid(True)  
  plt.show()  
    
  # Calculate the Pearson correlation coefficient between 'Liczba\_bledow' and 'Nr\_miesiaca'  
  corr, \_ = pearsonr(data['Liczba\_bledow'], data['Nr\_miesiaca'])  
  print(f"Pearson correlation: {corr}")  
    
  # Add a new column 'Wystapienia\_lacznie' to store the cumulative number of errors  
  data['Wystapienia\_lacznie'] = 0  
    
  # Loop through all rows to calculate the cumulative sum of errors  
  for i in range(len(data)):  
   # Cumulative sum of errors up to and including the current row  
   data.loc[i, 'Wystapienia\_lacznie'] = data['Liczba\_bledow'][:i + 1].sum()  
    
    
  corr, \_ = pearsonr(data['Wystapienia\_lacznie'], data['Nr\_miesiaca'])  
  print(f"Pearson correlation: {corr}")  
    
    
  X=sm.add\_constant(data['Nr\_miesiaca'])  
  y=data['Wystapienia\_lacznie']  
  wynik= sm.OLS(y, X).fit()  
  print(wynik.summary())  
    
  # Plot the cumulative sum of errors for visualization  
  plt.plot(data['Nr\_miesiaca'], wynik.predict(X), label='Regresja liniowa', color='red')  
  plt.plot(data['Nr\_miesiaca'], data['Wystapienia\_lacznie'], label='Skumulowane błędy', color='orange')  
  plt.title("Zaleznosc sumarycznej liczby bledow od miesiaca")  
  plt.xlabel("Numer miesiaca")  
  plt.ylabel("liczba bledow")  
  plt.grid(True)  
  plt.legend()  
  plt.show()  
  #regresja  
    
  # wzor regresji  
  # sza  
    
  # napisz funcje logistyczna metodą szacowania parametrow hottellinga  
  from scipy.optimize import least\_squares  
  import numpy as np  
  import matplotlib.pyplot as plt  
    
    
  # Funkcja logistyczna  
  def logistic\_function(t, alpha, beta, gamma):  
   return alpha / (1 + beta \* np.exp(-gamma \* t))  
    
    
  # Funkcja błędu (różnica między danymi rzeczywistymi a modelowanymi)  
  def residuals(params, t, y):  
   alpha, beta, gamma = params  
   return y - logistic\_function(t, alpha, beta, gamma)  
    
    
  # Dopasowanie funkcji logistycznej  
  x\_data = data['Nr\_miesiaca']  
  y\_data = data['Wystapienia\_lacznie']  
    
  # Początkowe wartości parametrów  
  initial\_guess = [max(y\_data), 1, 0.1]  
    
  # Ustawienie maksymalnej liczby iteracji  
  max\_iterations = 15 # Tutaj ustawisz maksymalną liczbę iteracji  
    
  # Dopasowanie funkcji za pomocą least\_squares z ograniczeniami  
  result = least\_squares(  
   residuals, # Funkcja błędu  
   initial\_guess, # Wartości początkowe  
   args=(x\_data, y\_data), # Argumenty przekazywane do funkcji błędu  
   bounds=([max(y\_data) \* 0.8, 0, 0], [max(y\_data) \* 2.0, np.inf, 1.0]), # Ograniczenia  
   verbose=2, # Włączenie szczegółów iteracji  
   max\_nfev=max\_iterations # Maksymalna liczba iteracji  
  )  
    
  # Wyciągnięcie dopasowanych parametrów  
  alpha, beta, gamma = result.x  
  print(f"Parametry funkcji logistycznej: alpha={alpha:.2f}, beta={beta:.2f}, gamma={gamma:.4f}")  
    
  # Przewidywanie wartości przy pomocy dopasowanej funkcji  
  data['Logistic\_Prediction'] = logistic\_function(x\_data, alpha, beta, gamma)  
    
  # Wizualizacja  
  plt.figure(figsize=(10, 6))  
  plt.plot(data['Nr\_miesiaca'], data['Wystapienia\_lacznie'], label='Skumulowana liczba błędów', color='orange',  
   marker='o')  
  plt.plot(data['Nr\_miesiaca'], data['Logistic\_Prediction'], label='Funkcja logistyczna', color='blue', linestyle='--')  
  plt.xlabel('Numer miesiąca')  
  plt.ylabel('Skumulowana liczba błędów')  
  plt.title(f'Analiza dopasowania funkcji logistycznej (limit iteracji: {max\_iterations})')  
  plt.legend()  
  plt.grid(True)  
  plt.show()