

*Zahvaljujem mentorici izv. prof. dr. sc. Mihaeli Vranić na mentorstvu i vodstvu te  
fleksibilnosti i razumijevanju tijekom izrade ovog rada.*

# Sadržaj

|  |    |
|--|----|
| <b>1. Uvod</b>                                   | 3  |
| <b>2. Teorija portfelja i matematičke osnove</b> | 5  |
| 2.1. Investicije                                 | 5  |
| 2.2. Portfelj                                    | 5  |
| 2.3. Povrati                                     | 6  |
| 2.3.1. Aritmetički povrat                        | 6  |
| 2.3.2. Logaritamski povrat                       | 6  |
| 2.3.3. Očekivani povrat                          | 6  |
| 2.4. Volatilnost                                 | 7  |
| 2.5. Geometrijsko Brownovo gibanje               | 7  |
| 2.6. Monte Carlo simulacije                      | 8  |
| 2.7. Cholesky dekompozicija                      | 9  |
| 2.8. PCA dekompozicija                           | 9  |
| <b>3. Implementacija aplikacije</b>              | 11 |
| 3.1. Podatci                                     | 11 |
| 3.2. Algoritmi i komponente                      | 13 |
| 3.2.1. Parsiranje podataka                       | 13 |
| 3.2.2. Model kriptovalute                        | 14 |
| 3.2.3. Model portfelja                           | 15 |
| 3.2.4. Matematički okvir                         | 16 |
| 3.2.5. Monte Carlo simulacija                    | 20 |
| 3.2.6. PCA dekompozicija                         | 22 |
| 3.2.7. Vizualizacija rezultata                   | 23 |

|  |           |
|--|-----------|
| <b>4. Rezultati i rasprava</b>                         | <b>25</b> |
| 4.1. Analiza i diskusija implementacijskih rezultata   | 25        |
| 4.1.1. Funkcionalnost učitavanja i parsiranja podataka | 25        |
| 4.1.2. Matematički okvir                               | 25        |
| 4.1.3. Monte Carlo simulacija                          | 26        |
| 4.1.4. PCA dekompozicija                               | 31        |
| 4.1.5. Vizualizacija rezultata                         | 32        |
| <b>5. Upute za korištenje i pokretanje aplikacije</b>  | <b>34</b> |
| <b>6. Zaključak</b>                                    | <b>35</b> |
| <b>Literatura</b>                                      | <b>37</b> |
| <b>Sažetak</b>   | <b>39</b> |
| <b>Abstract</b>  | <b>40</b> |

# 1. Uvod

Modeliranje ponašanja portfelja je jedna od ključnih metoda pri odabiru investicijskog pristupa. Od odabira pojedinačnih dionica, raznih derivata, sigurnih obveznica ili investicijskih fondova sve do mogućnosti čuvanja novčanih rezervi.

U posljednjem desetljeću, kriptovalute su postale sveprisutna komponenta financijskih tržišta, karakterizirana visokom volatilnošću, nelinearnim ovisnostima i globalnom dostupnošću što kroz iznimno pouzdane izvore što kroz izrazito nepouzdane izvore. Upravljanje rizikom u takvom okruženju zahtijeva napredne alate za modeliranje budućih scenarija. Jedan od takvih alata u standarnim financijskim modelima je Monte Carlo simulacija.

Monte Carlo simulacija, kao statistička metoda temeljena na ponovljenom uzorkovanju slučajnih varijabli i često korištena metoda u modeliranju ostalih financijskih instrumenata, nameće se kao potencijalno uspješan pristup za analizu portfelja kriptovaluta.

U ovom radu fokusira se na ručnoj implementaciji svih matematičkih operacija i modela u C++ programskom jeziku bez korištenja vanjskih biblioteka potrebnih za modeliranje i provođenje Monte Carlo simulacija. Cilj je stečeno znanje na kolegijima primjeniti na barem naivne implementacije kako bi se moglo usporediti rad takog pristupa sa razvijenim i maksimalno optimiziranim bibliotekama.

To uključuje Monte Carlo metodu za predviđanje vrijednosti portfelja s primjenom Cholesky dekompozicije kako bi se osigurala realistična obrada korelacija između kriptovaluta koje su još uvijek specijalna skupina investicija s visokom međusobnom korelacijom.

Osim Monte Carlo simulacija za predviđanje budućih cijena portfelja, implementira se i PCA (Principal Component Analysis) dekompozicija kao jedna od jednostavnih metoda za efikasnu diverzifikaciju portfelja i samim time smanjenje rizika upravo kroz smanjenje dimenzionalnosti podataka i identifikaciju glavnih komponenti koje objašnjavaju najveći dio varijance podataka.

Glavni izazov u modeliranju kriptovaluta leži u njihovoј inherentnoj nestabilnosti. Dok

tradicionalne finansijske instrumente karakteriziraju relativno predvidljivi obrasci, kriptovalute pokazuju ekstremne fluktuacije koje zahtijevaju precizno podešavanje parametara poput driftova i volatilnosti. Osim velike i teško predvidive volatilnosti dodatan problem predstavlja kratak period postojanja tog finansijskog instrumenta. Većina kriptovaluta postoji tek nekoliko godina, a neke su čak i nestale samo nekoliko mjeseci nakon što su se pojavile. Period manji od 10 godina je u finansijskom svijetu vrlo kratak pogotovo ako uzmemu u obzir da nije bilo prevelikih promjena ili kriza na finansijskim tržištima u vremenu postojanja kriptovaluta.

U radu je razvijen C++ programski okvir koji integrira povijesne podatke kriptovaluta, obavlja potrebne matematičke operacije i generira simulacije. Generirane simulacije omogućuju analizu različitih scenarija kretanja cijena, a korisnik može odabrat različite portfelje i vremenske okvire. Sva interakcija s korisnikom odvija se putem grafičkog sučelja koje omogućuje jednostavno upravljanje parametrima simulacije i vizualizaciju rezultata.

Rad je strukturiran kako slijedi: U drugom poglavlju opisuju se teorijske osnove teorije portfelja, Monte Carlo metode, PCA dekompozicija te sve potrebne matematičke osnove i teorije potrebne za razumijevanje i implementaciju modela. Treće poglavlje detaljno opisuje implementaciju algoritama i struktura, uključujući model portfelja, model kriptovalute, postupke poput Cholesky dekompozicije, traženja svojstvene dekompozicije, brzog "parsiranja" finansijskog skupa podataka i optimizacije za velike skupove podataka. U četvrtom poglavlju analiziraju se rezultati simulacija za različite konfiguracije portfelja, dok se u zaklučku raspravlja o primjenjivosti modela, mogućim i očitim praktičnim problemima i smjerovima dalnjeg istraživanja.

Ovakav rad može poslužiti kao osnova za daljnje istraživanje i razvoj naprednijih modela i aplikacija koji će omogućiti bolje razumijevanje i upravljanje rizicima povezanim s kriptovalutama, ali i ostalim finansijskim instrumentima. Pogotovo jer se kroz izradu sustava i aplikacije kao sekundarni rezultat pokrenuo razvitak biblioteke za rad s finansijskim podatcima i potrebnim matematičkim okvirima koja će se moći koristiti u budućim radovima i projektima.

## 2. Teorija portfelja i matematičke osnove

Teorija portfelja, čiji su začetnici Harry Markowitz i James Tobin, daje strogu matematičku definiciju financijskim pojmovima. Ključni optimizacijski problem teorije portfelja je *dualni cilj*: maksimizacija očekivanog povrata uz istovremeno minimiziranje rizika.

### 2.1. Investicije

Investicija ili ulaganje je proces alokacije ograničenih resursa kao što su novac, vrijeme ili energija u svrhu ostvarivanja većeg povrata ili dodatne koristi u budućnosti [1]. Ovime kažemo kako investicija ne mora nužno biti financijska, ali u ovom radu se fokusiramo na financijske investicije koje nam predstavljaju izdvajanje i ulaganje novca u različite financijske instrumente kroz određeni vremenski period s ciljem ostvarivanja većeg povrata u budućnosti ili smanjenje gubitka vrijednosti novca.

### 2.2. Portfelj

Investicijske portfelje matematički prikazujemo kao linearnu kombinaciju pojedinih investicija s vektorom pojedinih udjela  $\mathbf{w}$ .

**Definicija 1.** Vektor  $\mathbf{w}$  predstavlja udjele investicija u portfelju.

$$\mathbf{w} = \begin{pmatrix} w_1 \\ w_2 \\ \dots \\ w_N \end{pmatrix}, \quad \sum_{i=1}^N w_i = 1 \quad (2.1)$$

## 2.3. Povrati

Povrat investicije je osnovna mjera uspješnosti investicije.

### 2.3.1. Aritmetički povrat

**Definicija 2.** Neka je  $P_t$  cijena financijskog instrumenta u trenutku  $t$  te  $P_{t-1}$  cijena istog instrumenta u trenutku  $t - 1$ . Aritmetički povrat  $R_t$  definiramo kao:

$$R_t = \frac{P_t}{P_{t-1}} - 1 = \frac{P_t - P_{t-1}}{P_{t-1}} = \frac{\Delta P}{P_{t-1}} \quad (2.2)$$

Neka buduća cijena nam neće biti poznata pri investiranju te zato povrat promatramo kao slučajnu varijablu. Vidimo kako je moguće imati negativan povrat ako je cijena koju promatramo manja od početne cijene i to je upravo situacija koju nastojimo izbjegći.

### 2.3.2. Logaritamski povrat

Logaritamski povrat  $r_t$  definiramo preko prirodnog logaritma omjera cijena.

**Definicija 3.** Neka je  $P_t$  cijena financijskog instrumenta u trenutku  $t$  te  $P_{t-1}$  cijena istog instrumenta u trenutku  $t - 1$ . Logaritamski povrat  $r_t$  definiramo kao:

$$r_t = \ln\left(\frac{P_t}{P_{t-1}}\right) = \ln(P_t) - \ln(P_{t-1}) \quad (2.3)$$

Logaritamski povrat u pravilu koristimo zbog njegovih pogodnih matematičkih svojstava kao što je svojstvo simetrije  $\ln(a) = -\ln(1/a)$  te svojstvo aditivnosti  $r_{0,T} = \sum_{t=1}^T r_t$ .

### 2.3.3. Očekivani povrat

**Definicija 4.** Očekivani povrat promatramo kao srednju vrijednost prijašnjih povrata jer je upravo srednja vrijednost nepristran procjenitelj očekivanja slučajne varijable  $R_t$  za koji vrijedi:

$$E(R_t) = \frac{1}{N} \sum_{i=1}^N R_i \quad (2.4)$$

**Definicija 5.** Očekivani povrat portfelja je linearna kombinacija očekivanih povrata pojedinačnih asseta:

$$\mathbb{E}[R_p] = \mathbf{w}^\top \boldsymbol{\mu} = \sum_{i=1}^n w_i \mu_i \quad (2.5)$$

gdje je  $\boldsymbol{\mu} = (\mu_1, \dots, \mu_n)^\top$  vektor očekivanih povrata.

## 2.4. Volatilnost

Drugi dio optimizacijskog problema teorije portfelja je smanjenje rizika. Volatilnost je upravo jednostavna mjera rizika koja ima pogodna matematička svojstva. Promatramo je kao standardnu devijaciju slučajne varijable  $R_t$ , a ima je smisla tako promatrati jer će nam upravo takva mjera kvantificirati kretanje povrata.

**Definicija 6.** Volatilnost investicije definiramo kao nepristran procjenitelj standardne devijacije slučajne varijable  $R_t$ :

$$\sigma_R = \sqrt{\frac{1}{N-1} \sum_{i=1}^N [R_i - E(R_t)]^2} \quad (2.6)$$

**Definicija 7.** Volatilnost portfelja mjeri se standardnom devijacijom povrata i dana je kvadratnim korijenom varijance:

$$\sigma_p = \sqrt{\mathbf{w}^\top \boldsymbol{\Sigma} \mathbf{w}} \quad (2.7)$$

gdje je  $\boldsymbol{\Sigma}$  matrica kovarijance s elementima  $\Sigma_{ij} = \text{Cov}(r_i, r_j)$ .

## 2.5. Geometrijsko Brownovo gibanje

Geometrijsko Brownovo gibanje (GBM) je jedan od standardnih stohastičkih procesa za modeliranje kretanja cijena financijskih instrumenata. Diferencijalna jednadžba GBM-a

je:

$$dS_t = \mu S_t dt + \sigma S_t dW(t) \quad (2.8)$$

gdje je  $W(t)$  Wienerov proces.

Eksplisitno rješenje GBM-a daje formulu za cijenu u trenutku  $t$ :

$$S_t = S_0 \exp \left[ \left( \mu - \frac{\sigma^2}{2} \right) t + \sigma W_t \right] \quad (2.9)$$

gdje je  $S_0$  početna cijena,  $\mu$  drift,  $\sigma$  volatilnost i  $W_t$  Wienerov proces.

Detaljno objašnjene GBM-a i njegovih svojstava te kompletan derivacijski postupak možete pronaći u [2].

## 2.6. Monte Carlo simulacije

Monte Carlo metoda je numerička metoda koja koristi slučajno uzorkovanje za rješavanje problema. Jedan iznimno intuitivan i elegantan primjer je određivanje vrijednosti broja  $\pi$ . Ideja je generiranje što većeg broja točaka unutar jediničnog kvadrata koji u sebi sadrži jedinični krug te određivanje omjera broja točaka unutar kruga i ukupnog broja točaka te onda preko omjera površina kvadrata i kruga dobijemo procjenu vrijednosti broja  $\pi$ .

U kontekstu financija, Monte Carlo simulacije koriste se za generiranje vjerojatnosnih scenarija budućih cijena. Za portfelj od  $n$  instrumenata, koraci su:

1. Generiraj nezavisne slučajne varijable  $Z_i \sim N(0, 1)$  (i.i.d.)
2. Transformiraj slučajni vektor nezavisnih varijabli  $Z$  u slučajni vektor koreliranih varijabli  $\mathbf{Y} = L\mathbf{Z}$  gdje je  $L$  donje trokutasta matrica Cholesky dekompozicije  $\Sigma$
3. Ažuriraj cijene simulacije po GBM formuli za svaki instrument:

$$S_t^{(i)} = S_0^{(i)} \exp \left( \left( \mu_i - \frac{\sigma_i^2}{2} \right) \Delta t + \sigma_i Y_i \sqrt{\Delta t} \right) \quad (2.10)$$

4. Izračunaj vrijednost portfelja  $V_t = \sum_{i=1}^n w_i S_t^{(i)}$

## 2.7. Cholesky dekompozicija

Cholesky dekompozicija je numerička metoda koja se koristi za dekompoziciju simetričnih pozitivno definitnih matrica. I upravo je matrica kovarijance  $\Sigma$  simetrična pozitivno definitna matrica. Teorem u [3].

**Definicija 8.** Neka je  $\Sigma$  simetrična pozitivno definitna matrica kovarijance. Tada postoji jedinstvena donja trokutasta matrica  $L$  takva da:

$$\Sigma = LL^\top \quad (2.11)$$

gdje je  $L$  donja trokutasta matrica.

Ova faktorizacija omogućuje generiranje vektora koreliranih slučajnih varijabli iz vektora nezavisnih slučajnih varijabli.

**Teorem 1.** Neka je  $\mathbf{Z} = (Z_1, \dots, Z_n)^\top$  vektor nezavisnih  $N(0, 1)$  varijabli. Tada vektor  $\mathbf{Y} = L\mathbf{Z}$  ima kovarijacijsku matricu  $\Sigma$ .

**Dokaz 1.**

$$Cov(\mathbf{Y}) = \mathbb{E}[L\mathbf{Z}(L\mathbf{Z})^\top] - \mathbb{E}[L\mathbf{Z}]\mathbb{E}[(L\mathbf{Z})^\top] = L\mathbb{E}[\mathbf{Z}\mathbf{Z}^\top]L^\top - 0 = L \cdot I \cdot L^\top = LL^\top = \Sigma \quad (2.12)$$

## 2.8. PCA dekompozicija

PCA (Principal Component Analysis) je tehnika koja se koristi za redukciju dimenzionalnosti podataka i identifikaciju glavnih komponenti koje objašnjavaju što veći dio ukupne varijance podataka. Ovo nam omogućuje najefikasniju diverzifikaciju portfelja i samim time smanjenje rizika [1].

**Definicija 9.** Neka je  $\Sigma$  kovarijacijska matrica slučajnog vektora  $\mathbf{X} = (X_1, X_2, \dots, X_n)^\top$ . Svojstvena dekompozicija matrice  $\Sigma$  daje:

$$\Sigma = \mathbf{V} \Lambda \mathbf{V}^\top \quad (2.13)$$

gdje je  $\mathbf{V}$  ortogonalna matrica vlastitih vektora, a  $\Lambda$  dijagonalna matrica vlastitih vrijednosti.

Ovaj rezultat sljedi iz spektralnog teorema [4]. Sada možemo definirati PCA dekompoziciju.

**Definicija 10.** *PCA dekompozicija slučajnog vektora  $\mathbf{X}$  daje novi vektor  $\mathbf{Y} = (Y_1, Y_2, \dots, Y_n)^\top$  gdje su  $Y_i$  glavne komponente slučajnog vektora  $\mathbf{X}$ .*

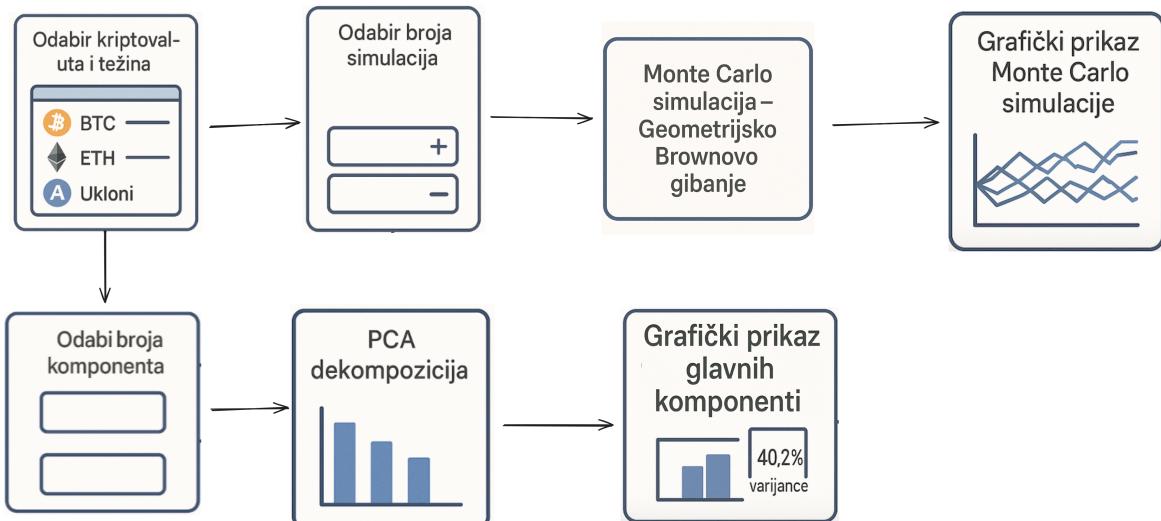
*Glavne komponente su linearne kombinacije originalnih varijabli  $\mathbf{X}$  i definiraju se kao:*

$$\mathbf{Y} = \mathbf{V}^\top \mathbf{X} \quad (2.14)$$

*gdje je  $\mathbf{V}$  matrica vlastitih vektora kovarijacijske matrice  $\Sigma$ , a  $Y_i$  su glavne komponente.*

### 3. Implementacija aplikacije

Ovaj dio rada opisuje implementaciju aplikacije za simulaciju portfelja kriptovaluta. Aplikacija je napisana u C++ programskom jeziku i koristi isključivo standardnu biblioteku C++-a za matematičke operacije i modeliranje. Za interakciju s korisnikom koristi se grafičko sučelje(implementirano sa Qt bibliotekom za C++) koje omogućuje vizualizaciju rezultata. Prikaz modela rada aplikacije je dan na slici 3.1.



Slika 3.1. Model rada aplikacije

#### 3.1. Podatci

Podatci korišteni u aplikaciji su povjesne vrijednosti kriptovaluta preuzete sa stranice [5]. Podatci su u CSV formatu i sadrže informacije o otvorenoj, zatvorenoj, najvišoj i najnižoj cijeni te volumenu kriptovalute i ukupnom tržišnom kapitalu za svaki vremenski interval. Za potrebe aplikacije, ovakav pristup je dovoljan jer se fokusiramo na analizu i implementaciju modela portfelja i potrebnih matematičkih operacija. U budućnosti bi

se moglo implementirati i dohvaćanje podataka u realnom vremenu preko nekog javno dostupnog API-ja. Time bi aplikacija postala korisna i za praćenje portfelja u realnom vremenu.

U trenutnim podatcima nalaze se povijesne vrijednosti za 23 različite kriptovalute.

- Aave (AAVE)
- Bitcoin (BTC)
- Ethereum (ETH)
- Binance Coin (BNB)
- Crypto.com Coin (CRO)
- Cardano (ADA)
- Solana (SOL)
- Ripple (XRP)
- Polkadot (DOT)
- Dogecoin (DOGE)
- Litecoin (LTC)
- Chainlink (LINK)
- Uniswap (UNI)
- Bitcoin Wrapped (WBTC)
- Stellar (XLM)
- TRON (TRX)
- Cosmos (ATOM)
- Monero (XMR)
- Thether (USDT)

- USD Coin (USDC)
- Eos (EOS)
- Iota (IOTA)
- NEM (XEM)

## 3.2. Algoritmi i komponente

Aplikacija se sastoji od nekoliko ključnih komponenti:

- **Parsiranje podataka (3.2.1.)**: učitavanje i obrada povijesnih podataka o cijenama kriptovaluta.
- **Model kriptovaluta (3.2.2.)**: Implementacija modela kriptovaluta.
- **Model portfelja (3.2.3.)**: Definiranje portfelja kriptovaluta.
- **Matematički okvir (3.2.4.)**: Implementacija matematičkih operacija potrebnih za simulaciju i modeliranje portfelja. Npr. (Množenje matrica, vektora, operacije nad vektorima i matricama te razne matrične dekompozicije).
- **Monte Carlo simulacija (3.2.5.)**: Generiranje simulacija budućih cijena portfelja kriptovaluta.
- **PCA dekompozicija (3.2.6.)**: Implementacija PCA dekompozicije za redukciju dimenzionalnosti podataka i identifikaciju glavnih komponenti.
- **Vizualizacija rezultata (3.2.7.)**: Prikaz rezultata simulacija kroz grafičko sučelje.

### 3.2.1. Parsiranje podataka

Efikasno parsiranje povijesnih podataka o cijenama kriptovaluta je prvi korak u simulaciji i stavarjanju modela portfelja. Podatci se učitavaju iz CSV datoteka koje sadrže povijesne cijene kriptovaluta. Povijesni podatci su preuzeti sa [5]. Podatci se parsiraju u *candle* strukturu koja sadrži otvorenu, zatvorenu, najvišu i najnižu cijenu te volumen kriptovalute za svaki vremenski interval. Također se pri parsiranju računaju i logaritamski povrati za

svaki vremenski interval te se izvršava poravnanje vremenskih oznaka svih kriptovaluta. U pravilu se kriptovalutama trguje 24/7, ali je problem bio što neke kriptovalute nisu postojale ili nisu imale podatke za sve dane svih ostalih kriptovaluta.

Kod 3.2.1: Struktura *candle* koja sadrži povijesne podatke o cijenama kriptovaluta

```
1 struct Candle {
2     public:
3         // OHLC
4         double open, high, low, close;
5         double volume, marketcap;
6         timestamp time;
7         double log_return;
8
9         // Constructors, getters, setters, etc.
10    };
```

### 3.2.2. Model kriptovalute

Model kriptovalute je implementiran kao klasa koja sadrži povijesne podatke o cijenama kriptovalute i metode za izračunavanje povrata, volatilnosti i driftova.

Kod 3.2.2: Model kriptovalute

```
1 class Cryptocurrency {
2     public:
3         std::string name, tick;
4         std::vector<Candle> hist_data;
5         bool metrics_calculated = false;
6
7         // Drift = sum(return_i) / n;
8         // Volatility = sqrt((1/n - 1) * sum((return_i - Drift)^2)
9         double drift, volatility;
10
11        // Constructors, getters, setters, etc.
12
13        void calculate_metrics();
```

```

14     void reevaluate_metrics();
15
16     void individual_monte_carlo(int sim_count, int
17         forecast_days);
18
19 }

```

### 3.2.3. Model portfelja

Model portfelja je implementiran kao klasa koja sadrži vektor kriptovaluta i njihove udjele u portfelju.

Kod 3.2.3: Model portfelja

```

1 class Portfolio {
2
3     private:
4
5         std::vector<std::string> _asset_names;
6         std::unordered_map<Cryptocurrency, double,
7             Cryptocurrency::Hash> _assets;
8
9
10    public:
11
12        // Covariance matrix
13
14        Doubles_Matrix aligned_log_return_matrix;
15
16        Doubles_Matrix covariance_matrix;
17
18        std::vector<double> aligned_means;
19
20        std::vector<double> aligned_volatilities;
21
22
23        timestamp aligned_start = timestamp();
24
25        timestamp aligned_end = timestamp();
26
27        timestamp aligned_stamp = timestamp();
28
29        bool aligned_returns_calculated = false;
30
31        bool covariance_matrix_calculated = false;
32
33        bool aligned_metrics_calculated = false;
34
35
36        void add_asset(const Cryptocurrency &crypto, double
37            ammount);
38
39
40        void remove_asset(const Cryptocurrency &crypto, double
41            ammount);

```

```

    ammount);

23

24     Cryptocurrency get_asset(const std::string &name) const;

25

26     std::vector<std::string> &get_asset_names();

27

28     Doubles_Matrix &aligned_log_returns(timestamp start);

29     Doubles_Matrix &calculate_covariance(timestamp start);

30     int calculate_aligned_metrics(timestamp start);

31     std::vector<Doubles_Matrix> monte_carlo(int simulations,
32
33         int steps,
34
35             timestamp start);

36     int PCA(timestamp start, int num_components,
37
38         std::vector<std::pair<double, std::vector<double>>>
39
40             &components,
41
42             double &total_variance, double &variance_explained);
43
44 };

```

### 3.2.4. Matematički okvir

Matematički okvir aplikacije implementira osnovne matematičke operacije nad vektorima i matricama, kao što su množenje matrica i vektora te razne matrične dekompozicije. Sve metode koriste samo standardnu biblioteku C++-a.

Kod 3.2.4: Metode implementirane u matematičkom okviru

```

1 template <typename T>
2 std::vector<std::vector<T>>
3 matrix_transpose(const std::vector<std::vector<T>> &matrix);
4
5 template <typename T>
6 int cholesky_decomposition(const std::vector<std::vector<T>>
7
8     &matrix,
9
10             std::vector<std::vector<T>> &L);
11
12 template <typename T> double vector_norm(const std::vector<T>
13
14     &v);

```

```

9
10 template <typename T> std::vector<T> normalized(const
11   std::vector<T> &v);
12
13 template <typename T>
14 std::vector<T> scalar_multiply(const std::vector<T> &v, T
15   scalar);
16
17 template <typename T>
18 std::vector<std::vector<T>>
19 matrix_scalar_multiply(const std::vector<std::vector<T>>
20   &matrix, T scalar);
21
22 template <typename T>
23 std::vector<T> vector_add(const std::vector<T> &v1, const
24   std::vector<T> &v2);
25
26 template <typename T>
27 std::vector<T> vector_subtract(const std::vector<T> &v1,
28                               const std::vector<T> &v2);
29
30 template <typename T>
31 std::vector<std::vector<T>>
32 matrix_multiply(const std::vector<std::vector<T>> &A,
33                  const std::vector<std::vector<T>> &B);
34
35 template <typename T>
36 std::vector<std::vector<T>>
37 matrix_subtract(const std::vector<std::vector<T>> &A,
38                  const std::vector<std::vector<T>> &B);

```

```

39
40
41
42 template <typename T>
43 int QR_decomposition(const std::vector<std::vector<T>> &matrix,
44                      std::vector<std::vector<T>> &Q,
45                      std::vector<std::vector<T>> &R);
46
47 template <typename T> int normalize(std::vector<T> &v);
48
49 template <typename T>
50 int matrix_inverse(const std::vector<std::vector<T>> &input,
51                     std::vector<std::vector<T>> &output) {
52
53 template <typename T>
54 std::vector<T> eigen_values(const std::vector<std::vector<T>>
55                           &matrix);
56
57 template <typename T>
58 int eigen_pairs(const std::vector<std::vector<T>> &matrix,
59                  std::vector<std::pair<T, std::vector<T>>>
60                  &results);

```

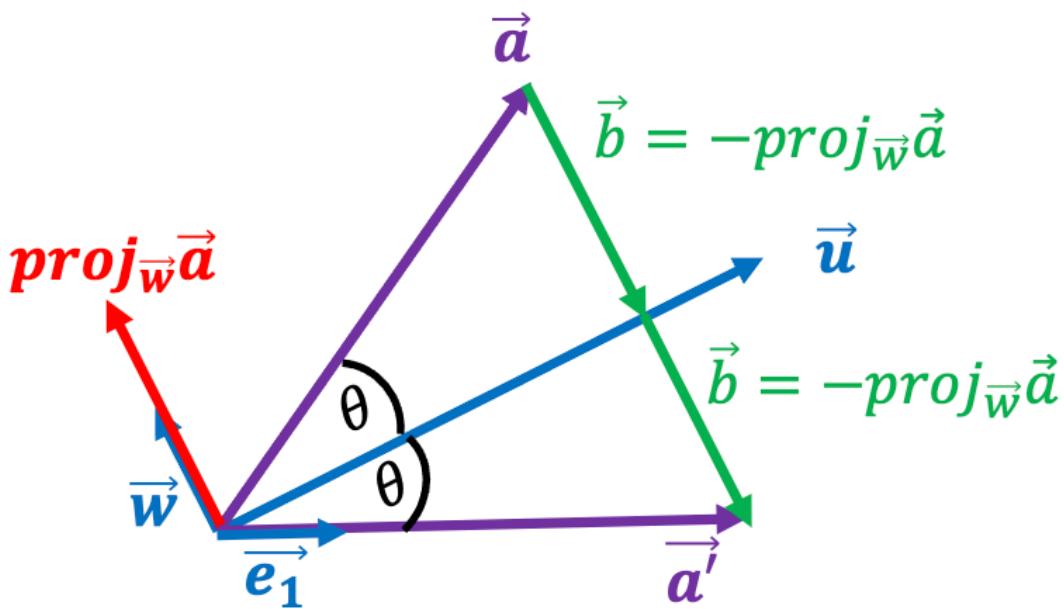
## Cholesky dekompozicija

Cholesky dekompozicija je implementirana kao metoda koja prima kovarijacijsku matricu i matricu  $L$  u koju se spremaju donja trokutasta matrica dekompozicije te metoda vraća int koji označava uspješnost dekompozicije. Algoritam radi točno prema koracima matematične notacije. Znamo da je matrica kovarijance uvijek simetrična i pozitivno definitna, pa je Cholesky dekompozicija uvijek moguća.

## QR dekompozicija

QR dekompozicija je implementirana kao metoda koja prima ulaznu matricu te matrice  $Q$  i  $R$  u koje se spremaju rezultati dekompozicije. Implementacija QR dekompozicije koristi

Householderove refleksije za dekompoziciju matrice. Ideja Householderove refleksije je pronaći refleksiju koja će transformirati prvi stupac matrice u vektor  $\begin{pmatrix} \|x\| \\ 0 \\ \vdots \\ 0 \end{pmatrix}$ . Nakon što se prvi stupac transformira, iterativno se primjenjuju Householderove refleksije na preostale stupce matrice.

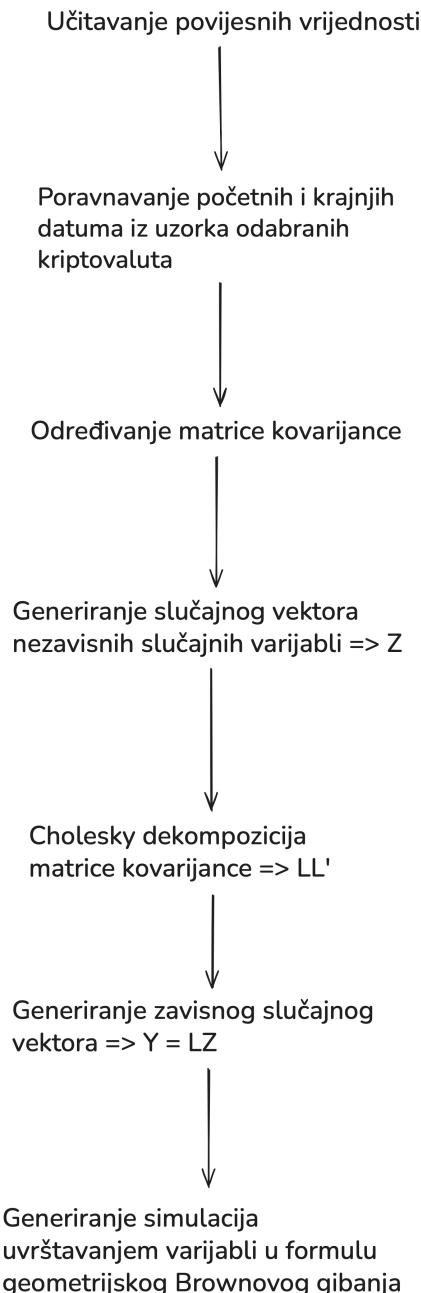


Slika 3.2. Primjer Householderove refleksije, preuzeto sa [6]

### Svojstvena dekompozicija

Svojstvena dekompozicija je implementirana kao metoda koja prima kovarijacijsku matricu te par vlastitih vrijednosti i vlastitih vektora u koje se spremaju rezultati dekompozicije. Za izračun vlastitih vrijednosti koristi se *metoda QR iteracija* [7] koja QR dekompoziciju iterativno primjenjuje na matricu i onda množi matricu  $R$  i  $Q$  te tako dobivamo novu matricu koja je približno gornje trokutasta. Zatim se vlastiti vektori dobivaju posebno za svaku vlastitu vrijednost koristeći *metodu inverznih iteracija* [7]. Ovakva implementacija nije najefikasnija i sigurno među prvim optimizacijama koje bi se mogle i trebale napraviti.

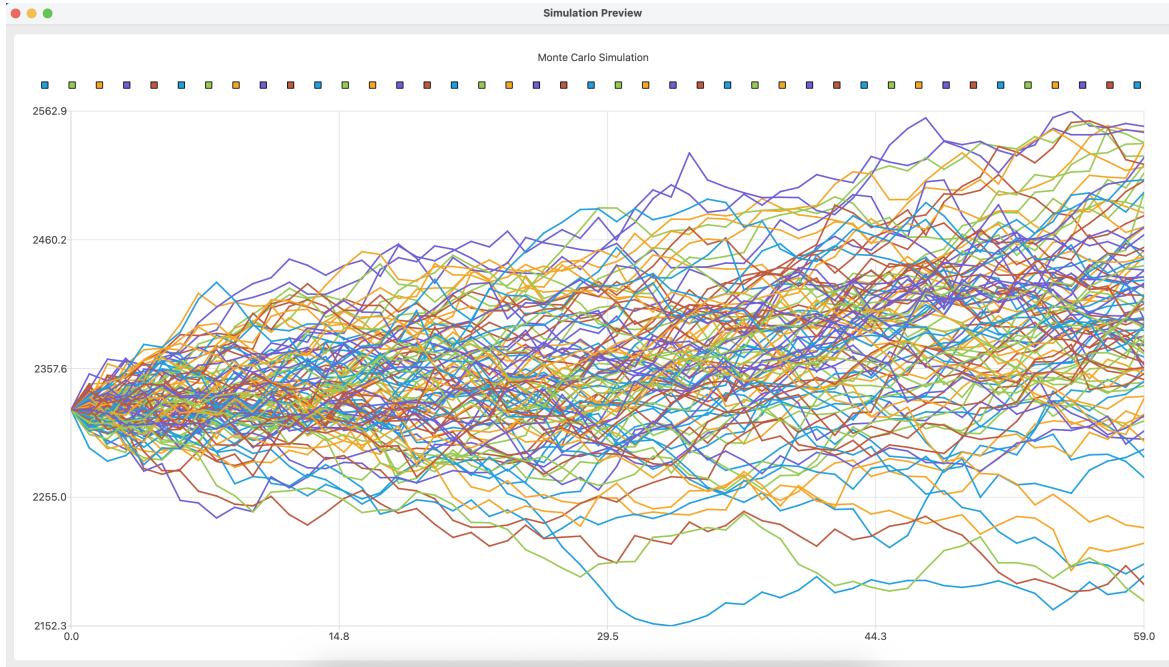
### 3.2.5. Monte Carlo simulacija



**Slika 3.3.** Monte Carlo simulacija portfelja kriptovaluta

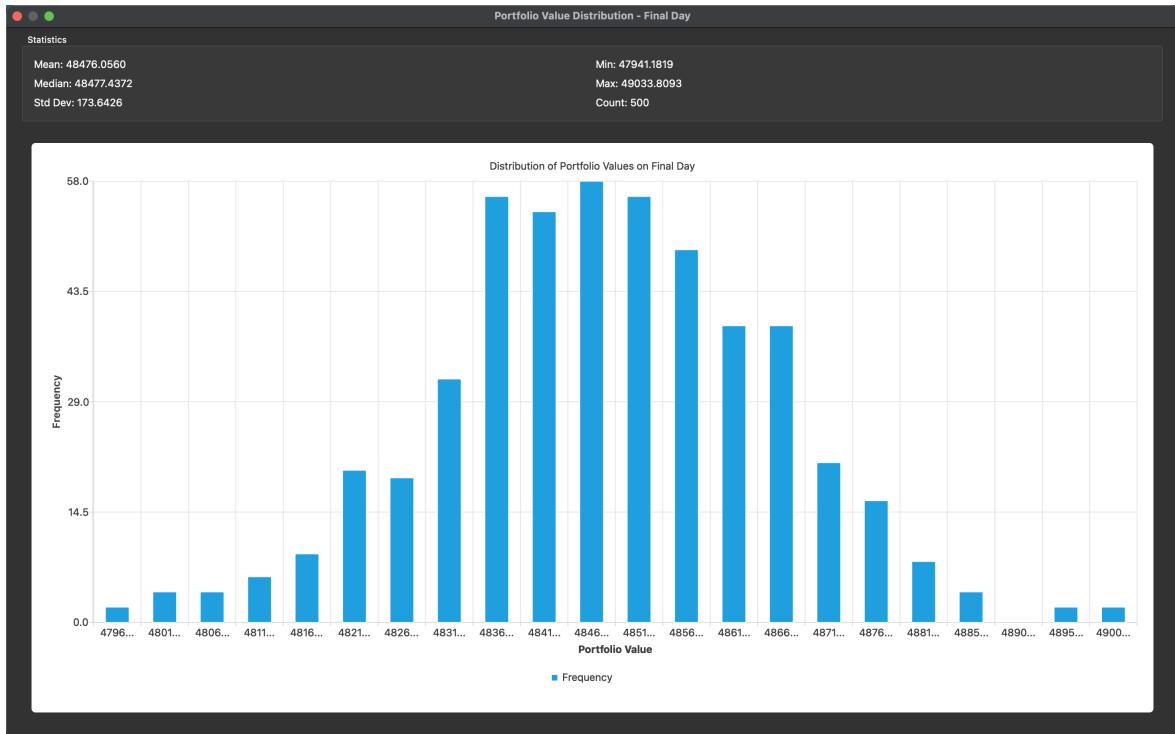
Monte Carlo simulacija je implementirana kao metoda klase `Portfolio` koja generira simulacije budućih cijena portfelja kriptovaluta. Simulacije koriste model geometrijskog Brownovog gibanja za generiranje budućih cijena te Cholesky dekompoziciju za osiguranje uzoračke korelacije između kriptovaluta. Implementacija Monte Carlo simulacija prvo koristi metodu za izračuvanje kovarijacijske matrice portfelja iz povijesnih podataka kriptovaluta. Također se može postaviti početni datum simulacije, broj simulacija i broj

koraka simulacije. Zatim se koristeći `std::normal_distribution(0,1)` generiraju nezavisne slučajne varijable( $\sim N(0,1)$ ) koje se transformiraju u korelirane slučajne varijable pomoću Cholesky dekompozicije. Zatim se ažuriraju cijene simulacije koristeći geometrijsko Brownovo gibanje prema formuli: 2.5. Primjer rezultata simulacije je dan na slici 3.4.



**Slika 3.4.** Primjer Monte Carlo simulacije portfelja kriptovaluta (100 simulacija, 60 dana)

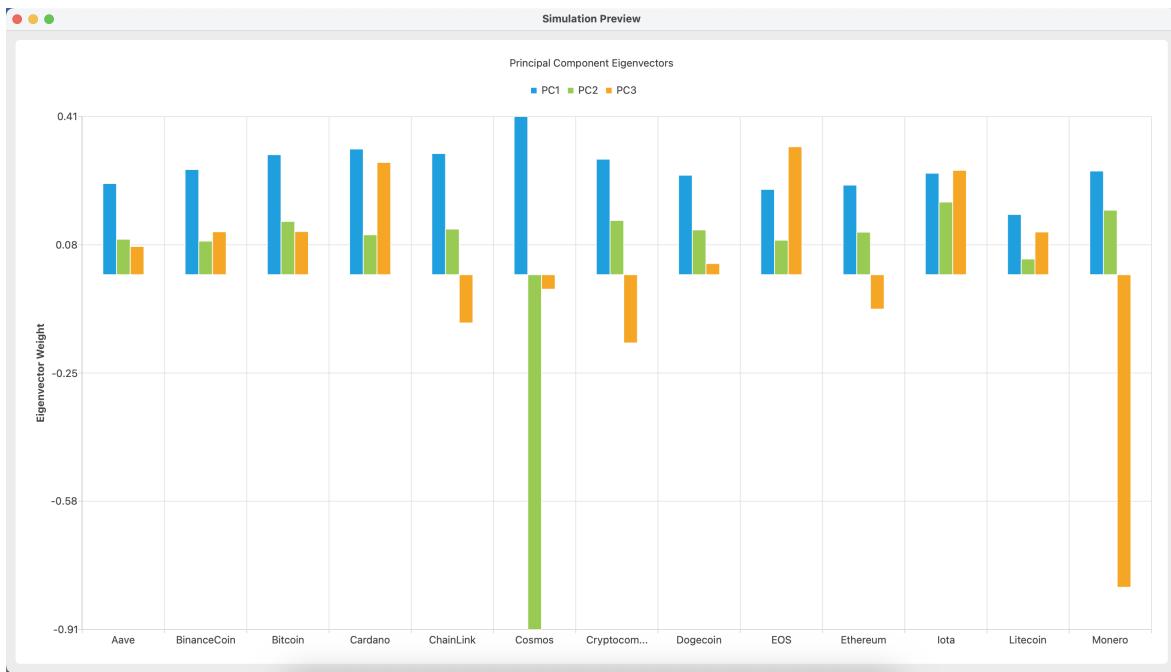
Osim prikaza same simulacije, aplikacija također implementira i mogućnost prikaza statističkih rezultata simulacije kao što su očekivana vrijednost portfelja, medijan vrijednosti portfelja, volatilnost i vizualizacija distribucije prikazane simulacije. Primjer rezultata statističke analize i distribucije simulacije dan je na slici 3.5.



**Slika 3.5.** Primjer statističke analize i distribucije simulacije portfelja kriptovaluta

### 3.2.6. PCA dekompozicija

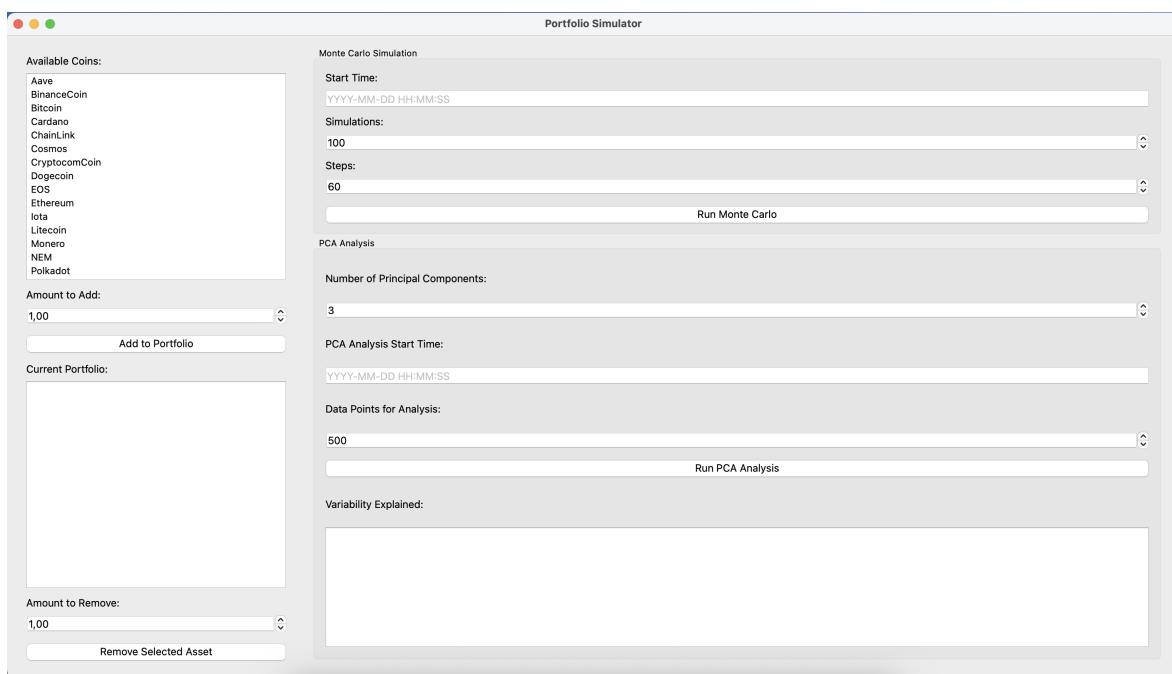
PCA dekompozicija je također implementirana kao metoda klase `Portfolio` koja prima ulaznu kovarijacijsku matricu, broj glavnih komponenti i vektor u koji se spremaju glavne komponente poredane po količini varijance koju objašnjavaju te ukupnu varijancu (u ovom slučaju to je suma svih vlastitih vrijednosti) i varijancu objašnjenu glavnim komponentama. Metoda vrati `int` koji označava uspješnost dekompozicije. Za izračun glavnih komponenti koristi se svojstvena dekompozicija koja daje vlastite vrijednosti i vlastite vektore kovarijacijske matrice te je objašnjena u 3.2.4. primjer rezultata PCA dekompozicije je dan na slici 3.6.



Slika 3.6. Primjer PCA dekompozicije portfelja kriptovaluta (3 glavne komponente)

### 3.2.7. Vizualizacija rezultata

Za vizualizaciju rezultata i interakciju s korisnikom koristi se grafičko sučelje implementirano u C++-u koristeći Qt biblioteku. Grafičko sučelje omogućuje korisniku da odabere kriptovalute koje želi staviti u portfelj, postavi udjele kriptovaluta u portfelju, odabere vremenski period simulacije i broj simulacija. Rezultati simulacija se prikazuju u obliku grafova koji se otvaraju u novom prozoru aplikacije i prikazuju teoretsko kretanje vrijednosti portfelja u budućnosti. Korisnik također može odabrati broj glavnih komponenti te izvršiti PCA dekompoziciju portfelja. Prikaz rezultata PCA dekompozicije je također u obliku grafova koji se otvara u novom prozoru aplikacije i prikazuje glavne komponente portfelja te varijancu koju objašnjavaju.



**Slika 3.7.** Grafičko sučelje aplikacije za simulaciju portfelja kriptovaluta

## 4. Rezultati i rasprava

U ovom poglavlju prikazujemo rezultate postignute implementacijom aplikacije te raspravljamo o njihovoj važnosti i primjenjivosti u stvarnom svijetu. Također, analiziramo prednosti i nedostatke implementacije te mogućnosti optimizacija i proširenja aplikacije.

### 4.1. Analiza i diskusija implementacijskih rezultata

#### 4.1.1. Funkcionalnost učitavanja i parsiranja podataka

Ručna implementacija parsiranja podataka omogućila je optimizaciju učitavanja i obrade podataka zbog specifičnosti izgleda samog skupa podataka. Osim što odmah znamo koji su stupci prisutni dodatna optimizacija je postignuta pregledavanjem veličine datoteke i aproksimacijom broja redaka u datoteci. To nam omogućuje da unaprijed rezerviramo memoriju za vektor *candle* struktura što značajno ubrzava proces parsiranja pogotovo kod većih skupova podataka jer ne moramo konstantno realocirati memoriju pri proširivanju vektora. Ova implementacija je na dostupnim skupovima podataka postigla brzinu parsiranja usporedivu s popularnom bibliotekom *csv2* [8] koja je optimizirana za parsiranje CSV datoteka.

#### 4.1.2. Matematički okvir

Matematički okvir aplikacije implementiran je tako da se koristi samo standardna biblioteka C++-a. Iako je ovakav pristup produljio vrijeme izrade aplikacije te samo izvođenje aplikacije nije optimizirano kao što bi bilo da se koriste biblioteke poput *Eigen* [9] ili *Armadillo* [10], koje koriste optimizirane algoritme i strukture podataka za matematičke operacije te napredne optimizacije poput SIMD instrukcija, template metaprogramming-a i expression templates-a, ovakav pristup je omogućio dublje razumijevanje i primjenu stečenih znanja.

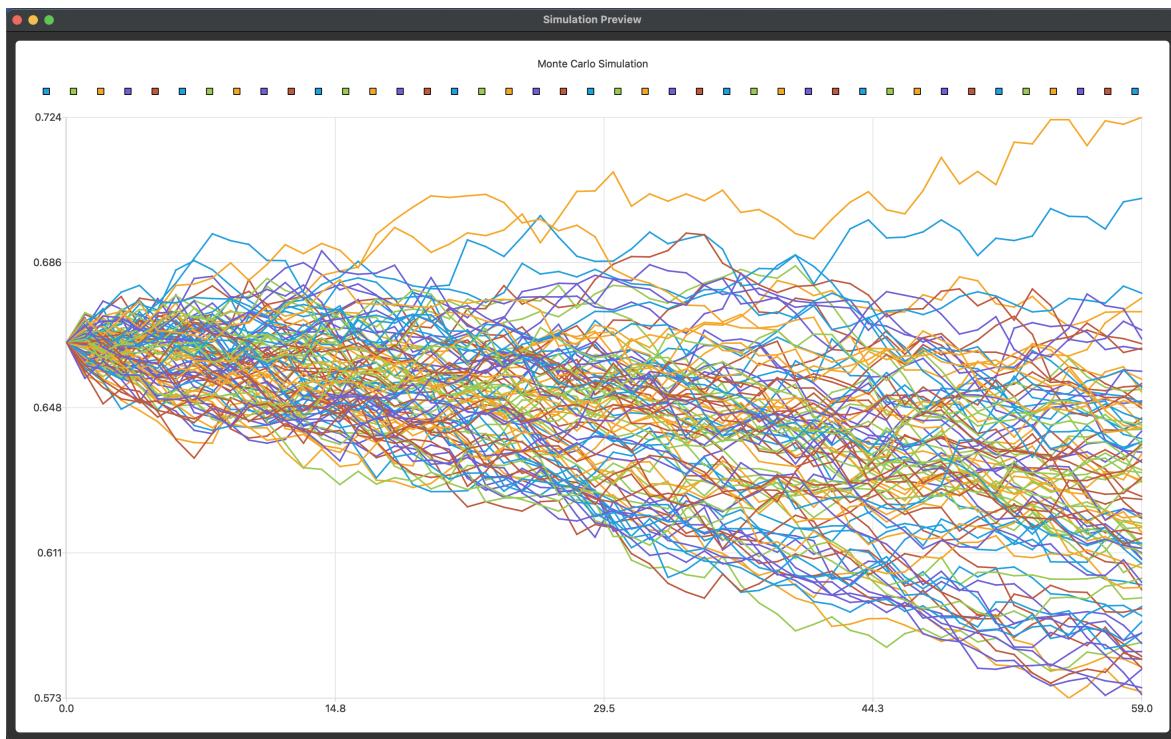
Implementacija se specifično oslanja na `std::vector` kao osnovnu strukturu podataka za vektore i matrice. `std::vector` je dinamički niz koji omogućuje efikasno upravljanje memorijom i automatsko proširivanje kada je potrebno.

Većina matematičkih operacija je implementirana na *naivni* način kako bi se osiguralo da su svi koraci jasni i razumljivi. Jedine optimizacije su napravljene koristeći *move semantics* i kako bi se izbjeglo nepotrebno kopiranje podataka te korištenje OpenMP-a [11] za paralelizaciju određenih operacija. Sve operacije su testirane na prikupljenim skupovima podataka i vrijeme izvođenja je zadovoljavajuće za većinu operacija. Jedino što je potrebno optimizirati je traženje vlastitih vektora koristeći neki od asimptotski efikasnijih algoritama u odnosu na *metodu inverznih iteracija*.

#### 4.1.3. Monte Carlo simulacija

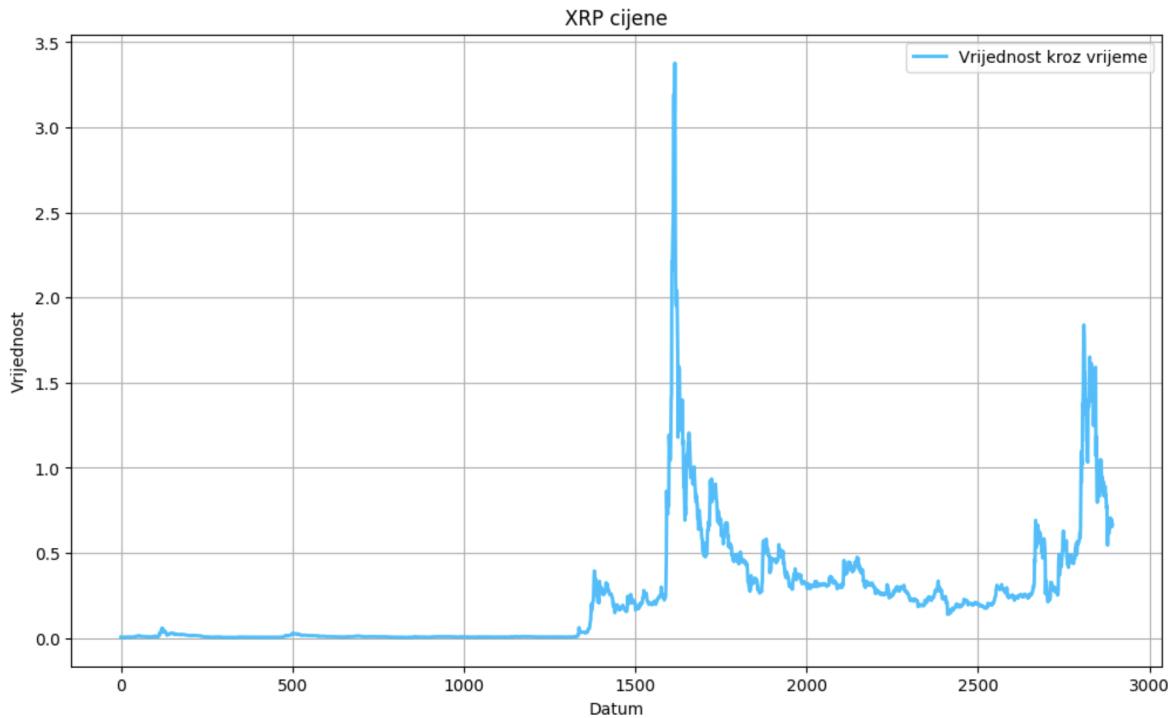
Monte Carlo simulacija se pokazala kao efikasan način generiranja vjerojatnosnih scena-rija budućih cijena portfelja kriptovaluta ukoliko je inicijalni uzorak smislen i dovoljno velik. Treba napomenuti kako bi Monte Carlo metoda modelirana sa geometrijskim Brownovim gibanjem (GBM) trebala biti korištena na kratkim vremenskim intervalima jer ipak uzima malo mogućih značajki iz povijesnih podataka i dalje se bazira na simuliranju stohastičkog procesa. Za duže vremenske periode se preporučuju regresijske metode koje prate trendove i sezonalnost. Jedan primjer simulacije koji izgleda realistično i smisleno dan je ranije na slici 3.4. Na tom se primjeru vidi blagi pozitivni *drift* cijena portfelja te volatilnost koja je u skladu s očekivanjima za portfelj kriptovaluta.

Drugi primjer simulacije je dan na slici 4.1.



**Slika 4.1.** Primjer Monte Carlo simulacije portfelja (XRP) kriptovaluta (100 simulacija, 60 dana, drift = 0.00154516, volatilnost = 0.0727809)

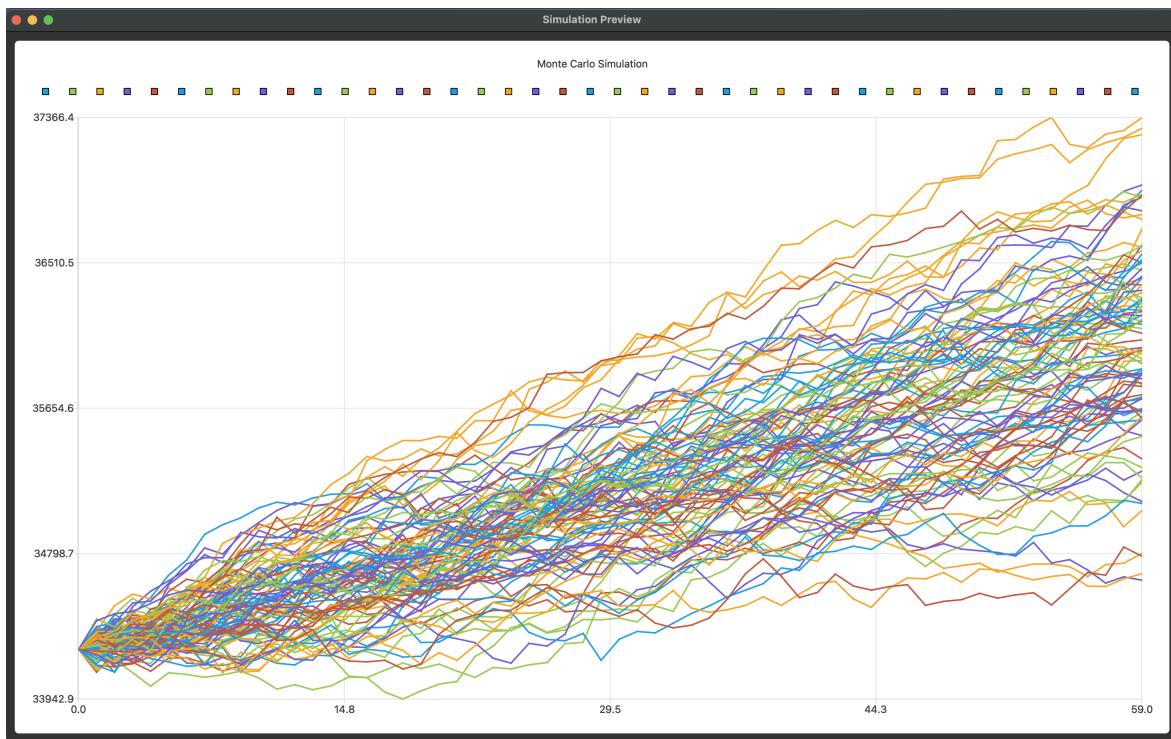
Ovaj primjer simulacije pokazuje da je *drift* povrata portfelja izrazito nisko pozitivan, ali da je volatilnost portfelja toliko visoka da model predviđa budući pad. Ovakav izgled simulacije je očekivan ako uzmemo u obzir da koristimo statističku metodu koja gleda samo povijesne podatke i ne uzima u obzir puno drugih faktora koji utječu na cijene kriptovaluta. Izgled kretanja vrijednosti portfelja u povijesnim podatcima dan je na slici 4.2. Na slici se vidi da je volatilnost cijene kriptovalute XRP visoka te je vrijednost iste u zadnjih nekoliko godina većinom opadala uz 2 velika i nagla porasta.



**Slika 4.2.** Povijesno kretanje cijene kriptovalute XRP

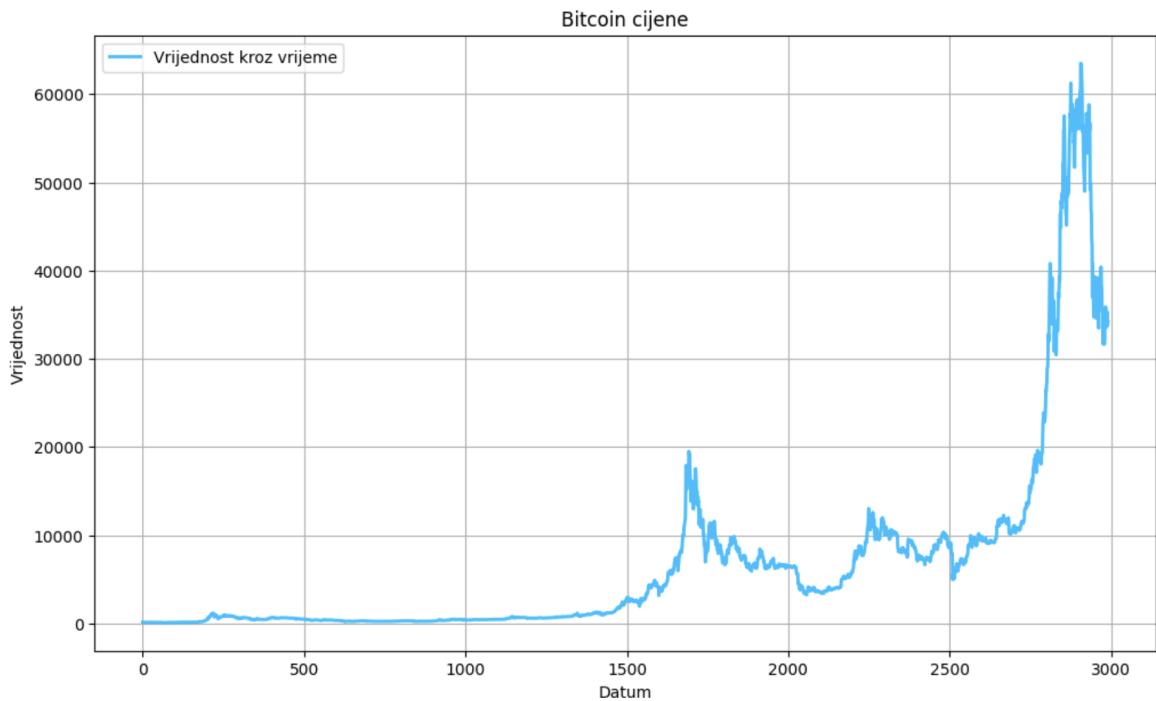
Treći primjer simulacije je dan na slici 4.3. Na slici se vidi da je *drift* povrata portfelja pozitivan, ali da je volatilnost nešto niža nego u prethodnom primjeru te model predviđa stabilniji budući rast.

U ovom primjeru uočavamo problem kod modela GMB te Monte Carlo simulacija za kriptovalute za dulje vremenske periode. Budući da se radi o finacijskim instrumentima koji su relativno novi i malo je povijesnih podataka, Monte Carlo simulacije koje su statistička metoda i temelje se na povijesnim podacima i modeliranju stohastičkog procesa, ne daju uvijek realistične rezultate. Problem je u zanemarivanju dodatnih informacija o tržištu koje bi se mogle modelirati korištenjem regresijskih metoda. Iz ove bi se simulacije moglo zaključiti da očekujemo rast vrijednosti od nekoliko posto u sljedećih 60 dana, ali to nije nužno točno (iako je vrijednost *Bitcoin*-a stvarno značajno porasla u proteklom vremenu).



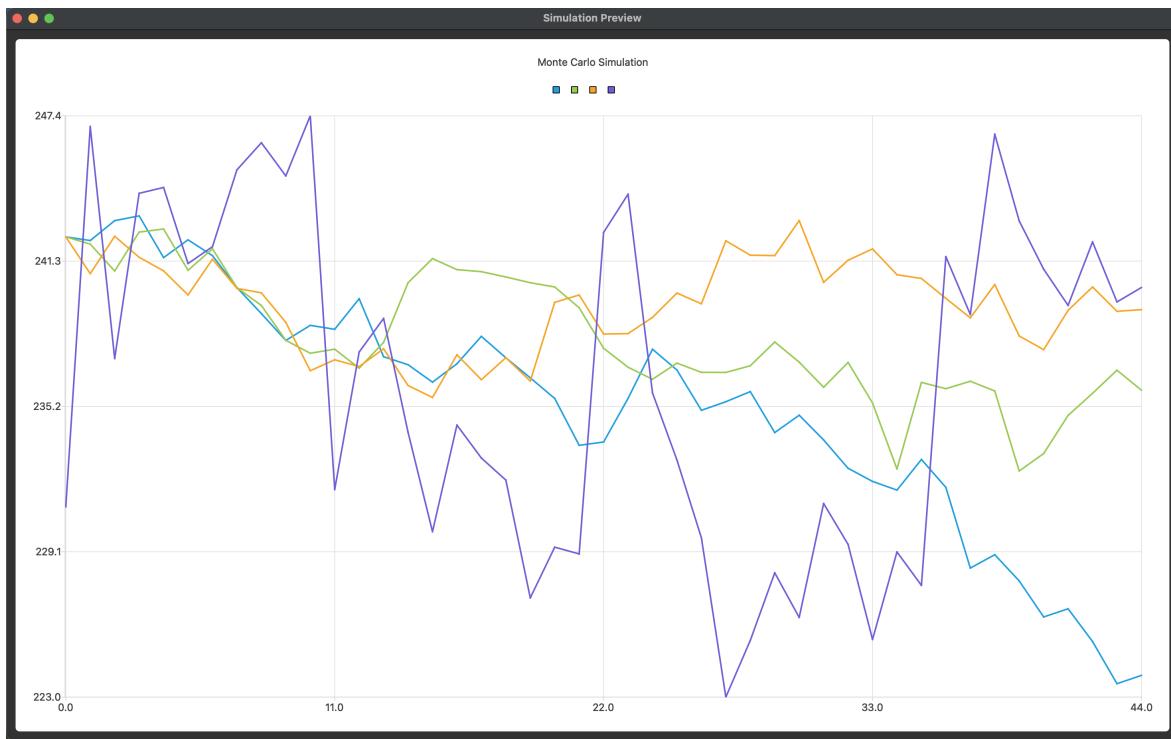
**Slika 4.3.** Primjer Monte Carlo simulacije portfelja (Bitcoin) kriptovaluta (100 simulacija, 60 dana, drift = 0.0017602, volatilnost = 0.0426928)

Promatajući povijesne podatke *Bitcoin*-a, dane na slici 4.4., vidimo da je volatilnost cijene *Bitcoin*-a također visoka, ali da je zapravo cijena *Bitcoin*-a u zadnjih nekoliko godina većinom snažno rasla uz nekoliko manjih padova. Ovakav izgled povijesnih podataka zapravo potpuno opravdava pozitivni *drift* i sam izgled simulacije u 4.3.



**Slika 4.4.** Povijesno kretanje cijene kriptovalute Bitcoin

Primjer testiranja Monte Carlo simulacije je dan na slici 4.5. Na slici se jasno vidi kako model može simulirati generalno kretanje vrijednosti portfelja te intervale u kojima očekuje volatilnost, ali je problem odrediti točan smjer kretanja volatilnosti. Također vidimo da što je manji interval predviđanja to je simulacija točnija.



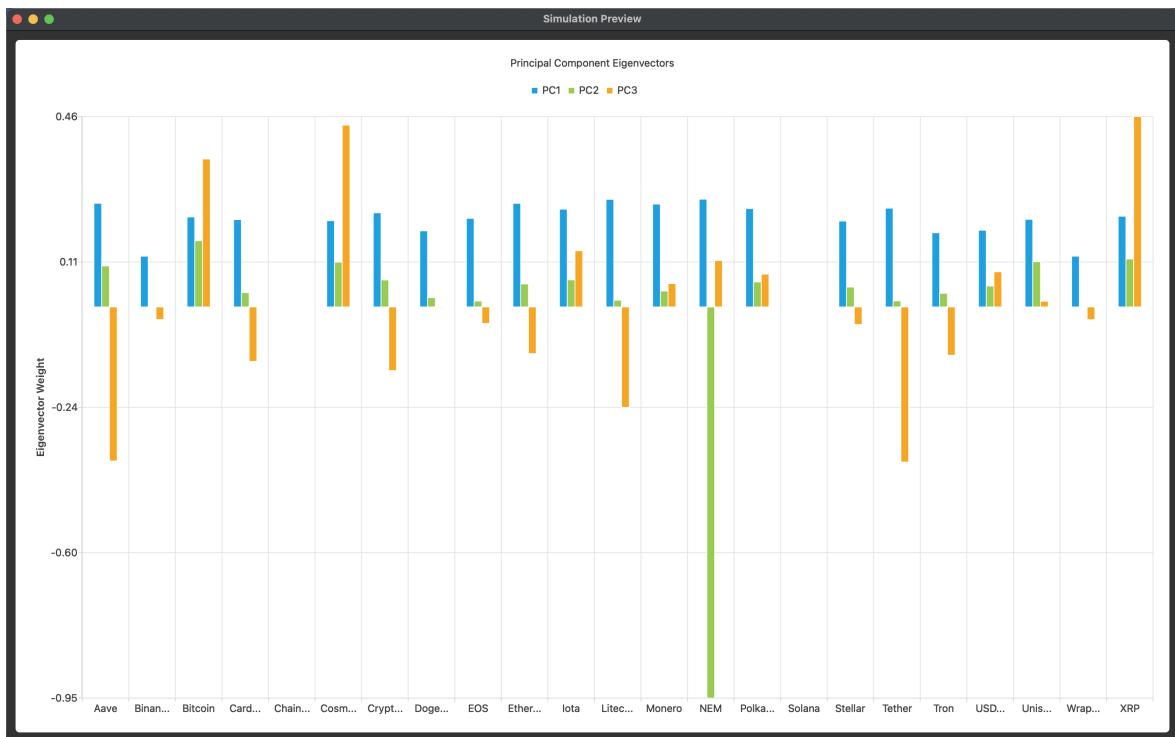
**Slika 4.5.** Test Monte Carlo simulacije portfelja kriptovaluta (3 simulacije, 45 dana)  
 Ljubičasti portfelj - stvarne cijene kriptovalute  
 Plavi, zeleni i žuti portfelj - simulirane cijene kriptovalute

#### 4.1.4. PCA dekompozicija

Implementacija PCA dekompozicije koristi implementirane metode za izračunavanje uzoračke kovarijacijske matrice te nakon toga poziva metoda za određivanje svojstvenih vrijednosti i vektora. To zapravo znači da je većina vremena izvođenja ovisna o implementaciji svojstvene dekompozicije jer je asimptotski najsporija od svih operacija. Kao što je ranije spomenuto, implementacija koristi *metodu QR iteracija* za izračun svojstvenih vrijednosti i *metodu inverznih iteracija* za izračun svojstvenih vektora. Ovakva implementacija je dovoljno brza za manje skupove podataka i za potrebe ove aplikacije, ali već za matrice veličine  $30 \times 30$  i više vrijeme izvođenja je par sekundi što je neprihvatljivo za aplikaciju koja bi trebala raditi u realnom vremenu. Ovakva bi se implementacija trebala optimizirati korištenjem pametnijih algoritama za izračun svojstvenih vrijednosti i vektora te optimizacijama na razini C++ koda, kao što su *SIMD instrukcije* i *Cache optimizacije*.

Primjer rezultata PCA dekompozicije je dan na slici 3.6. Iz tog i sljedećeg primjera danog na slici 4.6. se vidi da su kriptovalute jako međusobno korelirane i može se primijetiti da

je prva glavna komponenta zapravo svojstveni portfelj približno jednakih težina.



**Slika 4.6.** Primjer PCA dekompozicije portfelja kriptovaluta (3 glavne komponente)

#### 4.1.5. Vizualizacija rezultata

Grafičko sučelje aplikacije omogućuje korisnicima jednostavno upravljanje portfeljem, pokretanje simulacija i vizualizaciju rezultata. Vizualizacija rezultata simulacija i PCA dekompozicije omogućuje korisnicima bolje razumijevanje rizika i mogućnosti portfelja kriptovaluta. Grafovi su interaktivni i omogućuju korisnicima da pregledavaju rezultate simulacija i PCA dekompozicije na različite načine. Implementacija grafičkog sučelja je napravljena korištenjem Qt biblioteke koja omogućuje jednostavno kreiranje grafičkog sučelja i apstrakciju od niskorazinskih detalja upravljanja prozorima i događajima. Iako je ovakvo rješenje dovoljno dobro za potrebe ove aplikacije, u budućnosti bi se mogla razmotriti implementacija ručnog rješenja za grafičko sučelje kako bi se dobila veća fleksibilnost i kontrola nad izgledom i ponašanjem sučelja. Također, mogla bi se razmotriti izrada boljeg rješenja za vizualizaciju rezultata simulacija i PCA dekompozicije kao što su 3D grafovi ili animacije koje bi omogućile bolje razumijevanje rizika i mogućnosti portfelja kriptovaluta.

Navedena poboljšanja bi omogućila bolji dizajn aplikacije te unapređenje korisničkog

iskustva u vidu *UX* i *UI* dizajna.

## 5. Upute za korištenje i pokretanje aplikacije

Aplikacija se može pokrenuti na bilo kojem sustavu koji podržava C++ i Qt biblioteku. Potrebno je instalirati Qt biblioteku na lokalno računalo te imati dostupan Clang kompilator. Nakon toga se može ili ručno kompajlirati aplikacija koristeći dani CMakeLists.txt unutar direktorija App ili se može pokrenuti build\_script.sh skripta koja je također unutar direktorija App i automatski stvara izvršnu datoteku aplikacije. Skripta je napisana za Unix okruženje. Nakon kompajliranja aplikacije unutar direktorija App/build će se stvoriti izvršna datoteka PortfolioSimulator koja se može pokrenuti direktno iz terminala. Aplikacija će se pokrenuti u grafičkom sučelju i korisnik dalje može interaktivno koristiti aplikaciju. Uz samu aplikaciju stvara se i dodatna datoteka run\_tests koja služi za provođenje testova aplikacije. Testovi su napisani opet u C++-u i koriste samo standardnu biblioteku C++-a. Napisani su tako da pokrivaju sve ključne komponente aplikacije i provjeravaju ispravnost implementacije. Sve naredbe i primjeri za pokretanje aplikacije i testova dani su u nastavku.

Kod 5.0.1: Primjeri naredbi za pokretanje aplikacije i testova

```
1 # Kloniranje repozitorija
2 git clone https://github.com/IvanDzanija/PortfolioSimulator.git
3 # Pozicioniranje u direktorij aplikacije
4 cd PortfolioSimulator/App
5 # Pokretanje skripte za kompajliranje aplikacije
6 ./build_script.sh
7 # Pokretanje aplikacije
8 ./build/PortfolioSimulator
9 # Pokretanje testova
10 ./build/run_tests
```

## 6. Zaključak

U ovom radu prikazan je razvoj aplikacije za modeliranje investicijskih portfelja kriptovaluta korištenjem Monte Carlo simulacija i PCA dekompozicije. Kroz teorijski pregled objašnjeni su ključni financijski i matematički koncepti poput teorije portfelja, modela geometrijskog Brownovog gibanja, kovarijacijske matrice, Cholesky i PCA dekompozicije, čime je postavljen temelj za razumijevanje i implementaciju modela. Implementacija je realizirana u C++ programskom jeziku, uz korištenje standardne biblioteke i vlastitih rješenja za matematičke operacije, što je omogućilo detaljno razumijevanje svakog koraka i fleksibilnost u razvoju. Rezultati pokazuju da Monte Carlo simulacije omogućuju generiranje vjerojatnosnih scenarija budućih vrijednosti portfelja, ali i ukazuju na ograničenja povezana s visokom volatilnošću i kratkom povješću kriptovaluta. PCA dekompozicija dodatno je omogućila redukciju dimenzionalnosti i identifikaciju glavnih izvora rizika u portfelju, pri čemu su rezultati potvrđili visoku međusobnu koreliranost kriptovaluta. Analiza funkcionalnosti aplikacije pokazala je da je ručna implementacija parsiranja i matematičkog okvira bila dovoljno brza i pouzdana za manje skupove podataka, ali bi za veće skupove i rad u realnom vremenu bilo potrebno dodatno optimizirati algoritme, osobito algoritam za izračun svojstvenih vrijednosti i vektora. Aplikacija, uz grafičko sučelje, omogućuje korisnicima jednostavno upravljanje portfeljem, pokretanje simulacija i vizualizaciju rezultata, što može pomoći u boljem razumijevanju i upravljanju rizicima povezanimi s kriptovalutama. Ograničenja modela proizlaze iz oslanjanja na povijesne podatke i prepostavke o distribuciji i parametrima procesa, zbog čega rezultati simulacija ne mogu u potpunosti predvidjeti buduće tržišne događaje.

Tijek daljnog razvoja aplikacije bi uključivao integraciju naprednijih statističkih modela koji nisu toliko ovisni o početnoj prepostavci distribucije i parametara, proširenje podrške za *real-time* podatke te optimizacija postojećih algoritama korištenjem specija-

liziranih matematičkih algoritama i optimizacijom prisupa *Cache* memoriji. Razvijeni programski okvir i biblioteka za rad s financijskim podacima mogu poslužiti kao temelj za buduća istraživanja i razvoj naprednijih alata u području financijske analitike i upravljanja rizikom te svojom modularnošću pružiti dodatni edukacijski resurs.

## Literatura

- [1] Z. Bodie, A. Kane, i A. J. Marcus, *Investments*. McGraw-Hill, 2014.
- [2] J. Campana i C. Grenon, “Deriving the black-scholes formula from brownian motion”, 2021.
- [3] J. P. Milišić i A. Žgaljić Keko, *Uvod u numeričku matematiku za inženjere*. Element, 2014.
- [4] A. A. Aljinović, N. Elezović, i D. Žubrinić, *Linearna algebra*. Element, 2017.
- [5] S. Rajkumar, “Cryptocurrency historical prices”, <https://www.kaggle.com/datasets/sudalairajkumar/cryptocurrencypricehistory/data>.
- [6] A. Kwok, “Detailed explanation of qr decomposition by householder transformation”, <https://kwokanthony.medium.com/detailed-explanation-with-example-on-qr-decomposition-by-householder-transformation-5e964d7f7656>.
- [7] N. Truhar, *Numerička linearna algebra*. Odjel za matematiku, Svučilište J. J. Strossmayera u Osijeku, 2010.
- [8] “Csv2”, <https://github.com/p-ranav/csv2>.
- [9] G. Guennebaud, B. Jacob *et al.*, “Eigen v3”, <http://eigen.tuxfamily.org>, 2010.
- [10] C. Sanderson i R. Curtin, “Armadillo: an efficient framework for numerical linear algebra”, u *2025 17th International Conference on Computer and Automation Engineering (ICCAE)*, 2025., str. 303–307. <https://doi.org/10.1109/ICCAE64891.2025.10980539>

- [11] L. Dagum i R. Menon, “Openmp: an industry standard api for shared-memory programming”, *IEEE Computational Science and Engineering*, sv. 5, br. 1, str. 46–55, 1998. <https://doi.org/10.1109/99.660313>

## **Sažetak**

### **Razvoj aplikacije za procjenu rizika u investicijskim portfeljima uz pomoć Monte Carlo simulacija**

Ivan Džanija

“PortfolioSimulator“ je aplikacija za modeliranje investicijskih portfelja kriptovaluta korištenjem Monte Carlo simulacija i PCA dekompozicije. Aplikacija je implementirana u C++ programskom jeziku, koristeći standardnu biblioteku STL i vlastita rješenja za matematičke, matrične i vektorske operacije te Qt biblioteku za grafičko sučelje. Povijesni podaci o cijenama 23 kriptovalute preuzeti su s javno dostupnog izvora, a korisniku je omogućeno upravljanje portfeljem i simulacijama. U uvodnom dijelu objašnjeni su ključni financijski i matematički koncepti: teorija portfelja, geometrijsko Brownovo gibanje, kovarijacijska matrica, Cholesky i PCA dekompozicija. Analiza rezultata pokazuje da su kriptovalute visoko međusobno korelirane i volatilne, što ograničava preciznost modela temeljenih isključivo na povijesnim podacima. U sljedećem poglavlju detaljno je opisana implementacija svega potrebnog za rad aplikacije, uključujući parsiranje podataka, matematički okvir, modele kriptovaluta i portfelja te Monte Carlo simulacije i PCA dekompoziciju. Na kraju je dan pregled postignutih rezultata, rasprava o funkcionalnosti aplikacije i upute za korištenje i pokretanje.

**Ključne riječi:** PortfolioSimulator; Monte Carlo simulacije; PCA dekompozicija; kriptovalute; C++; Qt; STL; stohastički modeli; financijska analitika;

# **Abstract**

## **Development of an application for risk assessment in investment portfolios with the help of Monte Carlo simulations**

Ivan Džanija

"PortfolioSimulator" is an application for modeling investment portfolios of cryptocurrencies using Monte Carlo simulations and PCA decomposition. The application is implemented in the C++ programming language, utilizing the standard library STL and custom solutions for mathematical, matrix, and vector operations, as well as the Qt library for the graphical interface. Historical price data for 23 cryptocurrencies were obtained from a publicly available source, and the user is able to manage the portfolio and simulations. The introductory section explains key financial and mathematical concepts: portfolio theory, geometric Brownian motion, covariance matrix, Cholesky and PCA decomposition. The analysis of the results shows that cryptocurrencies are highly intercorrelated and volatile, which limits the accuracy of models based solely on historical data. The following chapter describes in detail the implementation of everything needed for the application to work, including data parsing, mathematical framework, cryptocurrency and portfolio models, Monte Carlo simulations, and PCA decomposition. Finally, an overview of the achieved results, a discussion of the application's functionality, and instructions for use and execution are provided.

**Keywords:** PortfolioSimulator; Monte Carlo simulation; PCA decomposition; cryptocurrency; C++; Qt; STL; stochastic models; financial analytics;