Računanje s Vandermondeovom matricom i determinantom u programskom jeziku C#

Franković, Patrik

Undergraduate thesis / Završni rad

2023

Degree Grantor / Ustanova koja je dodijelila akademski / stručni stupanj: **Josip Juraj Strossmayer University of Osijek, Faculty of Electrical Engineering, Computer Science and Information Technology Osijek / Sveučilište Josipa Jurja Strossmayera u Osijeku, Fakultet elektrotehnike, računarstva i informacijskih tehnologija Osijek**

Permanent link / Trajna poveznica: https://urn.nsk.hr/urn:nbn:hr:200:751320

Rights / Prava: In copyright / Zaštićeno autorskim pravom.

Download date / Datum preuzimanja: **2024-05-10**

Repository / Repozitorij:

Faculty of Electrical Engineering, Computer Science and Information Technology Osijek

SVEUČILIŠTE JOSIPA JURJA STROSSMAYERA U OSIJEKU
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA OSIJEK

Sveučilišni studij

RAČUNANJE S VANDERMONDEOVOM MATRICOM I DETERMINANTROM U PROGRAMSKOM JEZIKU C#

Završni rad

Patrik Franković

Osijek, 2023.

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

Obrazac Z1P - Obrazac za ocjenu završnog rada na preddiplomskom sveučilišnom studiju

Osijek, 21.09.2023.

Odboru za završne i diplomske ispite

Prijedlog ocjene završnog rada na preddiplomskom sveučilišnom studiju

Ime i prezime Pristupnika:	Patrik Franković
Studij, smjer:	Sveučilišni prijediplomski studij Elektrotehnika i informacijska
Mat. br. Pristupnika, godina upisa:	4819, 29.07.2020.
OIB Pristupnika:	24940345314
Mentor:	doc. dr. sc. Tomislav Rudec
Sumentor:	izv. prof. dr. sc. Alfonzo Baumgartner
Sumentor iz tvrtke:	
Naslov završnog rada:	Računanje s Vandermondeovom matricom i determinantom u programskom jeziku C#
Znanstvena grana rada:	**Programsko inženjerstvo (zn. polje računarstvo)**
Zadatak završnog rad:	Student će opisati matematičke algoritme za računanje s Vandermondeovim matricama i determinantama i onda izraditi aplikaciju u programskom jeziku C#. Tema je zauzeta (Patrik Franković). Sumentor s FERIT-a: Alfonzo Baumgartner.
Prijedlog ocjene završnog rada:	Dovoljan (2)
Kratko obrazloženje ocjene prema Kriterijima za ocjenjivanje završnih i diplomskih radova:	Primjena znanja stečenih na fakultetu: 2 bod/boda Postignuti rezultati u odnosu na složenost zadatka: 1 bod/boda Jasnoća pismenog izražavanja: 2 bod/boda Razina samostalnosti: 1 razina
Datum prijedloga ocjene od strane mentora:	21.09.2023.
Datum potvrde ocjene od strane Odbora:	25.09.2023.

Potvrda mentora o predaji konačne verzije rada:	Mentor elektronički potpisao predaju konačne verzije.
	Datum:

FERIT
FAKULTET ELEKTROTEHNIKE, RAČUNARSTVA I INFORMACIJSKIH TEHNOLOGIJA **OSIJEK**

IZJAVA O ORIGINALNOSTI RADA

Osijek, 25.09.2023.

Ime i prezime studenta:	Patrik Franković
Studij:	Sveučilišni prijediplomski studij Elektrotehnika i informacijska tehnologija
Mat. br. studenta, godina upisa:	4819, 29.07.2020.
Turnitin podudaranje [%]:	8

Ovom izjavom izjavljujem da je rad pod nazivom**: Računanje s Vandermondeovom matricom i determinantom u programskom jeziku C#**

izrađen pod vodstvom mentora doc. dr. sc. Tomislav Rudec

i sumentora izv. prof. dr. sc. Alfonzo Baumgartner

moj vlastiti rad i prema mom najboljem znanju ne sadrži prethodno objavljene ili neobjavljene pisane materijale drugih osoba, osim onih koji su izričito priznati navođenjem literature i drugih izvora informacija.
Izjavljujem da je intelektualni sadržaj navedenog rada proizvod mog vlastitog rada, osim u onom dijelu za koji mi je bila potrebna pomoć mentora, sumentora i drugih osoba, a što je izričito navedeno u radu.

Potpis studenta:

SADRŽAJ

1. UVOD 1

1.1. Zadatak završnog rada 2

2. PREGLED PODRUČJA TEME 3

3. TEORIJSKA PODLOGA ZADATKA 5

3.1. Vandermondeova matrica 5

3.1.1. Vandermondeova determinanta 5

3.2. Polinomijalna interpolacija 6

3.2.1. Interpolacija s Vandermondeovom matricom 6

4. KORIŠTENE TEHNOLOGIJE I OKRUŽENJA 8

4.1. Programski jezik C# 8

4.2. UWP aplikacija 8

4.3. Microsoft Visual Studio 9

5. RAZVOJ APLIKACIJE 10

5.1. Klasa MatrixCalc.cs 10

5.1.1. Metoda *VandermondeDeterminant* 11

5.1.2. Metoda *VandermondeMatrix* 12

5.1.3. Metoda *MatrixProduct* 13

5.1.4. Metoda *MatrixInverse* 14

5.2. Korisničko sučelje aplikacije 16

5.3. Testiranje i analiza rezultata 21

6. ZAKLJUČAK 25

LITERATURA 26

SAŽETAK 27

ABSTRACT 28

1. UVOD

Vandermondeova matrica i determinanta su ključni matematički pojmovi koji igraju značajnu ulogu u mnogim znanstvenim, inženjerskim i računalnim disciplinama. Fokusirat ćemo se na dubinsku analizu implementacije ovih matematičkih pojmova unutar programskog jezika C# i kako ih koristiti za rješavanje specifičnih problema i izazova u znanstvenom istraživanju. Koristi računanja s Vandermondeovom matricom i determinantom najviše dolaze do izražaja pri računanju polinomijalne interpolacije za određeni broj različitih točaka. Polinomijalna interpolacija predstavlja proces pronalaska polinoma koji prolazi kroz set koordinata. Inverz Vandermondeove matrice olakšava taj procesa, pa se zato Vandermondeova matrica često koristi u polinomijalnoj interpolaciji.

Cilj ovog završnog rada je istražiti načine i prednosti primjene Vandermondeove matrice i determinante u programskom jeziku C#. Ovaj završni rad sastoji se od pet dijelova. Prvi dio je pregled područja teme, drugi i treći pristavljaju teoriju i korišteni alat za izradu aplikacije, četvrti dio je razvoj aplikacije s testiranjem i na kraju je zaključak.

Na samom početku rada spomenut ćemo i usporediti projekte, aplikacije ili stranice s istim ili sličnim funkcija kao s onim postignutim u aplikaciji ovoga rada. Sljedeće poglavlje će spomenuti povijest Vandermondeove matrice i determinante te objasniti teoriju i primjenu tih matematičkih pojmova.

Iduće poglavlje objasniti će teoriju i funkciju polinomijalne interpolacije i opisati kako koristi računanja s Vandermondeovom matricom i determinantom najviše dolaze do izražaja pri računanju polinomijalne interpolacije za određeni broj različitih točaka.

Nakon što se pojasni matematička teorija zadatka opisuje se tehnologije koje se koriste za izradu aplikacije, točnije ukratko će se opisati razvojno okruženje *Microsoft Visual Studio*, platforma *Universal Windows Platform* i programski jezik *C#.*

Unutar šestog poglavlja detaljno će se prikazati izrada biblioteke (engl. *library*) sa metodama za računanje s Vandermondeovom matricom i determinantom za čiju izradu će se koristi prethodno objašnjena teorija. Naknadno će se opisati izrada sučelja te rukovanje s iznimkama. Nakon opisa izrade aplikacije, pokazana su odrađena testiranja i usporedba rezultata dobivenih aplikacijom završnog rada i slične.

Na kraju, u zaključku dajemo osvrt na postignuti cilj prilikom računanja s Vandermondeovom matricom i determinantom u programskom jeziku C#. Također, sažete su sve zamijećene koristi i nedostatci prilikom izrade aplikacije završnog rada.

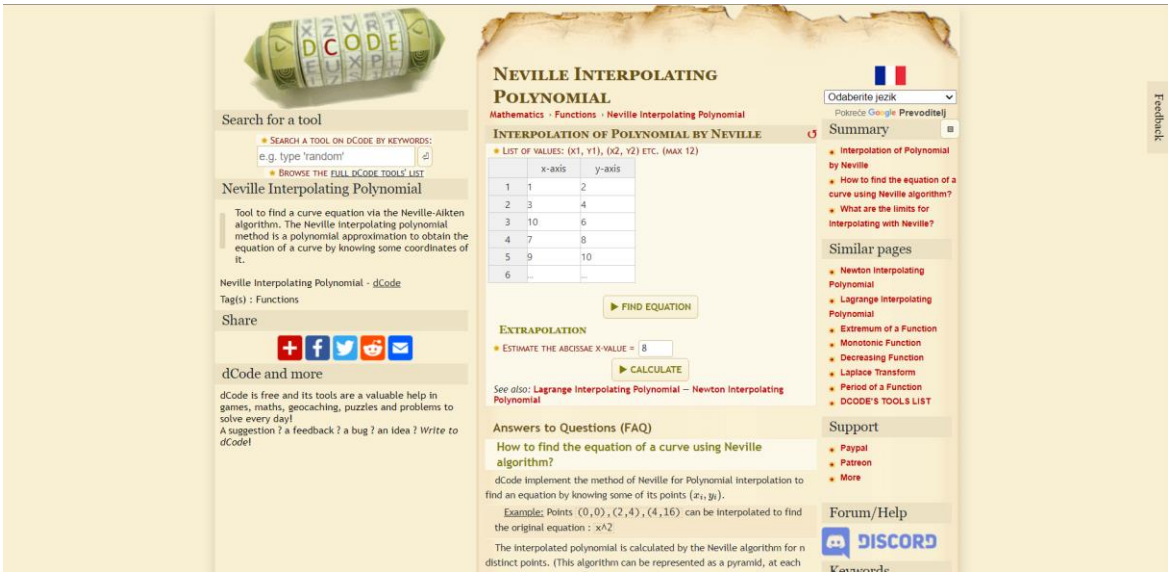
1.1. Zadatak završnog rada

Zadatak završnog rada je pomoću C# programskog jezika izraditi UWP (*Universal Windows Platform*) aplikaciju koja od korisnika traži broj točaka koje želi unesti te za koje postoji polinom. Nadalje, aplikacija za korisnički unesene točke traži polinom procesom polinomijalne interpolacije pri čemu se koriste svojstva Vandermondeove matrica i determinante. Grafičko sučelje kao rezultat izbacuje dobiveni polinom i detaljne korake pronalaska tog polinoma izračunom inverza Vandermondeove matrice.

2. PREGLED PODRUČJA TEME

Pregledavajući internet (engl. *Web*) možemo naići na velik broj različitih matematičkih kalkulatora tako i kalkulatora koji primarno računaju s matricama, no kad je u pitanju rad s Vandermondeovom matricom i determinantom taj broj je nešto manji. Neke aplikacije i *web* stranice također imaju neku vrstu kalkulatora polinomijalne interpolacije, ali većina njih koristi drugačije metode izračuna polinom od onoga obrađenog u ovome radu.

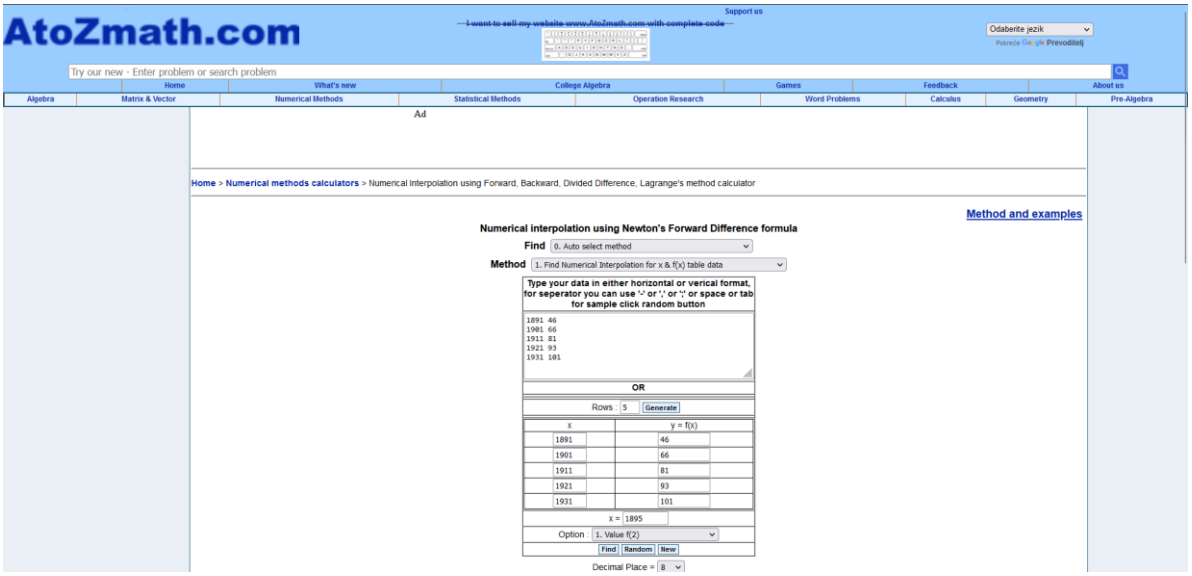
Jedan takav kalkulator, koji se kasnije u radu koristi za usporedbu dobivenih rezultata, nalazi se na stranici je dCode.fr (prikazana slikom 2.1) na poveznici „https://www.dcode.fr/newton-interpolating-polynomial“. Taj kalkulator koristi Newtonovu metodu za izračun interpolirajućeg polinoma. Stranica je dosta jednostavno napravljena sa dovoljno dobrim pregledom metode koja se koristi. Unos podataka je malo otežan jer se svaki podatak mora unositi jedan po jedan. Ovaj kalkulator veoma brzo izbacuje rješenja sa par koraka postupka dobivanja rješenja. Dodatno, stranica nudi još dvije vrstu kalkulatora polinomijalne interpolacije koji računaju preko drugih metoda.



Sl. 2.1 Početni zaslon kalkulatora na *web* stranici dCode.fr

Drugi primjer stranice s kalkulatorom polinomijalne interpolacije je stranica atozmath.com (prikazana slikom 2.1). Kalkulator se može pronaći na poveznici „https://atozmath.com/CONM/NumeInterPola.aspx“. Ova stranica veliki broj metoda i formula za izračun interpolirajućeg polinoma uz još dodatne opcije prikaza rezultata. Uz to svaki postupak izračuna je detaljno prikazan. Međutim, ova stranica ima dvije mane. Prvo, stranica

ima veliki broj oglasa gdje su neki od njih video zapisi. Drugo, za prikaz rješenja stranica traži od korisnika 15 sekundi čekanja nakon pritiska na prikaz rješenja.



Sl. 2.2 Početni zaslon kalkulatora na *web* stranici atozmath.com

Za kraj treba spomenuti da višenamjenska aplikacija MATHLAB sadrži funkciju sa ključnom riječju *vander* koja za uneseni vektor predaje Vandermondeovu matricu toga vektora. Ta funkcija je ista kao jedna od metoda unutar ove aplikacije koje će biti kasnije dodatno pojašnjena. Dokumentacija za tu funkciju nalazi se na poveznici „https://www.mathworks.com/help/matlab/ref/vander.html". Ova funkcija lako se može koristiti u kombinaciji sa funkcijama za interpolaciju polinoma koje MATHLAB sadrži.

3. TEORIJSKA PODLOGA ZADATKA

3.1. Vandermondeova matrica

Vandermondeova matrica je posebna vrsta kvadratne matrice koja se koristi u matematici i linearnoj algebri. Ime je dobila po francuskom matematičaru Alexandreu-Théophile Vandermondeu [1]. Vandermondova matrica ima oblik matrice V:

$$\boldsymbol{V} = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^n \\ 1 & x_2 & x_2^2 & \cdots & x_2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n+1} & x_{n+1} & \cdots & x_{n+1}^n \end{bmatrix} \tag{3-1}$$

u kojoj vrijedi da je $x_i \neq x_j$ za sve $i \neq j$.

Dakle, matrica ima *n+1* redova i *n+1* stupaca, pri čemu je *n+1* broj različitih elemenata koje se potenciraju. Glavno svojstvo ove matrice je da se elementi *x* unutar retka *i, i = 1, 2, ..., n+1*, potencira po *j* po stupcu *j, j = 0, 1, ..., n*.

Važna je napomenuti da prema standardnoj linearnoj algebri vrijedi da je Vandermondova matrica V, invertibilna ako i samo ako vrijedi da je $\det(V) \neq 0$.

Vandermondeova matrica i njena determinanta imaju široku primjenu u područjima kao što su statistika, interpolacija i numerička analiza. Također se koriste u mnogim primjenama izvan matematike, kao što su digitalna obrada signala, inženjering i računalna grafika. Unutar ovoga završnog rada Vandermondeova matrica i determinanta koristiti će se pri izračunu polinoma u procesu polinomijalne interpolacije.

3.1.1. Vandermondeova determinanta

Vandermondeova determinanta ima poseban oblik i može se izračunati na sljedeći način:

$$det(V) = \prod_{0 \leq i < j}^{n} (x_j - x_i) \tag{3-2}$$

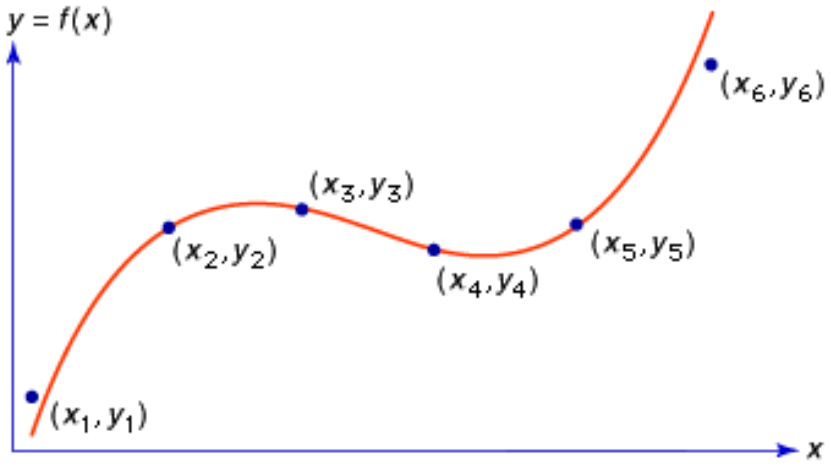
pri čemu vrijedi da nije nula ako i samo ako su svi x_i distinktni.[2]

Determinanta Vandermondeove matrice je stoga produkt svih razlika između svaka dva elementa u matrici.

3.2. Polinomijalna interpolacija

Polinomijalna interpolacija je matematička metoda koja se koristi za pronalaženje polinoma koji prolazi kroz zadane točke u prostoru, to jest kroz skup koordinata x_i, y_j. Cilj interpolacije je da se polinom "prilagodi" tim točkama na najbolji mogući način.

Ako su dati $n+1$ distinktne točke $(x_0, y_0), \ldots, (x_n, y_n)$ postoji jedinstveni polinom n-tog stupnja $p_n(x)$ interpolira x_i, y_i.



Sl. 3.1 Primjena polinomijalne interpolacije[3]

Na slici 3.1 je prikazan primjer dobivenog polinoma nakon polinomijalne interpolacije nad 6 distinktnih točaka.

Postoji nekoliko pristupa polinomijalnoj interpolaciji kao što su Newtonov interpolacijski oblik i Lagrangeov interpolacijski oblik. No kao što je u uvodu spomenuto unutar ovog završni rad koristiti ćemo se interpolacijom s Vandermondeovom matricom.

3.2.1. Interpolacija s Vandermondeovom matricom

Interpolacija s Vandermondeovom matricom [4] je jedna od matematičkih metoda koja koristi Vandermondeovu matricu kako bi se konstruirao interpolacijski polinom.

Vandermondeova matrica, kao što smo ranije spomenuli, je matrica koja se sastoji od potencija različitih zadanih skalara. U kontekstu interpolacije, Vandermondeova matrica se gradi na temelju poznatih točaka interpolacije, gdje su skalarne vrijednosti obično x-koordinate tih točaka.

Ako imamo $n + 1$ distinktnih točki, možemo stvoriti isti broj jednadžbi njihovim uklapanjem

na polinom n-tog stupnja:

$$p_n(x) = a_0 + a_1 x + a_2 x^2 + a_3 x^3 + \ldots + a_n x^n. \tag{3-3}$$

Ovo se zove polinomijalna interpolacija kada je $y_i = p_n(x_i)$ za sve (x_i, y_i), $i = 1, 2, ..., n + 1$.

Sustav jednadžbi za interpolaciju polinoma je:

$$\begin{aligned} y_1 &= a_0 + a_1 x_1 + a_2 x_1^2 + a_3 x_1^3 + \ldots + a_n x_1^n \\ y_2 &= a_0 + a_1 x_2 + a_2 x_2^2 + a_3 x_2^3 + \ldots + a_n x_2^n \\ &\vdots \\ y_{n+1} &= a_0 + a_1 x_{n+1} + a_2 x_{n+1}^2 + a_3 x_{n+1}^3 + \ldots + a_n x_{n+1}^n \end{aligned} \tag{3-4}$$

Jednadžba matrica koja proizlazi iz ovog sustava jednadžbi je $\boldsymbol{Y = V \cdot A}$:

$$\begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_{n+1} \end{bmatrix} = \begin{bmatrix} 1 & x_1 & x_1^2 & \cdots & x_1^n \\ 1 & x_2 & x_2^2 & \cdots & x_2^n \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{n+1} & x_{n+1} & \cdots & x_{n+1}^n \end{bmatrix} \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_n \end{bmatrix} \tag{3-5}$$

Nadalje, ako okrenemo prethodnu jednadžbu $\boldsymbol{Y = V \cdot A \rightarrow A = V^{-1} \cdot Y}$, te ako vrijedi da inverz od V matrice postoji to jest ako vrijedi da je det(V) ≠ 0. Daljnjim računanjem dolazi se do matrice A. Dobivenom matricom A dobili smo i sve potrebne vrijednosti koje unosimo u formulu (3-3) te dobivamo polinoma za zadane točke.

4. KORIŠTENE TEHNOLOGIJE I OKRUŽENJA

Kao što je već u uvodu spomenuto za izradu aplikacija je korišteno razvojno okruženje Microsoft Visual Studio, platforma UWP i programski jezik C#. U ovome poglavlju kratko je opisan rad, povijest i koristi pojedinih programskih alata koji su korišteni pri izradi aplikacije.

4.1. Programski jezik C#

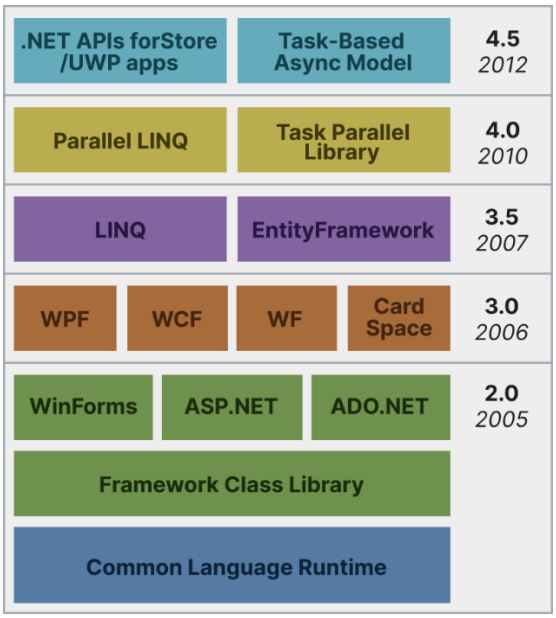
C# (izgovara se "*C sharp*") [5] je objektno-orijentiran programski jezik koji je razvila tvrtka Microsoft. Pojavu C#-a kao programskog jezika obilježava sredina 1990-ih. Prva verzija, C# 1.0, službeno je predstavljena u veljači 2002. godine kao ključni dio Microsoftove platforme .NET. Nastanak C#-a proizašao je iz potrebe za suvremenim programskim jezikom koji kombinira sigurnost, objektnu orijentiranost, visoku produktivnost te prilagodljivost razvoju aplikacija za Microsoftovu platformu.

Osnovne karakteristike C#-a uključuju ključne principe objektno orijentiranog programiranja, poput klasa, objekata, nasljeđivanja, polimorfizma i enkapsulacije. Ovi koncepti omogućuju programerima strukturiranje koda na logičan način, unapređuju modularnost, olakšavaju održavanje i potiču ponovnu upotrebu koda.

Klase su predlošci ili obrasci koji definiraju strukturu i ponašanje objekata. Klase služe kao planovi za stvaranje objekata. Objekti su konkretne instance klasa. Klase definiraju svojstva (atribute) i metode koje objekti nasljeđuju i koriste. Ostale karakteristike, iako jako korisne, u nisu korištene pri izradi ove aplikacije jer nisu bile nužne za željeni rad aplikacije.

4.2. UWP aplikacija

UWP (*Universal Windows Platform*) je platforma razvijena od strane Microsofta koja omogućava razvoj aplikacija koje se mogu pokretati na različitim uređajima i platformama koje koriste operacijski sustav Windows 10 ili novije verzije. UWP je nastao s ciljem stvaranja jedinstvene platforme koja omogućava programerima da razvijaju aplikacije koje se mogu izvoditi na različitim uređajima kao što su računala, tableti, pametni telefoni i drugim. Osim što omogućava korištenje jednostavnog API-ja na svim uređajima koji pokreću Windows, UWP aplikacija predstavlja sigurno, responzivno i stabilno programersko okruženje, te smanjuje takozvano truljenje softvera (engl. *software rot*). To su samo neki od razloga zbog kojih je korištena UWP aplikacija u ovom završnom radu.



Sl. 4.1 Komponente Microsoft .NET Frameworka [6]

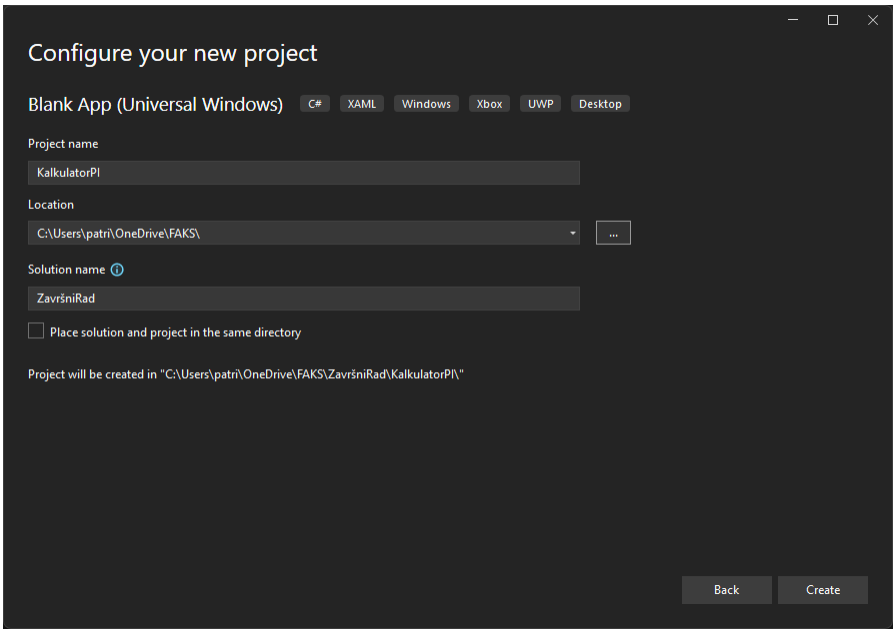
Na slici 4.1 prikazane su komponente Microsoft .NET Framework prema godini izlaska, one omogućuju razvoj i izvođenje različitih vrsta aplikacija, uključujući Windows aplikacije, web aplikacije i servise. Jedna od njih je i UWP aplikacija za koju iz slike vidimo da je izašla 2012. godine, te i danas predstavlja jednu od boljih platformi za izradu aplikacija za Windows uređaje.

4.3. Microsoft Visual Studio

Microsoft Visual Studio je integrirano razvojno okruženje (IDE) koje je korišteno za izradu aplikaciju ovog završnog rad. Ono pruža alate za razvoj različitih aplikacija, uključujući web i desktop aplikacije. Visual Studio podržava različite jezike i integrira se s .NET okvirom te omogućava proširenja i cloud razvoj. Odabrana je ovo okruženje za izradu aplikaciju jer sadrži alate za pisanje koda, otklanjanje pogrešaka (engl. *debugging*), dizajniranje korisničkog sučelja i suradnju programera.

5. RAZVOJ APLIKACIJE

Kao što je prethodno spomenuto, aplikacija je izrađena unutar Microsoft Visual Studia gdje je početni korak bio stvaranje projekta pod nazivom *KalkulatorPI* i rješenja (engl. *solution*) pod nazivom *ZavršniRad.sln*. Kao predložak (engl. *template*) postavljen je *Blank app (Universal Windows)* za C#. Taj proces prikazan je slikom 5.1. Nakon stvaranja projekta, dodana je biblioteka pod nazivom *MatrixLibrary* koji sadrži javnu klasu *MatrixCalc.cs* u kojoj se nalaze sve potrebne metode za računanje s matricama i njenim determinantama u ovome projektu.



Sl. 5.1 Proces stvaranja projekta u programu Microsoft Visual Studio

5.1. Klasa MatrixCalc.cs

Korištenje klase *MatrixCalc.cs* unutar projekta *KalkulatorPI* omogućeno je pomoću implementacije biblioteke *MatrixLibrary* pomoću ključne riječi *using* unutar projekta, što je uvjetovano i ključnom riječju *public* ispred deklaracije klase *MatrixCalc.cs*.

Kako je prethodno spomenuto unutar promatrane klase nalaze metode koje odrađuju računanje s matricama pod nazivima *VandermondeDeterminant, VandermondeMatrix, MatrixProduct* i *MatrixInverse*. U nastavku će se detaljno opisati i objasniti rad svake ove metode.

5.1.1. Metoda *VandermondeDeterminant*

Metoda *VandermondeDeterminat*, kako i njezin naziv upućuje, računa determinantu Vandermondeove matrice. Determinantu Vandermondeove matrice može se izračunati pomoću jednostavne formule (3-2) prikazane u poglavlju 3. Metoda *VandermondeDeterminant* implementira spomenutu formulu za izračun determinante Vandermondeove matrice. Na slici 5.2 prikazan je cjeloviti kod metode.

```
 1.          public static decimal VandermondeDeterminant(decimal[][] matrix)
 2.          {
 3.              int rows = matrix.Length;
 4.              decimal[] vector = new decimal[rows];
 5.              for (int i = 0; i < rows; ++i)
 6.              {
 7.                  vector[i] = matrix[i][1];
 8.              }
 9.              decimal res = 1;
10.              for (int j = 0; j < rows - 1; ++j)
11.                  for (int i = j + 1; i < rows; ++i)
12.                  {
13.                      res *= vector[i] - vector[j];
14.                  }
15.              return res;
16.          }
```

Sl. 5.2 Kod metode *VandermondeDeterminant*

Ključna riječ *public* omogućuje korištenje metode izvan klase *MatrixCalc*. Iduća ključna riječ *static* predstavlja modifikator za deklaraciju *static* članova koji onda pripada samoj klasi za razliku od specifičnog objekta. Riječ *decimal* ispred naziva metode predstavlja povratni tip podatka, što znači da ta metoda na kraju mora vratiti podatak tipa *decimal*. Većina vrijednosti u projektu koji se koriste za izračun su tip *decimal* zbog njegove preciznosti, precizan je na 28 – 29 decimale. Unutar zagrade poslije naziva metode se nalazi argument koji metoda očekuje prilikom pozivanja. U ovome slučaju Metoda zahtjeva da joj se preda argument koji je predstavlja dvodimenzionalni niz (engl. *jagged array*) *decimal* podatka koji će unutar metode biti pod nazivom *'matrix'*.

Unutar prvih vitičastih zagradi nalazi se kod metode koji se odrađuje prilikom poziva metode. Prva linija koda određuje broj redaka matrice tako što dohvaća duljinu vanjskog niza *'matrix'*. Varijabla *'rows*' bit će broj redaka u matrici. Nadalje se stvara niz *'vector'* koji će sadržavati elemente druge kolone matrice. Broj elemenata ovog vektora jednak je broju redaka matrice. Varijabla *'res'* inicijalizirana je na vrijednost 1. Ovdje će se akumulirati rezultat izračunavanja determinante.

Ugniježđena *for* petlja koristi se za prolazak kroz elemente matrice kako bi se izračunala determinanta. Prva petlja (vanjska) iterira pod duljini vektora, od 0 do *rows*-1, dok druga petlja (unutarnja) nastavlja iterirati od trenutne vrijednosti prve petlje *j*+1 do *rows*. Ovdje se izračunava produkt razlika elemenata vektora '*vector*[i] - *vector*[j]' i množi s trenutnom vrijednošću *'res*'. Konačni rezultat koji će biti vraćen iz metode je determinanta Vandermondeove matrice izračunata prema gore navedenom pristupu.

5.1.2. Metoda *VandermondeMatrix*

Metoda *VandermondeMatrix* generira Vandermondeovu matricu na temelju ulaznog vektora tako da svaki redak matrice popunjava potencijama pojedinog element vektora. U nastavku je prikazan kod te metode slikom 5.3.

```
1.          public static decimal[][] VandermondeMatrix(double[] vector)
2.          {
3.              int leng = vector.Length;
4.              decimal[][] matrix = MatrixCreate(leng, leng);
5.              for (int i = 0; i < leng; ++i)
6.                  for (int j = 0; j < leng; ++j)
7.                      matrix[i][j] = (decimal)Math.Pow(vector[i], j);
8.              return matrix;
9.          }
```

Sl. 5.3 Kod metode *VandermondeMatrix*

Ova metoda uzima kao ulaznu vrijednost *double* niz pod nazivom '*vector'*. Ta vrijednost je tipa *double* zato što će elementi predani u sučelju ključnom vrijednosti *Value* koja vrača taj tip podatka, no prilikom izračuna u metodi vrijednosti se pretvaraju u *decimal* tip podatka. Varijabla '*leng'* predstavlja dužinu predanog vektora. Idući korak je stvaranje matrice *'matrix'* metodom *MatrixCreate* s dimenzijama (*leng* , *leng*).

```
1.          static decimal[][] MatrixCreate(int rows, int cols)
2.          {
3.              decimal[][] res = new decimal[rows][];
4.              for (int i = 0; i < rows; ++i)
5.                  res[i] = new decimal[cols];
6.              return res;
7.          }
```

Sl. 5.4 Kod metode *MatrixCreate*

MatrixCreate je privatna metoda jer nije mijenjana razina pristupačnosti (engl. *accessibility level*), koristi se samo unutar klase. Kod ove metode prikazan je slikom 5.4. Ona prvo stvara i inicijalizira praznu matricu zadanim brojem redaka *'rows'*. Nakon toga se u svakom koraku petlje stvara novi jednodimenzionalni niz koji predstavlja redak matrice i dodjeljuje se

odgovarajućem indeksu matrice *'res'*. Na kraju, metoda vraća stvorenu matricu. U ovome slučaju je će ona biti kvadratna matrica jer su predane varijable iste vrijednosti.

Unutar ugniježđenih petlji, izračunava se vrijednost elementa na poziciji (*i*, *j*) u matrici. To se odrađuje podizanjem *i*-tog elementa vektora na potenciju *j* pomoću funkcije *Math.Pow*. Rezultat se pretvara u decimalni tip podataka kako bi odgovarao tipu podataka matrice. Konačno, dovršena matrica predstavlja Vandermondeovu matricu i vraća se kao rezultat funkcije VandermondeMatrix.

5.1.3. Metoda *MatrixProduct*

Ova metoda omogućuje množenje dviju matrica, uz provjeru dimenzija kako bi se osiguralo da su matrice ulančane za množenje. Rezultat je nova matrica koja predstavlja produkt matrica '*matA'* i '*matB'*. Kod ove metoda je prikazana slikom 5.5.

```
 1.          public static decimal[][] MatrixProduct(decimal[][] matA,
 2.          decimal[][] matB)
 3.         {
 4.               int aRows = matA.Length;
 5.               int aCols = matA[0].Length;
 6.               int bRows = matB.Length;
 7.               int bCols = matB[0].Length;
 8.               if (aCols != bRows)
 9.                    throw new Exception("Matrice A i B ne
10.                    možemo pomnožiti jer nisu ulančane, " +
11.                    "odnosno broj stupaca matrice A
12.                    je drukčiji od broja redaka matrice B.");
13.               decimal[][] res = MatrixCreate(aRows, bCols);
14.               for (int i = 0; i < aRows; ++i)
15.                   for (int j = 0; j < bCols; ++j)
16.                       for (int k = 0; k < aCols; ++k)
17.                           res[i][j] += matA[i][k] * matB[k][j];
18.               return res;
19.          }
```

Sl. 5.5 Kod metode *MatrixProduct*

Ulazni argumenti *decimal[][] matA* i *decimal[][] matB* predstavljaju matrice koje želimo pomnožiti. S linijama 3 i 4 dobiva se broj redaka i broj stupaca matrice *'matA'*, a sa linijama 5 i 6 dobiva se broj redaka i broj stupaca matrice *'matB'*.

Nadalje pošto znamo da matrice moraju biti ulančane da se mnogu množiti dodana je provjera preko uvjetne izjave *'if (aCols != bRows)'*. Matrice se mogu pomnožiti samo ako broj stupaca matrice *'matA'* jednak je broju redaka matrice *'matB'*. Ako ovo nije ispunjeno, Metoda baca iznimku s porukom da matrice nisu ulančane.

Nakon provjere množivosti, deklarira se rezultirajuća matrica *'res'* koja se inicijalizira s pozivom prethodno već spomenute pomoćne metode *MatrixCreate* s predanim vrijednostima *'aRows'* i *'bCols'*.

Tri ugniježđene *for* petlje koriste se za iteraciju kroz elemente ulaznih matrica te za zbrajanje sa matricom *'res'*. Prva petlja (*i* petlja) iterira kroz redove matrice *'matA'*, druga petlja (*j* petlja) iterira kroz stupce matrice *'matB'*, a treća petlja (*k* petlja) koristi se za iteraciju kroz elemente matrica *'matA'* i *'matB'* koji se množe i sumiraju s rezultirajućom matricom *'res'*.

5.1.4. Metoda *MatrixInverse*

Metoda *MatrixInverse* izračunava inverz matrice metodom Gaussove eliminacije. Ova metoda se temelji na principima postupka eliminacije i matematičkih operacija kako bi se transformirala originalna matrica u gornje trokutastu matricu ili identitetsku matricu na lijevoj strani, dok se desna strana transformira u rješenje, to jest inverz matricu [7]. U nastavku je u kratko objašnjen matematički proces računanja inverz matrice metodom Gaussove eliminacije, nakon čega je objašnjeno ostvarivanje istoga u aplikaciji metodom *MatrixInverse*.

Originalnu matricu koju želimo transformirati proširujemo tako da ima dva dijela: lijevi dio sadrži originalnu matricu, a desni dio identitetsku matricu istih dimenzija. Počinjemo iterirati kroz redove matrice, obično od vrha prema dnu. Za svaki red, radimo sljedeće:

Dijelimo sve elemente trenutnog retka kako bismo postavili dijagonalni element (element na dijagonali) na 1. Ovo se radi tako da podijelimo cijeli red s vrijednošću dijagonalnog elementa. Za svaki red ispod trenutnog retka, eliminiramo vrijednosti ispod dijagonalnog elementa tako da postanu nula. Što se postiže tako da oduzmemo odgovarajući faktor ovog retka pomnožen s trenutnim redom.

Nakon završene Gaussove eliminacije, lijevi dio proširene matrice sada sadrži identitetsku matricu ili gornje trokutastu matricu, dok desni dio sadrži inverz originalne matrice.

Na slici 5.6 slijedi kod metode *MatrixInverse*, koja kako je već spomenuto koristi Gaussovu metodu eliminacije za izračun inverza matrice.

```
1.          public static decimal[][] MatrixInverse(decimal[][] matrix)
 2.          {
 3.              int rows = matrix.Length;
 4.              decimal[][] tempMatrix = MatrixCreate(rows, 2 * rows);
 5.              for (int i = 0; i < rows; i++)
 6.              {
 7.                  for (int j = 0; j < rows; j++)
```

```
 8.                 {
 9.                     tempMatrix[i][j] = matrix[i][j];
10.                     tempMatrix[i][j + rows] = (i == j) ? 1.0M : 0.0M;
11.                 }
12.             }
13.             for (int i = 0; i < rows; i++)
14.             {
15.                 decimal point = tempMatrix[i][i];
16.                 for (int j = 0; j < 2 * rows; j++)
17.                 {
18.                     tempMatrix[i][j] /= point;
19.                 }
20.                 for (int j = 0; j < rows; j++)
21.                 {
22.                     if (j != i)
23.                     {
24.                         decimal factor = tempMatrix[j][i];
25.                         for (int k = 0; k < 2 * rows; k++)
26.                         {
27.                           tempMatrix[j][k] -= factor * tempMatrix[i][k];
28.                         }
29.                     }
30.                 }
31.             }
32.             decimal[][] res = MatrixCreate(rows, rows);
33.             for (int i = 0; i < rows; i++)
34.             {
35.                 for (int j = 0; j < rows; j++)
36.                 {
37.                     res[i][j] = tempMatrix[i][j + rows];
38.                 }
39.             }
40.             return res;
41.         }
42.     }
```

Sl. 5.6 Kod metode *MatrixInverse*

Na početku metode inicijalizirana je vrijednost *'rows'* koja predstavlja broj redaka originalne matrice. Nadalje se stvaranja privremena matrica *'tempMatrix'* s dimenzijama (*rows*, (2 * *rows*)) s pomoćnom metodom *MatrixCreate*. Ova privremena matrica služi za pohranu originalne matrice s desnom stranom identitetske matrice.

Matrica *'tempMatrix'* inicijalizira se tako da s lijeve strane sadrži originalnu matricu, dok s desne strane ima identitetsku matricu iste veličine. U dvije ugniježđene *for* petlje prolazi se kroz sve elemente matrice *'tempMatrix'*. Prva petlja (*i* petlja) prolazi kroz redove originalne matrice, a druga petlja (*j* petlja) prolazi kroz stupce. Element *tempMatrix[i][j]*, dobiva istu vrijednost kao odgovarajući element originalne matrice *matrix[i][j]*. Tako se lijeva strana matrice *'tempMatrix'* postavlja na vrijednosti originalne matrice *'matrix'*. Elementi *tempMatrix[i][j + rows]* postavljaju se na 1.0 ako *i* i *j* imaju iste indekse (dijagonala identitetske matrice), inače se postavljaju na 0.0.

Slijedi primjena Gaussove eliminacije na *'tempMatrix'* kako bi se postigla identitetska matrica na lijevoj strani matrice, dok će se desna strana transformirati u inverz originalne matrice. Prva *for* petlja (*i* petlja) prolazi kroz redove matrice od 0 do *rows* - 1. U svakom koraku petlje, element *point* dobiva vrijednost *tempMatrix[i][i]*, tj. dijagonalni element trenutnog retka. Zatim se *for* petljom (*i* petlja) ide kroz sve elemente tog retka od 0 do 2 * *rows* - 1 i svaki element dijeli se s *point*. Time se osigurava da dijagonalni element postane 1, a drugi elementi tog retka se skaliraju tako da budu relativno isti. Nakon toga, *for* petljom prolazi se kroz sve redove matrice od 0 do *rows* - 1 osim trenutnog retka kako bi se neutralizirali elementi u tim redovima i postigla identitetska matrica na lijevoj strani.

Nakon provedene Gaussove eliminacije, identitetska matrica nalazi se na lijevoj strani matrice *'tempMatrix'*, dok se inverz originalne matrice nalazi na desnoj strani. Stvara se nova matrica *'res'* dimenzija (*rows, rows*), ponovno pomoću metode *MatrixCreate*, koja će sadržavati inverz originalne matrice. Zadnjom *for* petljom kopiraju se elementi s desne strane matrice *'tempMatrix'* u matricu *'res'*, čime se dobiva inverz originalne matrice. U zadnjoj liniji Metoda vraća rezultirajuću matricu *'res'*.

5.2. Korisničko sučelje aplikacije

Korisničko sučelje (engl. *User interface*, skraćeno UI) predstavlja određeni prostog gdje se odvija radnja između čovjeka i stroja, najčešće računala. Kao što je prethodno spomenuto projekt je izrađen kao UWP aplikacija, tako da je izgled korisničkog sučelja XAML UWP aplikacije, koji predstavlja deklarativan označni jezik.

Korisničko sučelje ove aplikacije izrađeno je unutar jednog prozora što znači da imamo jednu XAML datoteku na kojoj se odrađuje sve radnje aplikacije. U nastavku će biti opisan izgled pojedinog dijela sučelja i metode interaktivnih objekata u sučelju te opisan rad s iznimkama preko istočnih prozora (engl. *pop-up window*).

Sl. 5.7 Početni zaslon aplikacije

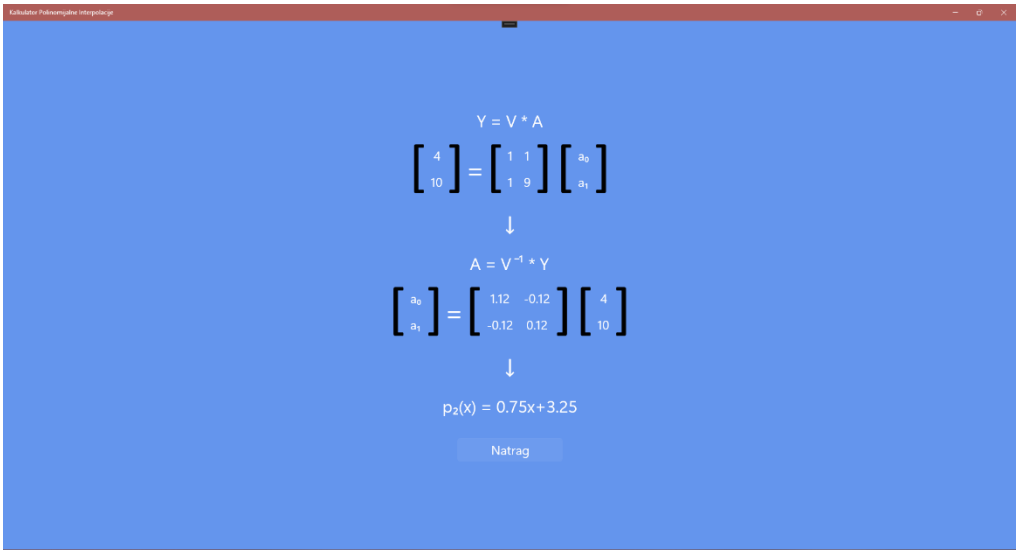
Na početnom zaslonu, koji je prikazan slikom 5.7, nalazi se naslov i podnaslov aplikacije unutar elementa *TextBlock*. Ispod njih se nalazi napomena koja upozorava da je aplikacija limitirana na odabir unosa od 2 do 8 točaka za izračun polinoma koji ih interpolira. Do napomene se nalazi element *NumberBox* koji je dodatno implementiran u aplikaciju pomoću linije koja deklarira vanjski imenik:

```
xmlns:controls="using:Microsoft.UI.Xaml.Controls"
```

Taj element je pod nazivom *numberNode* i funkcija mu je primiti korisnički unesenu *int* vrijednost koja određuje koliko se točaka unosi. Nakon što je odabran željeni broj točaka pritiskom na element *Button* s natpisom „Dalje“ sučelje mijenja izgled te je prikazano slikom 5.8.

Sl. 5.8 Sučelje nakon pritiska dugmeta s natpisom „Dalje“

Nakon pritiska dugmeta s natpisom „Dalje“ sa sučelja nestaju napomena, element za odabir broj točaka i samo pritisnuto dugme. Nadalje, na sučelju se pojavljuje elementi *NumberBox* koji predstavljaju mjesto za unos koordinata točaka, dva dugmeta i skica matrica koje se popunjava s danim koordinatama. Dugme s natpisom „Natrag“ vraća korisnika na početni zaslon, a dugme s natpisom „Izračunaj“ odrađuje računanje nad Vandermondeovom matricom, računa polinom korisnički unesenih točaka te se sučelje ponovno mijenja. Promjenjeno sučelje prikazano je slikom 5.9.



Sl. 5.9 Sučelje nakon pritiska dugmeta s natpisom „Izračunaj“

Ovo je zadnji izgled sučelja koji se pojavljuje ako je izračun polinomijalne interpolacije točaka prošao bez iznimki. Kod na slici 5.10 odgovoran je za prikaz podatke na sučelju u slici 5.9.

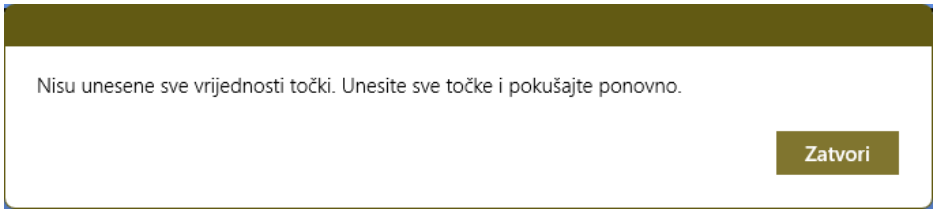
```
1.          private void CalcButton_Click(object sender, RoutedEventArgs e)
2.          {
3.              if (nodeNumber.Value == 2)
4.              {
5.                  if (
6.                      x0.Value.Equals(double.NaN) || y0.Value.Equals(double.NaN) ||
7.                      x1.Value.Equals(double.NaN) || y1.Value.Equals(double.NaN)
8.                     )
9.                  {
10.                     var dialog = new MessageDialog("Nisu unesene sve vrijednosti " +
11.                         "točki. Unesite sve točke i pokušajte ponovno.");
12.                     dialog.Commands.Add(new UICommand("Zatvori", null));
13.                     dialog.ShowAsync();
14.                 }
15.                 else
16.                 {
17.                     mn2x01.Text = x0.Value.ToString();
18.                     mn2x11.Text = x1.Value.ToString();
19.                     mn2y0.Text = y0.Value.ToString();
20.                     mn2y1.Text = y1.Value.ToString();
21.                     imn2y0.Text = y0.Value.ToString();
22.                     imn2y1.Text = y1.Value.ToString();
23.                     double[] v = new double[] { x0.Value, x1.Value };
24.                     decimal[][] mx = MatrixCalc.VandermondeMatrix(v);
25.                     decimal[][] my = new decimal[][] {
26.                         new decimal[] { ((decimal)y0.Value) },
```

```
                    new decimal[] { ((decimal)y1.Value) }};
                if (MatrixCalc.VandermondeDeterminant(mx) == 0)
                {
                    var dialog = new MessageDialog("Nije moguće dobiti rješenje! " +
                        "Determinanta matrice za unesene točke jednaka je nuli, " +
                        "inverz te matrice ne postoji.");
                    dialog.Commands.Add(new UICommand("Zatvori", null));
                    dialog.ShowAsync();
                }
                else
                {
                    decimal[][] inv = MatrixCalc.MatrixInverse(mx);
                    decimal[][] a = MatrixCalc.MatrixProduct(inv, my);
                    imn2x00.Text = Math.Round(inv[0][0], 2).ToString();
                    imn2x10.Text = Math.Round(inv[1][0], 2).ToString();
                    imn2x01.Text = Math.Round(inv[0][1], 2).ToString();
                    imn2x11.Text = Math.Round(inv[1][1], 2).ToString();
                    string str = $"p\x2082(x) = ";
                    string eform = "0.00e00";
                    if (a[1][0] > 0)
                    {
                        if (a[1][0] < 0.005M)
                            str += $"{a[1][0].ToString(eform)}x";
                        else
                            str += $"{a[1][0].ToString("F2")}x";
                    }
                    if (a[1][0] < 0)
                    {
                        if (a[1][0] > -0.005M)
                            str += $"{a[1][0].ToString(eform)}x";
                        else
                            str += $"{a[1][0].ToString("F2")}x";
                    }
                    if (a[0][0] > 0)
                    {
                        if (str[str.Length - 1] != ' ') { str += "+"; }
                        if (a[0][0] < 0.005M)
                            str += $"{a[0][0].ToString(eform)}";
                        else
                            str += $"{a[0][0].ToString("F2")}";
                    }
                    if (a[0][0] < 0)
                    {
                        if (a[0][0] > -0.005M)
                            str += $"{a[0][0].ToString(eform)}";
                        else
                            str += $"{a[0][0].ToString("F2")}";
                    }
                    p2.Text = str;
                    inputGrid.Visibility = Visibility.Collapsed;
                    outputGrid2.Visibility = Visibility.Visible;
                }
            }
            …
```

Sl. 5.10 Dio koda koji se odvija prilikom pritiska na dugme s natpisom „Izračunaj“ pri odabiru 2 točke

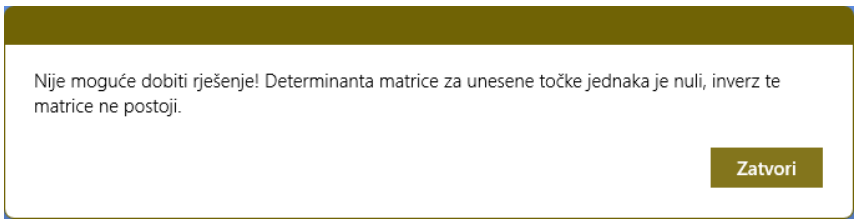
Na sučelju možemo vidjeti izračun polinomijalne interpolacije točaka korak po korak do krajnjeg rješenja te dani kod pruža funkcionalnost toga sučelja. Metoda *CalcButton_Click* predstavlja metodu koja se poziva prilikom pritiska na dugme s natpisom „Izračunaj“. Metoda počinje s *if* uvjetnom naredbom koja provjerava dali su odabrane dvije točke prilikom unosa. Sljedeća *if else* uvjetna naredba upravlja sa slučajem kad može doći do iznimke prilikom upisivanja vrijednosti koordinata točaka. Ako prilikom pritiska na dugme s natpisom „Izračunaj“ neki od mjesta nisu popunjena pojavljuje se iskočni prozor s porukom koji je

prikazan slikom 5.11 i aplikacija ne mijenja izgled sučelja nego čeka unos ispravnih podataka. Iskočni prozor se zatvara na dugme s natpisom „Zatvori“.



Sl. 5.11 Iskočni prozor 1

Prva jednadžba matrica koja je prikazana u sučelju je: $\boldsymbol{Y} = \boldsymbol{V} \cdot \boldsymbol{A}$, linijama 17 – 22 unesene su vrijednosti tih matrica u prikaz na sučelju. Nakon toga koristi se Metoda *VandermondeMatrix* za stvaranje Vandermondove matrice iz unesenih vrijednosti koje se predaju toj funkciji vektorom *v*. Iduća *if else* uvjetna naredba također ima metodu hvatanja iznimki. U ovome slučaju ona provjerava da li je dana Vandermondeova matrica inverzibilna te zato koristi prethodno opisanu metodu *VandermondeDeterminant*. Iz teorije u poglavlju 2. znamo da bi matrica bila inverzibilna, ne smije imati determinantu koja je jednaka 0. *If* dio naredbe izbacuje drugi iskočni prozor u slučaju da je determinanta jednaka 0, prikazan je slikom 5.12.



Sl. 5.12 Iskočni prozor 2

U nastavku se pod linijom 39 računa inverz Vandermondove matrice uz pomoć metode *MatrixInverse*. Dobivene vrijednosti prikazane su u sučelju u drugoj jednadžbi matrica koja je jednaka: $\boldsymbol{A} = \boldsymbol{V^{-1}} \cdot \boldsymbol{Y}$, njezin prikaz ostvaren je linijama 41 – 44. Linija 40 množi matrice $\boldsymbol{V^{-1}}$ i $\boldsymbol{Y}$ pomoću metode *MatrixProduct* da bi se dobila vrijednost matrice $\boldsymbol{A}$ kako je pokazano u jednadžbi. Rezultati dobiveni množenjem prikazani su polinom u na kraju sučelja, za uljepšani prikaz odgovorne su linije 45 – 76.

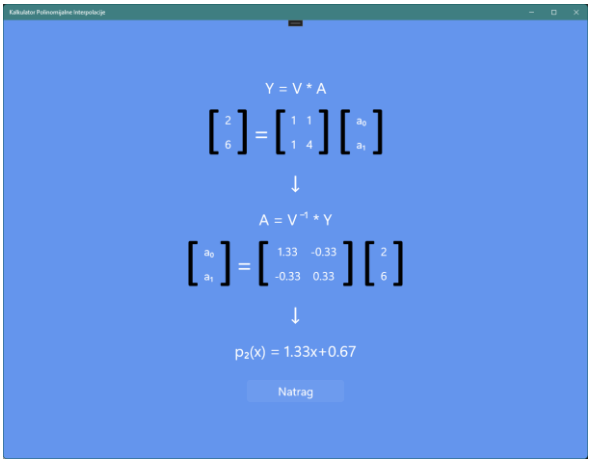
5.3. Testiranje i analiza rezultata

Testiranje je provedeno na krajnjem rješenju, to jest na polinomu koji opisuje unesene točke. Odrađeno je sedam testiranja, tako da je napravljena analiza za svaki izlaz odabira broja točaka, od 2 točke do 8 točki. Sučelja sa analiziranim rezultatima prikazana su slikama 5.13 – 5.19. Za usporedbu s aplikacijom korištena je *web* stranica dCode.fr koja je već spomenuta u unutar drugog poglavlja.

Ako usporedimo dobivene rezultate iz aplikacije s rezultatima s *web* stranice možemo uočiti da ako zanemarimo oblikovanje *string-a* rezultati se poklapaju u svakome primjeru. Također važno je napomenuti da su vrijednosti zaokružene na dvije decimale zbog velikih vrijednosti što pogotovo zna biti problematično na primjera s osam točki. Zbog toga u nekim slučajima može izgledati kao da je vrijednost 0, ali nije nego je vrijednost samo zaokružena.

Testiranje rješenja za dvije unesene točke:

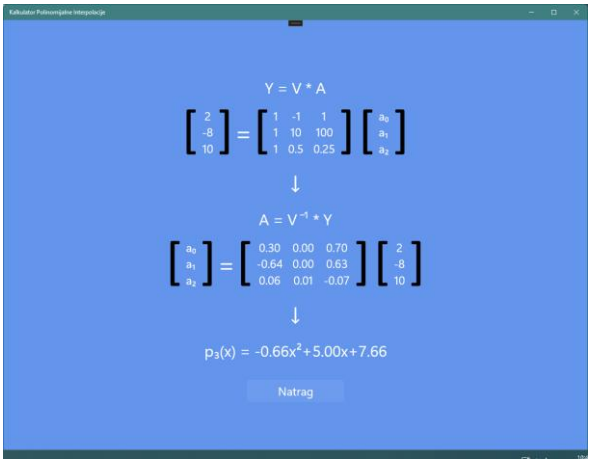
- Unesene točke: {{1,2},{4,6}}
- Rezultat sa stranice dCode: $\frac{4}{3}x + \frac{2}{3}$
- Rezultat iz aplikacije: 1.33x + 0.67



Sl. 5.13 Primjer dobivenog rješenja za dvije točke

Testiranje rješenja za tri unesene točke:

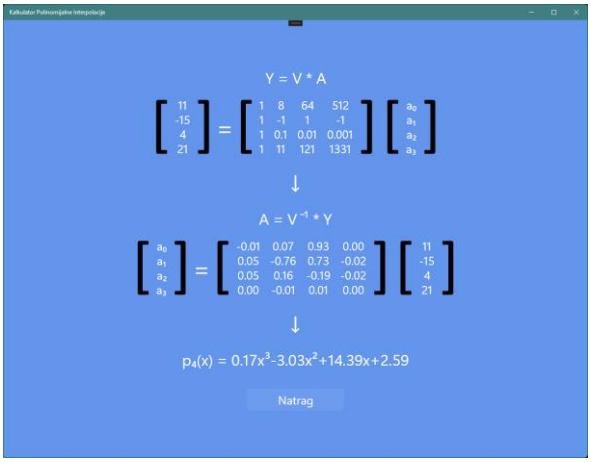
- Unesene točke: {{-1,2},{10,-8},{0.5,10}}
- Rezultat sa stranice dCode: $-0.657097x^2 + 5.00478x + 7.66188$
- Rezultat iz aplikacije: $-0.66x^2 + 5.00x + 7.66$



Sl. 5.14 Primjer dobivenog rješenja za tri točke

Testiranje rješenja za četiri unesene točke:

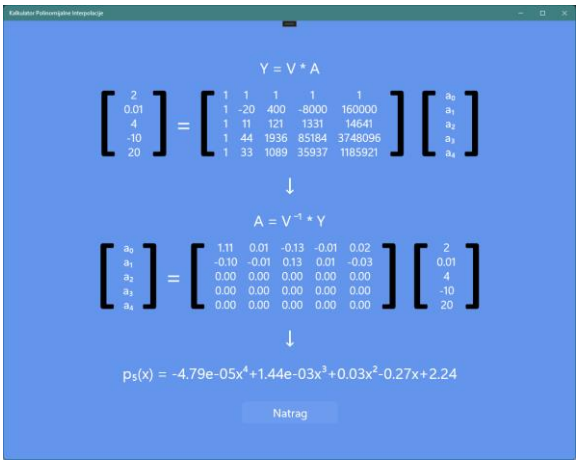
- Unesene točke: {{8,11},{-1,-15},{0.1,4},{11,21}}
- Rezultat sa stranice dCode: $0.170438x^3 - 3.03085x^2 + 14.3899x + 2.59115$
- Rezultat iz aplikacije: $0.17x^3 - 3.03x^2 + 14.39x + 2.59$



Sl. 5.15 Primjer dobivenog rješenja za četiri točke

Testiranje rješenja za pet unesenih točki:

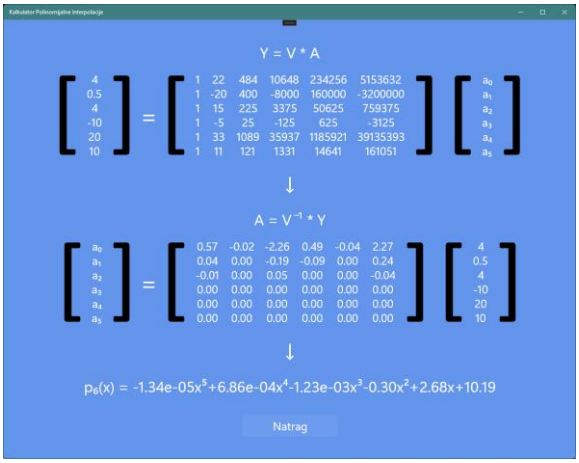
- Unesene točke: {{1,2},{-20,0.01},{11,4},{44,-10},{33,20}}
- Rezultat sa stranice dCode: $-0.0000478832x^4 + 0.00144392x^3 + 0.0289759x^2 - 0.269651x + 2.23928$
- Rezultat iz aplikacije: $-4.79e\text{-}05x^4 + 1.44e\text{-}03x^3 + 0.03x^2 - 0.27x + 2.24$



Sl. 5.16 Primjer dobivenog rješenja za pet točki

Testiranje rješenja za šest unesenih točki:

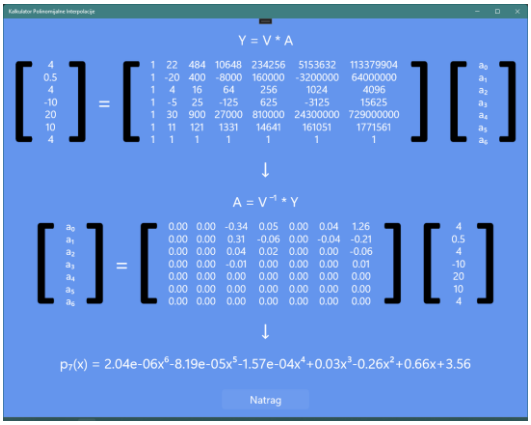
- Unesene točke: {{22,4},{-20,0.01},{11,4},{44,-10},{33,20},{11,10}}
- Rezultat sa stranice dCode: $-0.000013419x^5 + 0.000686469x^4 - 0.00123253x^3 - 0.296846x^2 + 2.67958x + 10.194$
- Rezultat iz aplikacije: $-1.34e\text{-}05x^5 + 6.86e\text{-}04x^4 - 1.23e\text{-}03x^3 - 0.30x^2 + 2.68x + 10.19$



Sl. 5.17 Primjer dobivenog rješenja za šest točki

Testiranje rješenja za sedam unesenih točki:

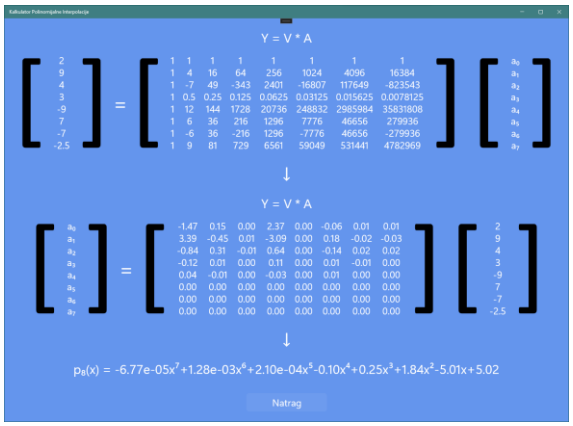
- Unesene točke: {{22,4},{-20,0.5},{4,4},{-5,-10},{30,20},{11,10},{1,4}}
- Rezultat sa stranice dCode: $2.0426290718131...\times10^{-6}x^6 - 0.0000819178x^5 - 0.000156863x^4 + 0.0317689x^3 - 0.258504x^2 + 0.663853x + 3.56312$
- Rezultat iz aplikacije: $2.04e\text{-}06x^6 - 8.19e\text{-}05x^5 - 1.57e\text{-}04x^4 + 0.03x^3 - 0.26x^2 + 0.66x + 3.56$



Sl. 5.18 Primjer dobivenog rješenja za sedam točki

Testiranje rješenja za osam unesenih točki:

- Unesene točke: {{1,2},{4,9},{-7,4},{0.5,3},{12,-9},{6,7},{-6,-7},{9,-2.5}}
- Rezultat sa stranice dCode: $0.0000676951x^7 + 0.00127633x^6 + 0.000209872x^5 - 0.100843x^4 + 0.251735x^3 + 1.83678x^2 - 5.00942x + 5.02032$
- Rezultat iz aplikacije: $-6.77e\text{-}05x^7 + 1.28e\text{-}03x^6 + 2.10e\text{-}04x^5 - 0.10x^4 + 0.25x^3 + 1.84x^2 - 5.01x + 5.02$



Sl. 5.19 Primjer dobivenog rješenja za osam točki

6. ZAKLJUČAK

Ovaj završni rad pokrio je različite aspekte matematičkih i programskih tema, s glavnim zadatkom povezivanja C# programski jezik i Vandermondeovih matrica i determinanti. Pokušalo se stvoriti što bolju vezu između ta dva pojma, tijekom čega su gledane koristi i nedostatci pojedinog pojma u njihovoj suradnji.

Kako bi se najbolje proučila ta veza dodan je posredni element, polinomijalnu interpolacija, s kojim se pokazao jednostavnije i razumnije upravljati radom u okruženju C#-a i Vandermondeove matrice. Programski jezik C# pokazao se kao jako prilagodljiva podloga za pisanje matematičke sintakse. Vandermondeova matrica i determinanta također se pokazala kao koristan alat u izračunu polinomijalne interpolacije. Njihova jednostavnost omogućila je da računski dio programa bude što kraći, gdje bih dodatno naglasio kako je metodu za izračun Vendermondeove determinante bila neočekivano lagana za implementirati. Preko izračuna polinomijalne interpolacije pokazalo se još jednom da su matematika i programiranje jako bliski predmeti koji međusobno pripomažu u svome napretku. Gdje znanje osobe u pojedinom području potpomaže razvijanje druge.

S negativne strane izdvojiv je problem preciznosti tipova podatka C#-a kod izračuna s velikim brojem znamenki gdje je dodatno potrebno limitirati rad u tim područjima. Manjak formata za oblikovanje teksta je otežao prikaz matrica i konačnog rezultata te manjak elemenata vezanih uz matrice unutar UWP aplikacije drastično je povećao količinu potrebnog koda za ostvarivanje responzivnog korisničkog sučelja.

Međutim na kraju pozitivne stvari rada u C# daleko prelaze utjecaj negativnih u priči programiranja i matematike.

LITERATURA

[1] Vandermonde Matrix. Wikipedia; 2023.
https://en.wikipedia.org/wiki/Vandermonde_matrix (pristupljeno 2023-09-14)

[2] Evar D. Nering: „Linear Algebra and Matrix Theory“, Second Edition, 1970. ,str 93.

[3] Interpolation | Data fitting, Approximation, Curve fitting | Britannica.
https://www.britannica.com/science/interpolation (pristupljeno 2023-09-14).

[4] MATH365, Polynomial Interpolation
https://jodimead.github.io/files/m365/interpolation.pdf (pristupljeno 2023-09-14)

[5] BillWagner, A tour of C# - Overview - C#
https://learn.microsoft.com/en-us/dotnet/csharp/tour-of-csharp/ (pristupljeno 2023-09-14).

[6] NET Framework. Wikipedia; 2023.
https://en.wikipedia.org/wiki/.NET_Framework (pristupljeno 2023-09-14)

[7] Heinkenschloss, M. Gaussian Elimination and Matrix Inverse. Matrix Analysis.

SAŽETAK

Zadatak ovog završnog rada bio je stvoriti i primijeniti određene metode vezane uz Vandermondoevu matricu i determinantu unutar programskog jezika C#. To je ostvareno pomoću UWP aplikacije u C# jeziku koja je ima metodu izračuna polinomijalne interpolacije korisnički unesenih koordinata točaka. Stvorene su metode *VandermondeDeterminant, VandermondeMatrix, MatrixProduct* i *MatrixInverse* koje su primijenjene unutar aplikacije za računanje sa Vandermondeovom matricom i njezinom determinantom kako bi se u konačnici preko njih izračunala polinomijalna interpolacija unesenih točaka. Koraci procesa izračuna tog polinoma ukratko su prikazani unutar zadnjeg prikaza korisničkog sučelja zajedno s Vandermondeovom matrice i determinante gdje se zamjećuju njihova svojstva. Cilj je bio pokazati korisnosti Vandermondeove matrice i determinant te jednostavnost i kompatibilnost programskog jezika C# za implementaciju matematičkih pojmova.

Ključne riječi: C#, Polinomijalna interpolacija, Vandermondeova matrica, Vandermondeova determinanta

ABSTRACT

Calculation with Vandermonde matrix and determinant in C# programming language

The task of this final thesis was to create and apply certain functions related to the Vandermonde matrix and determinant within the C# programming language. This was achieved using a UWP application in C# language that has the function of calculating the polynomial interpolation of user-entered point coordinates. The functions *VandermondeDeterminant*, *MatrixProduct* and *MatrixInverse* were created and is applied within the calculation application with the Vandermonde matrix and its determinant in order to ultimately calculate the polynomial interpolation of the entered points. Shortened steps of the calculation process of that polynomial are shown within the last view of the user interface together with the Vandermonde matrix and determine where their properties are shown. The goal was to demonstrate the usefulness of the Vandermonde matrix and determinant and the simplicity and compatibility of the C# programming language for implementing mathematical concepts.

Keywords: C#, Polynomial interpolation, Vandermonde matrix, Vandermonde determinant