SPRAWOZDANIE Z PROJEKTU: KOD HADAMARDA

1. 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?

2. 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 odległości Hamminga
  Pilega na tym żeby porównać blok z błędami z każdym wierszem macierzy Hadamarda wynikiem będzie ten wiersz, w którym będzie najmniesza ilość różnic
  Wzór: $$d_H(A,B) = \sum_{i=1}^{n} |A_i - B_i|$$

3. 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.

4. Wyniki badań

```
Poziom bledu: 10%, Skuteczność dekodowania: 95.30%
Poziom bledu: 20%, Skuteczność dekodowania: 62.90%
Poziom bledu: 30%, Skuteczność dekodowania: 22.20%
Poziom bledu: 40%, Skuteczność dekodowania: 3.80%
```

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

5. 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:

1. **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**.
2. **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.
3. **Rozkład błędów nie jest idealnie losowy**
   - W teorii błędy są rozłożone równomiernie, ale w rzeczywistości mogą skupić się na jednym bloku.

6. Wnioski końcowe

1. **Kod Hadamarda bardzo dobrze działa przy małym poziomie błędów** (do **10-15%**).
2. **Gdy poziom błędów przekracza 20%, dekodowanie przestaje być skuteczne**.
3. **Kod Hadamarda nie radzi sobie dobrze z błędami skumulowanymi w jednym miejscu**.
4. **Przy 30% błędów skuteczność spada poniżej 70%, co oznacza, że co trzecia wiadomość jest błędnie dekodowana**.

5.**W praktyce, aby zwiększyć odporność na błędy, warto używać większych macierzy Hadamarda lub dodatkowych metod korekcji błędów**.

7. 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.

8. 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}%")
```