Lab 6 - opis kodu

I. Zadanie 1

```
    H_curr = x_poly * H_prev - (m - 1) * H_prevprev
```

Julia dopuszcza mnożenie wielomianów z pakietu polynomias same przez siebie oraz przez stałe. Zapisuje się to poprostu jako $f(x) * g(x)$

```
    y = Hn.(x)
```

To wywołuje funkcję Hn na każdym elemencie z wektora x

```
    y = Hn.(x) .* exp.(-0.5 .* x.^2)
```

x.^2 podnieś każdy element wektora x do potęgi 2
-0.5 .* x pomnóż każdą wartość z wektora z przez $-\frac{1}{2}$
X .* exp.(Y) wyjdzie: $[x_1 * e^{y_1}, x_2 * e^{y_2}, ...]$

II. Zadanie 2

```
Matrix{Float64}(undef, 6, 6)
```

Tworzy macierz o wymiarach 6×6, w której każdy element będzie typu Float64, a wartości są nieinicjalizowane (na początku randomowe)
Prykłady:

```
B = Matrix{Int}(I, 3, 3)              # macierz jednostkowa typu Int
C = fill(3.14, 2, 5)                  # 2x5 wypełniona wartością 3.14
D = rand(5, 5)                        # losowa 5x5
E = zeros(Float64, 3, 2)             # zera 3x2
```

```
for (i, vec_i) in enumerate(poly_vals)
  for (j, vec_j) in enumerate(poly_vals)
      scalar_prods_no_weight[i, j] = dot(vec_i, vec_j)
  end
end
```

vec_i --> wektor wartości i-tego wielomianu hermita
dot(vec_i, vec_j) --> iloczyn wektorowy

```
basis(Hermite, i)
```

Tworzę wielomian hermita z pakietu SpecialPolynomials

```
normalize(vector)
```

Zwraca znormalizowany wektor

III. Zadanie 3

```
function gradient_color(i, total)
  alpha = i / (total - 1)
  return RGB(alpha, 1 - alpha, alpha)
end
```

$$\alpha = \frac{1}{\text{total} - 1}$$

$$\text{RGB} = (\alpha, 1 - \alpha, \alpha)$$

```
Vs = [c * y for (c, y) in zip(cs, ys)]
```

To tworzy listę wektorów, gdzie każdy wektor to:
$\text{Vs}\ [i] = [c_i * y_1, c_i * y_2, ...]$

```
partial_sums = cumsum(Vs)
```

cumsum(Vs) sumuje kolejne wektory punkt po punkcie czyli:

```
partial_sums[1] = Vs[1]
partial_sums[2] = Vs[1] .+ Vs[2]
partial_sums[3] = Vs[1] .+ Vs[2] .+ Vs[3]
```

IV. Zadanie 4

```
for k in 0:max_k
    hermite_functions = [hermite_basis_function(i, xx_ekg) for i in 0:k]
    dot_prods = [dot(yy_ekgN, y) for y in hermite_functions]
    Vs = [c * y for (c, y) in zip(dot_prods, hermite_functions)]
    approx = sum(Vs)
    err = norm(yy_ekgN - approx)
```

Wzór z którego korzystam:

$$\varepsilon = \|F - \sum_{j=0}^{k} a_j \varphi_j\|$$

gdzie:
F - Orginalny sygnał EKG
φ_j - funkcje bazowe hermita
a_j - czynniki rozwinięcia
$\sum_{j=0}^{k} a_j \varphi_j$ - przebliżenie funkcji F za pomocą k+1 f bazowych

```
approx = sum(Vs)
```

Sumuje wartości 'i' w liście wektorów czyli:
approx $[i] = V_1[i] + V_2[i] + ... = \text{Vs } [1][i] + \text{Vs } [2][i] + ...$