## SPRAWOZDANIE Z PROJEKTU: KOD HADAMARDA

# Cel projektu

Celem projektu było **poznanie i praktyczne zastosowanie kodu Hadamarda**, który jest jednym z kodów korekcji błędów (FEC – Forward Error Correction).

Chciałem dowiedzieć się:

* Jak działa kod Hadamarda?
* Jak kodować i dekodować dane za pomocą macierzy Hadamarda?
* Jak błędy wpływają na skuteczność dekodowania?
* W jakich warunkach kod Hadamarda działa poprawnie, a kiedy pojawiają się błędy?

# **Czego się nauczyłem?**

Podczas realizacji projektu zdobyłem następujące umiejętności:

* **Zrozumienie działania kodu Hadamarda** – jak działa i dlaczego jest odporny na błędy.
* **Generowanie macierzy Hadamarda** o różnych wymiarach.
* **Kodowanie i dekodowanie wiadomości** za pomocą iloczynu skalarnego.
* **Dodawanie błędów do sygnału i testowanie ich wpływu na dekodowanie**.
* **Analiza skuteczności kodowania Hadamarda** w różnych warunkach (różne poziomy błędów).
* **Ile błędów może wykrzyć (2^(r-1))-1 i korygować ((2^(r-1))-1)/2**
* **Dekodowane możliwe przez to że wiersze macierzy Hadamarda są ortogonalne**
* **Konwertacja z [-1 ; 1] na binarne [0 ; 1] używamy wzoru (kod\_hadamarda + 1)//2**
* **Z [0 ; 1] na [-1 ; 1] używamy wzoru (2 \* kod\_hadamarda)-1**
* Dlaczego przy 20% błędów, dekodowanie może nie być prawidłowe-dlatego że Kod Hadamarda może poprawić do 3 błędów na blok, ale jeśli w jednym bloku wystąpi więcej błędów dekodowanie nie powiedzie się
* Dekodować przy pomocy odłegłości Hamminga  
  Pilega na tym żeby porównać blok z błędami z każdym wierszem macierzy Hadamarda  
  wynikiem będzie ten wiersz, w którym będzie najmniesza ilość różnic, czyli ten, który będzie najbardziej podobny do otrzymanego sygnału  
  Wzór: Зображення, що містить текст, Шрифт, годинник, число

  Автоматично згенерований опис

Krok po kroku:  
1. Konwertujemy odebrany blok z wartościami 0 i 1.

2. Porównujemy go z każdym możliwym wierszem kodowym.

3. Wybieramy ten, który ma **najmniej różnic bitowych (najmniejszą odległość Hamminga)**.

# **Opis metody badawczej**

Aby zbadać skuteczność kodu Hadamarda, przeprowadziliśmy testy dla różnych poziomów błędów.

**Etapy badań:**  
 **1.Wygenerowanie macierzy Hadamarda** dla r=4 (rozmiar 16×16).  
 **2.Losowe generowanie 8-bitowych liczb** do testów.  
 **3.Podział liczby na bloki 4-bitowe**.  
 **4.Kodowanie bloków** za pomocą macierzy Hadamarda.  
 **5.Dodawanie błędów na poziomach 10%, 20%, 30%, 40%**.  
 **6.Dekodowanie zakłóconych bloków**.  
 **7.Sprawdzenie, czy dekodowana liczba jest zgodna z oryginalną**.  
 **8.Obliczenie procentu poprawnie odczytanych liczb**.

Każdy test został powtórzony **1000 razy** dla każdego poziomu szumu, aby wyniki były wiarygodne.

# Wyniki badań

Зображення, що містить текст, знімок екрана, Шрифт

Автоматично згенерований опис

**Obserwacje:**

* Przy **10% błędów** dekodowanie działa **niemal perfekcyjnie**.
* Przy **20% błędów** skuteczność spada, ale nadal jest wysoka .
* Przy **30% błędów** zaczynają się **poważne zacłócenia w dekodowaniu**.
* Przy **40% błędów** skuteczność jest już bardzo niska

# Dlaczego pojawiają się błędy przy 20% szumu?

Teoretycznie, macierz Hadamarda dla r=4 powinna korygować do **3 błędów na blok 16-bitowy**, ale w praktyce:

1️.**Błędy mogą skupić się w jednym miejscu**

* Jeśli więcej niż 3 bity w jednym bloku ulegną zmianie, kod Hadamarda **nie może poprawnie zidentyfikować oryginalnej sekwencji**.

2️.**Losowe błędy mogą mocno zmienić skalarne iloczyny**

* Dekoder wybiera najbardziej pasujący wiersz macierzy Hadamarda na podstawie **iloczynu skalarnego**.
* Jeśli błędy zmieniają zbyt wiele bitów, **iloczyn może wskazać zły wiersz**, co prowadzi do błędnego dekodowania.

3️.**Rozkład błędów nie jest idealnie losowy**

* W teorii błędy są rozłożone równomiernie, ale w rzeczywistości mogą skupić się na jednym bloku.

# Wnioski końcowe

1️.**Kod Hadamarda bardzo dobrze działa przy małym poziomie błędów** (do **10-15%**).  
2️.**Gdy poziom błędów przekracza 20%, dekodowanie przestaje być skuteczne**.  
3️.**Kod Hadamarda nie radzi sobie dobrze z błędami skumulowanymi w jednym miejscu**.  
4️.**Przy 30% błędów skuteczność spada poniżej 70%, co oznacza, że co trzecia wiadomość jest błędnie dekodowana**.  
5️.**W praktyce, aby zwiększyć odporność na błędy, warto używać większych macierzy Hadamarda lub dodatkowych metod korekcji błędów**.

# Jak można poprawić skuteczność kodowania?

* **Zwiększenie rozmiaru macierzy Hadamarda** – jeśli użyjemy r=5 (macierz 32×32), możemy poprawić skuteczność.
* **Dodatkowe kody korekcji błędów** – np. kod Hamming lub Reed-Solomon.
* **Inna strategia kodowania danych** – np. użycie filtrów redukujących wpływ błędów.
* **Lepsza kontrola nad błędami**– jeśli można przewidzieć charakterystykę błędów, można dopasować lepszą strategię korekcji.

# Wnioski

* Nauczyłem się, jak działa kod Hadamarda i jak stosuje się go do korekcji błędów.
* Zrozumiałem, dlaczego kod Hadamarda działa dobrze w pewnych warunkach, a w innych zawodzi.
* Przeprowadzone badania pokazały, że kod Hadamarda dobrze radzi sobie do pewnego poziomu błędów, ale ma swoje ograniczenia.
* Dzięki temu projektowi zdobyłem praktyczne doświadczenie w korekcji błędów i lepiej zrozumiałem, jak stosuje się te techniki w komunikacji cyfrowej.

# Kody

# Kod z implementacją metody Hadamarda

import numpy as np  
def generuj\_macierz\_hadamarda(r):  
 if r == 1:  
 return np.array([[1, 1], [1, -1]])  
 else:  
 poprzednia\_macierz = generuj\_macierz\_hadamarda(r - 1)  
 return np.block([  
 [poprzednia\_macierz, poprzednia\_macierz],  
 [poprzednia\_macierz, -poprzednia\_macierz]  
 ])  
  
def tekst\_na\_binarne(tekst):  
 binarne = ''.join(format(ord(znak), '08b') for znak in tekst)  
 return [int(bit) for bit in binarne]  
def podziel\_na\_bloki(kod\_binarne, r):  
 bloki = []  
 for i in range(0, len(kod\_binarne), r):  
 blok = kod\_binarne[i:i + r]  
 while len(blok) < r:  
 blok.append(0)  
 bloki.append(blok)  
 return bloki  
def koduj\_blok(blok, macierz\_hadamarda):  
 indeks = int("".join(map(str, blok)), 2)  
 return macierz\_hadamarda[indeks]  
def dodaj\_bled(zakodowane\_bloki, poziom\_bledu):  
 Z\_Bledami = []  
 for blok in zakodowane\_bloki:  
 bled = np.random.choice([-1, 1], size=len(blok), p=[poziom\_bledu, 1 - poziom\_bledu])  
 Z\_Bledami.append(blok \* bled)  
 return Z\_Bledami  
def dekoduj\_blok(otrzymany\_blok, macierz\_hadamarda):  
 odleglosci = np.dot(macierz\_hadamarda, otrzymany\_blok)  
 indeks\_max = np.argmax(odleglosci)  
 return indeks\_max  
def dekoduj\_wiadomosc(Z\_Bledami, macierz\_hadamarda, r):  
 dekodowane\_binarne = []  
 for blok in Z\_Bledami:  
 indeks = dekoduj\_blok(blok, macierz\_hadamarda)  
 binarny\_blok = [int(bit) for bit in bin(indeks)[2:].zfill(r)]  
 dekodowane\_binarne.extend(binarny\_blok)  
 return dekodowane\_binarne  
def binarne\_na\_tekst(kod\_binarne):  
 znaki = [chr(int("".join(map(str, kod\_binarne[i:i + 8])), 2)) for i in range(0, len(kod\_binarne), 8)]  
 return ''.join(znaki)  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 tekst = input("Wprowadź tekst do zakodowania: ")  
 r = int(input("Wprowadź wymiar r (np. 2, 3, 4): "))  
 poziom\_bledu = float(input("Podaj poziom bledu (np. 0.1 dla 10% bledu): "))  
  
 macierz\_hadamarda = generuj\_macierz\_hadamarda(r)  
 print("Macierz Hadamarda:")  
 print(macierz\_hadamarda)  
  
 kod\_binarne = tekst\_na\_binarne(tekst)  
 print("\nKod binarny wiadomości:")  
 print(kod\_binarne)  
  
 bloki = podziel\_na\_bloki(kod\_binarne, r)  
 print("\nBloki binarne:")  
 print(bloki)  
  
 zakodowane\_bloki = [koduj\_blok(blok, macierz\_hadamarda) for blok in bloki]  
 print("\nZakodowane bloki:")  
 for blok in zakodowane\_bloki:  
 print(blok)  
  
 Z\_Bledami = dodaj\_bled(zakodowane\_bloki, poziom\_bledu)  
 print("\nBloki z bledami:")  
 for blok in Z\_Bledami:  
 print(blok)  
  
 dekodowane\_binarne = dekoduj\_wiadomosc(Z\_Bledami, macierz\_hadamarda, r)  
 print("\nDekodowane binarne:")  
 print(dekodowane\_binarne)  
  
 dekodowany\_tekst = binarne\_na\_tekst(dekodowane\_binarne)  
 print("\nDekodowany tekst:")  
 print(dekodowany\_tekst)

# Kod z badań

import numpy as np  
from Hadamard import generuj\_macierz\_hadamarda, tekst\_na\_binarne, podziel\_na\_bloki, koduj\_blok, dodaj\_bled, dekoduj\_wiadomosc, binarne\_na\_tekst  
  
def przeprowadz\_badania(liczba\_testow=1000):  
 r = 4  
 macierz\_hadamarda = generuj\_macierz\_hadamarda(r)  
  
 poziomy\_bledu = [0.1, 0.2, 0.3, 0.4]  
 wyniki = {bled: 0 for bled in poziomy\_bledu}  
  
 for bled in poziomy\_bledu:  
 poprawne = 0  
 for \_ in range(liczba\_testow):  
 liczba = np.random.randint(0, 256)  
 binarna = format(liczba, '08b')  
 bloki = podziel\_na\_bloki([int(bit) for bit in binarna], r)  
  
 zakodowane\_bloki = [koduj\_blok(blok, macierz\_hadamarda) for blok in bloki]  
 Z\_Bledami = dodaj\_bled(zakodowane\_bloki, bled)  
 dekodowane\_binarne = dekoduj\_wiadomosc(Z\_Bledami, macierz\_hadamarda, r)  
 dekodowana\_liczba = int("".join(map(str, dekodowane\_binarne[:8])), 2)  
  
 if dekodowana\_liczba == liczba:  
 poprawne += 1   
  
 wyniki[bled] = poprawne / liczba\_testow  
  
 return wyniki  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 wyniki\_badan = przeprowadz\_badania()  
 print("\n=== Wyniki badań ===")  
 for bled, skutecznosc in wyniki\_badan.items():  
 print(f"Poziom bledu: {bled\*100:.0f}%, Skuteczność dekodowania: {skutecznosc\*100:.2f}%")