Web aplikacija za numeričko rješavanje nekih određenih integrala

Jakovac, Danijel

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

Stručni studij

RAZVOJ WEB APLIKACIJE ZA RJEŠAVANJE ODREĐENIH INTEGRALA

Završni rad

Danijel Jakovac

Osijek, 2022.

Sadržaj

1. UVOD 2
1.1. Zadatak završnog rada 2
2. TEORIJSKA PODLOGA 3
2.1. Integral 3
2.1.1. Neodređeni integral 3
2.1.2 Određeni integral 5
2.2 Metode rješavanja neodređenih integrala 5
2.2.2 Metoda supstitucije 5
2.2.3. Metoda parcijalne integracije 6
2.2. Metode rješavanja određenih integrala 6
2.2.1. Metoda trapeza 6
2.2.2. Simpsonova metoda 7
3. KORIŠTENI PROGRAMSKI JEZICI 9
3.1 Python 9
3.1 HTML 12
3.2 JavaScript 12
4. IZRADA PROGRAMA I VIZUALIZACIJA 13
4.1. Opis zadatka 13
4.2. Izrada backend dijela 13
4.2.1 Rute i upravljanje podacima 14
4.2.2 Pretvaranje podataka i formatiranje integralne funkcije 17
4.2.3 Izrada funkcije za integriraciju pomoću metode trapeza 18
4.2.4 Izrada funkcije za integriraciju pomoću Simpsonove metode 20
4.3. Izrada frontend dijela 21
4.3.1 Struktura frontend mape 21
4.3.2 Povezivanje backend i frontend dijela, asinkronizacija 22
4.3.3 Izrada korisničkog sučelja 23
5. ZAKLJUČAK 28
6. LITERATURA 29

1. UVOD

U radu je opisano rješavanje određenih integrala putem web aplikacije. Funkcija koju korisnik želi integrirati unositi će se putem korisničkog sučelja za unošenje matematičkih simbola i funkcija. Osim funkcije za integriranje, korisnik postavlja gornju, doljnju granicu i točnost, odnosno na koliko podintervala će se funkcija podijeliti tokom integriranja.

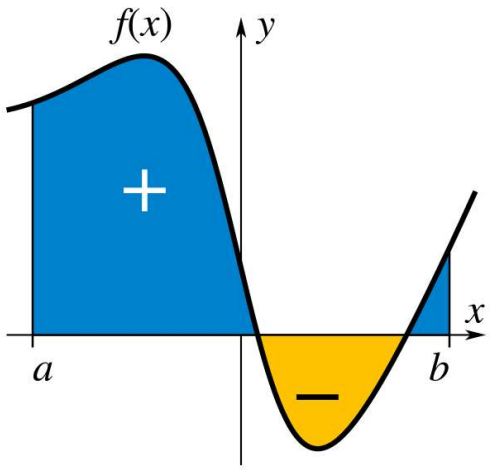
1.1. Zadatak završnog rada

Potrebno je razviti web aplikaciju pomoću koje ćemo moći integrirati željenu funkciju u određenim granicama po Trapeznoj ili Simpsonovoj metodi.

2. TEORIJSKA PODLOGA

2.1. Integral

Integral je ključna koncepcija više matematike. Koncept integriranja oblikovali su u Isaac Newton i Gottfried Wilhelm Leibniz. Integriranje je operacija koja je suprotna deriviranju, odnoso antiderivacija. Većina infinitezimalnih računa u znanosti i inženjerstvu nebi bila moguća bez integriranja. Integral funckije sa slike 2.1 računamo na način da površinu iznad x-osi na granicama od a do b umanjimo za vrijednost površine ispod x-osi za x unutar intervala [a,b]. Integrale dijelimo na određene i neodređene. Integriranje je složeniji postupak od deriviranja. Integral elementarne funkcije nije uvijek elementarna funkcija.



Slika 2.1.: Primjer funkcije

2.1.1. Neodređeni integral

Ukoliko vrijedi pravilo *F'(x) = f(x)*, tada kažemo da je *F(x)* primitivna funkcija funkcije *f(x)*. Skup svih primitivnih funkcija funkcije *f(x)* prikazujemo izrazom:

$$\int f(x)dx = F(x) + C \qquad (2\text{-}1)$$

gdje znak ∫ predstavlja znak za integriranje, *f(x)* podintegralnu funkciju, a *dx* onzaku koja govori po kojoj varijabli se obavlja integriranje. Kako je i prije spomenuto, neodređeni integral elementarne funkcije ne mora uvijek biti elementarna funkcija, neki od neelementarnih integrala su:

$$(1) \quad \int e^{-2x^2} \qquad (2\text{-}2)$$

$$(2) \quad \int \sin(x^2)dx \qquad (2\text{-}3)$$

$$(3) \quad \int \frac{sinx}{x} dx \qquad (2\text{-}4)$$

Za olakšani proces integriranja napravljena je tablica za integriranje osnovnih funkcija.

$\int dx = x + C$	$\int \frac{dx}{a^2 + x^2} = \frac{1}{a} arctan \frac{x}{a} + C$
$\int x^n dx = \frac{x^{n+1}}{n+1} + C, n \neq 1$	$\int \frac{dx}{a^2 - x^2} = \frac{1}{2a} \ln \left\lvert \frac{a+x}{a-x} \right\rvert + C$
$\int \frac{dx}{x} = \ln\lvert x \rvert + C$	$\int \frac{dx}{\sqrt{a^2 + x^2}} = \ln(x + \sqrt{a^2 + x^2}) + C$
$\int e^x dx = e^x + C$	$\int \frac{dx}{\sqrt{a^2 - x^2}} = \arcsin \frac{x}{a} + C$
$\int a^x dx = \frac{a^x}{lna} + C, a > 0, a \neq 1$	$\int \frac{dx}{\sqrt{x^2 - a^2}} = \ln(x + \sqrt{x^2 - a^2}) + C$
$\int sinxdx = -cosx + C$	$\int \frac{dx}{ax + b} = \frac{1}{a} \ln(ax + b) + C$
$\int cosxdx = sinx + C$	

Tablica 2.1.: Tablica integrala

Postoje tri računska pravila integrala. Ukoliko imamo funkciju $\int f(x)$ i funkciju $\int g(x)$ vrijede sljedeća pravila:

$$(1) \quad \int [f(x) + g(x)]dx = \int f(x) + \int g(x)\, dx \qquad (2\text{-}5)$$

$$(2) \quad \int [f(x) - g(x)]dx = \int f(x) - \int g(x)\, dx \quad (2\text{-}6)$$

$$(3) \quad \int cf(x)dx = c \int f(x)\, dx \quad (2\text{-}7)$$

2.1.2 Određeni integral

Kod određenih integrala proces integracije kao rezultat daje konačan broj. Određeni integral definiramo kao površinu ispod krivulje koju definira funkcija *f(x)* unutar granica [a,b]. Zapisujemo ga u obliku:

$$\int_a^b f(x)dx \quad (2\text{-}8)$$

Gdje a i b predstavljaju doljnju i gornju granicu integriranja. Određeni integrali mogu integrirati i složene funkcije, a ti integrali se nazivaju konturni integrali. Za integriranje određenih integrala vrijede tri osnovna pravila kao i za neodređene integrale uz dva dodatna pravila:

$$(1) \quad \int_a^b f(x)dx = -\int_b^a f(x)dx \quad (2\text{-}9)$$

$$(2) \quad \int_a^b f(x)dx = \int_a^c f(x)dx + \int_c^b f(x)dx \quad \text{za svaki c} \in [a,b] \quad (2\text{-}10)$$

Ukoliko je funkcija f: [a,b] $\rightarrow \mathbb{R}$ omeđena i neprekidna na skupu [a,b] \ A, pri čemu je $A \subset$ [a,b] diskretan podskup, tada kažemo da je funkcija f integrabilna na intervalu [a,b]. Ukoliko je funkcija f integrabilna na intervalu [a,b] i za nju postoji primitivna funkcija F: [a,b] $\rightarrow \mathbb{R}$ tako da vrijedi $F'(x) = f(x)$ za svaki $x \in$ (a,b) tada za tu funkciju vrijedi Newton-Leibnitzova formula:

$$\int_a^b f(x)dx = F(b) - F(a) \quad (2\text{-}11)$$

2.2 Metode rješavanja neodređenih integrala

2.2.2 Metoda supstitucije

Ukoliko je integrand, odnosno funkcija koju integriramo zadana u složenom obliku:

$$\int f\big(g(x)\big)dx \quad (2\text{-}12)$$

tada za rješavanje tog integrala možemo koristiti metodu supstitucije. Ako pretpostavimo da funkcija *g* ima derivabilnu inverznu funkciju g^{-1} bar na nekom intervalu, tada možemo primjeniti zamjenu *g(x) = t*:

$$\int f\big(g(x)\big)dx = \begin{cases} g(x) = t \\ x = g^{-1}(t) \\ dx = (g^{-1})'(t) \end{cases} = \int f(t)(g^{-1})'(t)dt \quad (2\text{-}13)$$

2.2.3. Metoda parcijalne integracije

Ukoliko se podintegralna funkcija sastoji od produkta dvije raznorodne funkcije, ili se nalazi jedna ili dvije transcedentne funkcije tako da niti jednom supstitucijom ne možemo doći do rješenja, tada primjenjujemo parcijalnu integraciju. Parcijalna integracija je metoda u kojoj do rješenja dolazimo tako da zamijenimo produkt funkcija sa derivacijom prve i integralom druge funkcije i obratno. Formulu za parcijalnu integraciju dobijemo tako da prvobitno deriviramo umnožak *f(x)* · *g(x)*, nakon toga integriramo obje strane i jedan integral izrazimo na lijevoj strani.

$$[f(x)g(x)]' = f'(x)g(x) + f(x)g'(x) / \int \dots dx \quad (2\text{-}14)$$

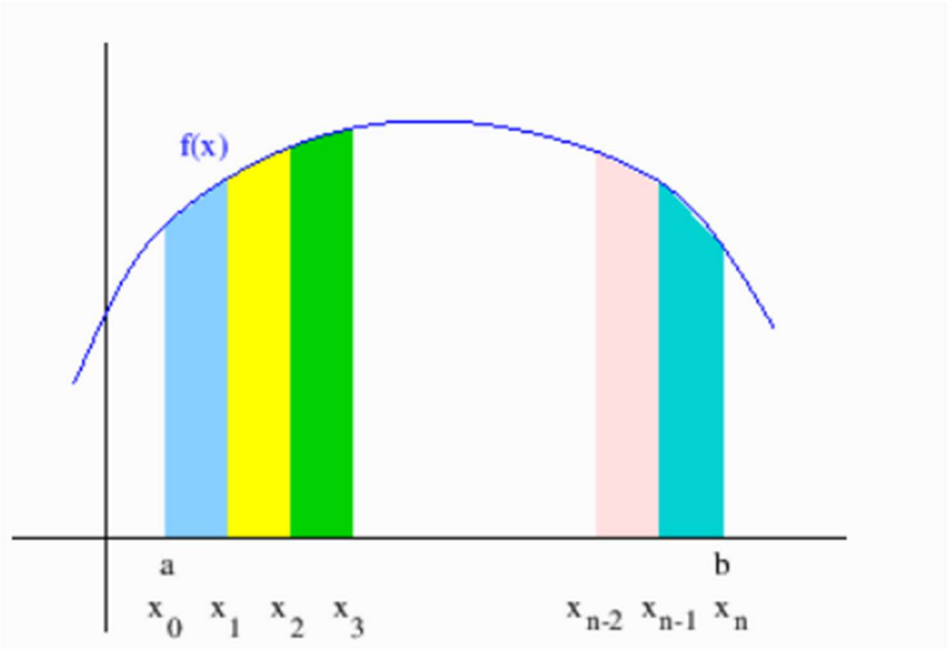
$$f(x)g(x) + C = \int f'(x)g(x)dx + \int f(x)g'(x)dx \quad (2\text{-}15)$$

$$f(x)g'(x)dx = f(x)g(x) - \int g(x)f'(x)dx \quad (2\text{-}16)$$

2.2. Metode rješavanja određenih integrala

2.2.1. Metoda trapeza

Integral pomoću metode trapeza rješavamo tako da podijelimo interval [a,b] na *n* jednakih dijelova kao što je prikazano na slici 2.2.



Slika 2.2.: Podjela površine ispod krivulje na *n* jednakih djelova

Integralnu suma je suma površina svih dobivenih trapeza. Površina *i*-tog trapeza računa se kao zbroj površine pravokutnika i površine trokuta. Izraz za trapeznu formulu glasi:

$$J_n = \sum_{i=1}^{n}(\Delta x y_{i-1} + \Delta x \frac{y_i - y_{i-1}}{2}) = \Delta x \sum_{i=1}^{n} \frac{y_{i-1} + y_i}{2} = \Delta x(\frac{y_0}{2} + \sum_{i=1}^{n-1} y_i + \frac{y_n}{2}) \quad (2\text{-}17)$$

gdje je Δx:

$$\Delta x = x_i - x_{i-1} = \frac{b-a}{n} \qquad (2\text{-}18)$$

2.2.2. Simpsonova metoda

Kod Simpsonove metode interval [a,b] dijelimo na paran broj točaka $n = 2k$. Funkciju $f(x)$ na intervalu $[x_{2i-2}, x_{2i}]$, $i = 1, ..., k$ aproksimiramo kvadratnom parabolom $p(x) = ax^2+bx+c$ koja prolazi kroz tri susjedne točke (x_{2i-2}, y_{2i-2}), (x_{2i-1}, y_{2i-1}), (x_{2i}, y_{2i}). Određeni integral parabole p(x) na intervalu $[x_{2i-2}, x_{2i}]$ računamo izrazom:

$$\int_{x2i-2}^{x2i}(ax^2+bx+c)dx = \frac{\Delta x}{3}(y_{2i-2} + 4y_{2i-1} + y_{2i}) \qquad (2\text{-}19)$$

Postavimo li ishodište koordinatnog sustava u točku x_{2i-1} dobijemo sljedeće vrijednosti:

$$x_{2i-2} = -\Delta x$$
$$x_{2i-1} = 0$$
$$x_{2i} = \Delta x$$

Tada sjecišta sa parabolom *p(x)* iznose:

$$y_{2i-2} = a\Delta x^2 - b\Delta x + c \qquad (2\text{-}20)$$
$$y_{2i-1} = c \qquad (2\text{-}21)$$
$$y_{2i} = a\Delta x^2 + b\Delta x + c \qquad (2\text{-}22)$$

Nakon što uvrstimo vrijednosti sjecišta sa parabolom *p(x)* dobijemo konačni izraz integrala:

$$\int_{x2i-2}^{x2i}(ax^2+bx+c)dx = \frac{\Delta x}{3}(2a\Delta x^2 + 6c) \qquad (2\text{-}23)$$

Konačni izraz sa Simpsonovu formulu glasi:

$$J_n = \frac{\Delta x}{3}(y_0 + 2(y_2 + y_4 + \cdots + y_{n-2}) + 4(y_1 + y_3 + \cdots + y_{n-1}) + y_n) \qquad (2\text{-}24)$$

3. KORIŠTENI PROGRAMSKI JEZICI

3.1 Python

Python je programski jezik visoke razine koji se može koristiti u razne svrhe. Jedna od područja gdje se *Python* primjenjuje su: znanost o podacima, strojno učenje, razvoj web i drugih aplikacija. Podržavaju ga razni operacijski sustavi ali se najviše koristi na *Linuxu*. Stilovi programiranja koje podržava su: objektno orijentirano, strukturno i apstraktno programiranje. *Python* ima ugrađenu provjeru indentacije koja sprječava pokretanje koda ukoliko sva udubljenja i uvlake nisu pravilno napravljeni, na taj način osigurava se pisanje čitkog koda. To je velika prednost naspram jezika kao što je npr. *JavaScript*, gdje za provjeru indentacije moramo instalirati vanjski paket. Na slici 3.1 vidimo primjer koda koji bi trebao ispisati brojeve do 10. Ovaj kod neće se izvršiti zato što indentacija nije dobro napisana, te će u terminalu biti prikazana poruka sa slike 3.2 koja govori da je indentacija krivo napisana. Iz slike se može vidjeti da je indentacija krivo napisana prvobitno na liniji 5, a zatim na liniji 9, ali program će u terminalu ispisati na kojoj je liniji posljednje očitana kriva indentacija.

```
1   parni = 0
2   neparni = 0
3   for i in range(51):
4       if i % 2 == 1:
5               neparni += 1
6       else:
7           parni += 1
8   print("Broj parnih brojeva:", parni)
9       print("Broj neparnih brojeva:", neparni)
10
```

Slika 3.1.: Programski kod sa krivo napisanom indentacijom

```
PS C:\Users\Danijel\desktop> python app.py
  File "C:\Users\Danijel\desktop\app.py", line 9
    print("Broj neparnih brojeva:", neparni)
IndentationError: unexpected indent
```

Slika 3.2.: Greška koju program prikazuje u terminalu

Slika 3.3 predstavlja isti kod, ali sa točnom indentacijom. Rezultat koji ispisuje napisani kod prikazuje slika 3.4.

```
1   parni = 0
2   neparni = 0
3   for i in range(51):
4       if i % 2 == 1:
5           neparni += 1
6       else:
7           parni += 1
8   print("Broj parnih brojeva:", parni)
9   print("Broj neparnih brojeva:", neparni)
10
```

Slika 3.3: Programski kod sa točno napisanom indentacijom

```
PS C:\Users\Danijel\desktop> python app.py
Broj parnih brojeva: 26
Broj neparnih brojeva: 25
```

Slika 3.4: Rezultat programskog koda

Pri deklariranju varijabli *parni* i *neparni*, nismo morali deklarirati koji tip podataka će varijable imati zato što *Python* nema strogo tipiziranje, odnosno deklariranje tipova podataka. Vanjski paketi se u aplikaciju implementiraju pomoću alata za instaliranje paketa *pip* (python install package). Od vanjskih paketa korišteni su:

- *Sympy* – Omogućuje *Pythonu* rad simboličkom matematikom. Dobiveni izraz tipa podatka string pretvara u simbolički izraz koji se poslije koristi za računanje.
- *Latex2Sympy* – Služi za pretvaranja *Latex* izraza u *Sympy* izraz.
- *Flask* – Povezuje backend i frontend. Služi za izvršavanje HTTP zahtjeva.
- *Numpy* – Omogućuje izvršavanje kompleksnih matematičkih operacija.

```
1  from sympy import sympify
2  from sympy.utilities.lambdify import lambdify
3
4  x = sympify('x')
5  izraz = sympify('sin(x) + x**2')
6  f = lambdify(x, izraz)
7  rezultat = f(5)
8  print(rezultat)
```

Slika 3.5.: Primjer koda sa vanjskim paketima

Slika 3.5 prikazuje ubacivanje vanjskog paketa *sympy*, te iz njega uzimamo programske funkcije *sympify* i *lambdify*. U četvrtoj liniji koda predajemo varijablu odnosno slovo koje će biti naša matematička varijabla, samu matematička funkciju predajemo u petoj liniji koda. Podatke koje smo predali programskoj funkciji *sympy* su tipa *string*, koji je običan tekst. Sa običnim teksom ne možemo realizirati izraze kao što je:

$$x = 5$$
$$f(5) = \sin(5) + 5^2 \quad (3\text{-}1)$$

Iz tog razloga moramo tekst pretvoriti u algebarski izraz koji će *Python* razumjeti i s kojim ćemo moći izvršavati računske operacije. Takav proces je vrlo kompleksan i iz tog razloga koristimo programske funkciju *sympify*. Našu matematičku funkciju smo na kraju kreirali tako da smo programskoj funkciji *lambdify* predali varijablu po kojoj želimo izvršiti matematičku funkciju i kao drugi argument predali smo samu matematičku funkciju. Nakon što je napravljena matematička funkcija što je matematička funkcija generirana, možemo joj predati određeni broj na koji želimo postaviti varijablu *x*, u ovom slučaju za primjer je korišten broj pet. Rezultat je prikazan na slici 3.6.

```
PS C:\Users\Danijel\desktop> python app.py
24.04107572533686
```

Slika 3.6.: Rezulat koda sa slike 3.5

3.1 HTML

HyperText Markup Language je prezentacijski jezik za izradu web stranica koji određuje sadržaj i funkciju određene web-stranice odnosno *HTML* dokumenta. Svaki dio stranice kao što su naslov, sadržaj, slike, poveznice i drugi sadržaj, napisani su pomoću *HTML*-a. Stranice napisane pomoću jezika *HTML* su statične i nemaju dinamičkog sadržaja. *HTML* dokument sastoji se od *HTML* elemenata koji su dodatno opisani pomoću atributa. Kako bi web stranice izgledale ljepše i uređenije, za opisivanje, odnosno uređivanje stila *HTML* elemenata, razvijen je *Cascading Style Sheets (CSS)*. *Cascading Style Sheets* je jezik pomoću kojeg *HTML* elementima dodjeljujemo određenu boju, pozadinu, oblik, animaciju i druge atribute.

3.2 JavaScript

JavaScript je skriptni jezik koji je namijenjen za izradu dinamičkih web stranica napisanih u *HTML* i *CSS* tehnologiji. Za *JavaScript* kažemo da je *client-side* jezik, to znači da nakon što se stranica učita sve funkcije i promjene mogu se izvoditi bez dodatne komunikacije sa poslužiteljem. JavaScript podržava, ali izvorno nije objektno orijentirani programski jezik. Danas se *JavaScript* osim za izradu frontend dijela također koristi i za izradu backend dijela te mnoge druge svrhe. Programski jezik *JavaSript* se izvorno može pokretati samo u pregledniku, ali kako bi to zaobišli, razvijen je alat *Node.js*, pomoću kojeg se *JavaScript* može pokretati izvan preglednika. *npm (node package manager)* je alat koji omogućuje *Node.js* – u instaliranje vanjskih paketa kao što *Python* - u omogućuje *pip*. Za bolju izradu aplikacije korištena je *JavaScript* biblioteka *React.js* koji omogućuje puno bolju, lakšu i atraktivniju izvedbu aplikacije u odnosnu na korištenje osnovnog *HTML*, *CSS* i *JavaScript* jezika. *React.js* se pokreće pomoću alata *Node.js*. Od vanjskih paketa na *frontend* dijelu aplikacije korišteni su:

- *mathlive* – Omogućuje korisniku lakši i interaktivniji način unošenja matematičkih funkcija i simbola.
- *function-plot* – Omogućuje grafički prikaz upisane matematičke funkcije.
- *react-toastify* – Služi za interaktivan ispis poruka korisniku.
- *axios* – Služi za slanje HTTP poziva.

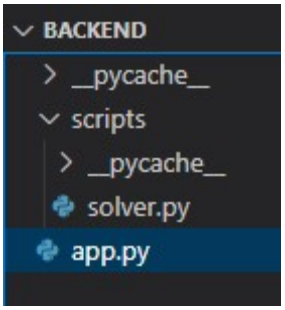
4. IZRADA PROGRAMA I VIZUALIZACIJA

4.1. Opis zadatka

Aplikacija se sastoji od dva dijela, backend i frontend. Frontend dio je zadužen za prikaz korisničkog sučelja i za interakciju sa korisnikom, odnosno omogućuje korisniku unošenje matematičke funkcije koja će se integrirati, postavljanje gornje, donje granice i na koliko podintervala će se podijeliti funkcija. Na *frontend* dijelu nema nikakvog računanja, za to je zadužen *backend*. Nakon što korisnik unese matematičku funkciju i postavi željene parametre, klikom na gumb za računanje, poslati će se poziv na *backend*, gdje se nalazi sva logika za računanje. Nakon što je na *backend* – u završeno računanje dobiveni rezultat poslati će se na *frontend*, ukoliko računanje nije dobro izvršeno poslati će se greška.

4.2. Izrada backend dijela

Kako bi mogli izvršavati matematičke operacije bilo je prvo potrebno izabrati programski jezik koji podržava kompleksnije matematičke i programske operacije. Integrirarnje je vrlo kompleksan proces, stoga je za izvršavanje računskih operacija odabran *Python*, obzirom da *JavaScript* trenutno nema razvijenu podršku za kompleksnije matematičke rroperacije. *Python* skripte ne možemo pisati u istoj datoteci u kojoj bi pisali *frontend* dio, stoga je cijeli računski dio napisan kao *backend* aplikacije, odnosno funkcionira na način da se pokreće odvojeno od *frontend* dijela aplikacije i komunicira sa istim putem HTTP zahtjeva.



Slika 4.1.: Struktura backend mape

U mapi *backend* napravili smo dvije datoteke. Prva datoteka je *app.py* u kojoj su definirane rute za određeni HTTP poziv i u njoj čitamo podatke dobivene sa *fronteda*. U mapi *scripts* napravljvena je datoteka *solver.py* koja sadrži programske funkcije za rješavanje integrala.

4.2.1 Rute i upravljanje podacima

Kako bi mogli koristiti napisani kod na *backend* – u potrebno ga je pokrenuti i postaviti na određeni URL. Za pokretanje *backend* dijela korišten je paket *Flask* i naredba *flask run* koja pokreće i postavlja *backend* na URL *http://localhost:5000* i omogućuje *frontend* dijelu pristup rutama definiranim na *backend* dijelu. *Frontend* će biti pokrenut isto na *localhost* URL-u, ali na portu 3000, iz tog razloga moramo koristiti *CORS (Cross-Origin Resource Sharing)* kako bi mogli slati zahtjeve i podatke između *frontend*a i *backenda*. Na slici 4.2 prikazano je *flask, flask_cors* biblioteka i njihovih klasa.

```
6   from flask import Flask,request
7   from flask_cors import CORS
8   app = Flask(__name__)
9   CORS(app)
```

Slika 4.2.: Implementiranje *flask, flask_cors* biblioteka i njihovih klasa

Slika 4.3 prikazuje definiranje rute sa naredbom *@app.route*. Kao prvi argument postavljamo URL rute kojoj pristupamo, u ovom slučaju to je */solve-with-all*, odnosno kada budemo pristupali ruti upisati ćemo *http://localhost:5000/solve-with-all*. Drugi argument su metode, odnosno vrste HTTP

zahtjeva pomoću kojih možemo pristupiti ruti. Obzirom da će sve naše rute primati određene podatke u zahtjevu, potrebno je koristiti POST metodu. Nakon rute, definirana je programska funkcija koja će se izvršiti nakon dolaska na rutu. Programska funkcija ne mora imati isti naziv kao i ruta, ali u ovom slučaju nazvana je istim nazivom radi jednostavnosti koda.

```
39  @app.route('/solve-with-all', methods=['POST'])
40  def solve_with_all():
41      data = request.get_json()
```

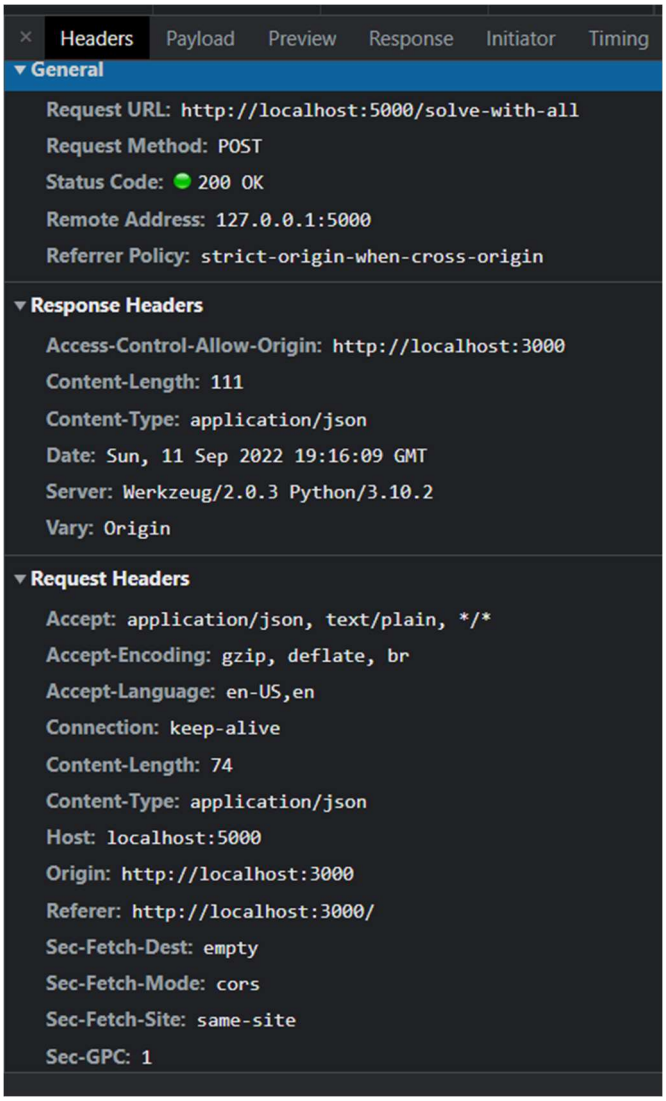
Slika 4.3.: Definiranje rute i primanje podataka

Osim klase *Flask* implementirali smo i klasu *request* koja sadrži sve informacije i podatke o zahtjevu poslanog sa frontenda. Kako bi mogli koristiti primljene podatke pozivamo *get_json* metodu klase *request* koja vraća podatke u *JSON (JavaScript Object Notation)* formatu i sprema ih u varijablu *data*. Varijabla *data* sada predstavlja objekt sa svim podacima koji su primljeni u zahtjevu. Iz objekta, odnosno varijable *data*, podatke uzimamo na način da napišemo *data* i zatim u uglate zagrade upišemo podatak koji želimo učitati.

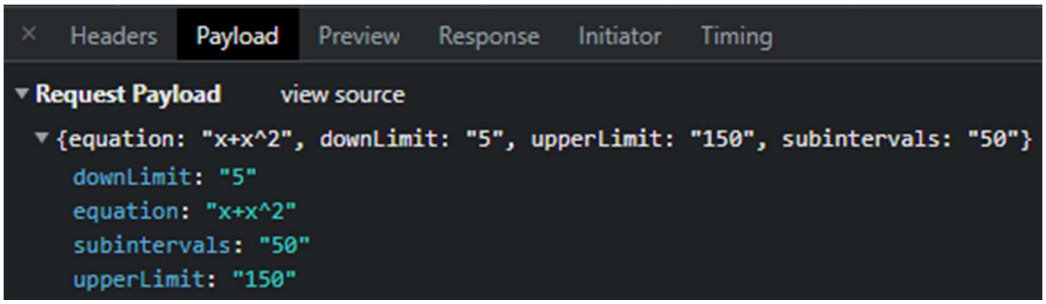
```
{'equation': 'x+x^2', 'downLimit': '5', 'upperLimit': '150', 'subintervals': '50'}
```

Slika 4.4.: Primjer podataka upisanih u varijablu *data*

Nazivi podataka su prethodno definirani na *frontend* - u, ali cijeli zhatjev zajedno sa podatcima također možemo vidjeti u pregledniku ukoliko odemo u *Developer Tools* i otvorimo *Network* karticu.



Slika 4.5.: Prikaz zahtjeva u Network kartici



Slika 4.6.: Prikaz primljenih podataka u Network kartici

Nakon izvršavanja zadanih operacija pozivamo ključnu riječ *return* i zatim ime varijable u kojoj su podaci koje želimo poslati nazad na *fronted.*

4.2.2 Pretvaranje podataka i formatiranje integralne funkcije

Svaki primljeni podatak potrebno je spremiti u zasebnu varijablu i pretvoriti ga u željeni tip podataka kako bi ga poslije mogli koristiti. Svi podatci koji su primljeni sa *frontenda* su tipa *String*. Izraz koji će se integrirati šalje se u *Latex* formatu koji je opisan u poglavlju 4.3.3.
Latex izraz formatiran je u *Sympy* izraz pomoću programske funkcije *latex2sympy*. Ostali podatci pretvoreni su u tip *Integer* obzirom da su to brojevi koji će biti korišteni pri integriranju. Varijable *upperLimit* i *downLimit* predstavljaju gornju i doljnju granicu, a varijabla *subintervals* predstavlja broj podintervala na koje će se matematička funkcija podijeliti pri integriranju.

```
equation = latex2sympy(data['equation'])
upper_limit = int(data['upperLimit'])
down_limit = int(data['downLimit'])
subintervals = int(data['subintervals'])
```

Slika 4.7.: Stvaranje željenih varijabli iz primljenih podataka

U poglavlju 3.1 objašnjeni su izrazi *sympify* i *lambdify*. Na slici 4.8 prikazana je njihova primjena u našoj aplikaciji. Varijabla *integrate_by* predstavlja matematičku varijablu po kojoj će se vršiti integracija, zatim konačnu matematičku funkciju stvaramo sa programskom funkcijom *lambdify*. Kao prvi argument predajemo joj varijablu po kojoj će se matematička funkcija integrirati, a kao drugi argument predajemo izraz prethodno formatiran sa *latex2sypmy* programskom funkcijom, odnosno dobivenu matematičku funkciju koju ćemo integrirati.

```
integrate_by = sympify('x')
f = lambdify(integrate_by, equation)
```

Slika 4.8.: Kreiranje konačne funkcije za integriranje

4.2.3 Izrada funkcije za integriraciju pomoću metode trapeza

U datoteci *solver.py* definirali smo programsku funkciju *trapz* sa slike 4.11. Kao argumente prima matematičku funkciju *f* koju je potrebno integrirati, *down_limit* i *upper_limit,* odnosno donju i gornju granicu, te broj *n* koji predstavlja količinu podintervala na koje će biti podijeljena matematička funkcija.

```
distance = upper_limit - down_limit
subinterval = down_limit
x = []
x.append(subinterval)
i = 0
while i < n:
    subinterval = subinterval + distance/n
    x.append(subinterval)
    i += 1
y = []
```

Slika 4.8.: Računanje udaljenosti i postavljanje vrijednosti podintervala

Na početku programske funkcije *trapz* udaljenost između dvije granice dobivena je kao razlika izeđu gornje i doljnje granice. Varijablu *subinterval* postavili smo na vrijednost donje granice i uvečavali ju za vrijednost omjera udaljenosti i broja *n*, te svaku novu vrijednost varijable dodali u listu *x* sve dok vrijednost varijable *i* ne bude jednaka broju *n*. Lista *x* predstavlja udaljenost između gornje i donje granice podjeljenu na *n* podintervala.

```
[0, 0.5, 1.0, 1.5, 2.0, 2.5, 3.0, 3.5, 4.0, 4.5, 5.0]
```

Slika 4.9.: Vrijednost liste *x* za vrijednosti *down_limit = 0*, *upper_limit = 5* i *n = 10*

Za svaku vrijednost u listi *x* izračunata je vrijednost matematičke funkcije *f* i svaki rezultat je dodan u novu listu *y.* Ukoliko matematička funkcija ima prekid ili nije definirana na određenom intervalu, program će izbaciti grešku.

```
y = []
for interval in x:
    try:
        if isinstance(f(round(interval)), float or int) or isinstance(f(round(interval)), int):
            y.append(f(interval))
    except:
        error = {
            'error': 'Funkcija nije definirana ili ima prekid u točki ' + str(int(interval)) + '.', 'isCommon': True}
        return error
```

Slika 4.10.: Računanje vrijednosti integrala za svaki pojedini interval

```
[0, 0.75, 2.0, 3.75, 6.0, 8.75, 12.0, 15.75, 20.0, 24.75, 30.0]
```

Slika 4.11.: Vrijednost liste *y* za vrijednost liste *x* sa slike 4.8 i vrijednost $f = x + x^2$

Za realizaciju izraza IZRAZ definirali smo dvije nove liste:

- *y_right_endpoint* - Sazdržava svaki element osim prvog elementa liste *y*, u izrazu 2-17 predstavlja y_i.
- *y_left _endpoint* - Predstavlja y_{i-1} u izrazu 2-17 i sadržava sve osim zadnjeg elementa liste *y*.

Varijabla *dx* predstavlja Δx i dobivena je kao omjer razlike gornje i doljnje granice sa količinom podintervala. Konačni rezultat dobiven je kao umnožak varijable *dx* podijeljene s dva i zbroja liste *y_right_endpoint* i *y_left _endpoint*.

```
y_right_endpoint = y[1:]
y_left_endpoint = y[:-1]
right_endpoint_sum = 0
left_endpoint_sum = 0
for interval_result in y_right_endpoint:
    right_endpoint_sum += interval_result
for interval_result in y_left_endpoint:
    left_endpoint_sum += interval_result
endpoint_sum = right_endpoint_sum + left_endpoint_sum
dx = (upper_limit - down_limit)/n
result = (dx/2) * endpoint_sum
return str(result)
```

Slika 4.12.: Konačne liste i kod za integriranje po metodi Trapeza

4.2.4 Izrada funkcije za integriraciju pomoću Simpsonove metode

Programska funkcija za računanje pomoću Simpsonove metode definirana je pod nazivom *simps*. Podijela podintervala i računanje vrijednosti za svaki interval jednako je napravljeno kao i kod *trapz* programske funkcije. Za realizaciju Simpsonove metode, odnosno izraza IZRAZ, potrebno je podijeliti listu *y* na tri nove liste:

- *y_first_endpoints* - Sadržava prvi i svaki drugi element liste y, u izrazu 2-19 predstavlja y_{2i-2}.
- *y_second_endpoints* – Sadržava svaki drugi element liste y, u izrazu 2-19 predstavlja y_{2i-1}.
- *y_third_endpoints* – Sadržava svaki drugi element liste y, u izrazu 2-19 predstavlja y_{2i}.

Varijabla *dx* dobivena je na isti način kao i kod programske funckije *trapz*, a konačni rezultat dobiven je kao umnožak varijable *dx* podijeljene s tri i sume sve tri liste. Isto kao i kod *trapz* programske funkcije, ukoliko matematička funkcija nije definira ili ima prekid na određenom intervalu program će izbaciti grešku. Program će također izbaciti grešku ukoliko je poslan neparan broj podintervala, obzirom da je za računanje sa Simpsonovom metodom potreban paran broj podintervala.

```
y_first_endpoints = y[0:-1:2]
y_second_endpoints = y[1::2]
y_third_endpoints = y[2::2]
first_endpoints_sum = 0
second_endpoints_sum = 0
third_endpoints_sum = 0
for interval_result in y_first_endpoints:
    first_endpoints_sum += interval_result
for interval_result in y_second_endpoints:
    second_endpoints_sum += 4*interval_result
for interval_result in y_third_endpoints:
    third_endpoints_sum += interval_result
endpoint_sum = first_endpoints_sum + second_endpoints_sum + third_endpoints_sum
result = dx/3 * endpoint_sum
return str(result)
```

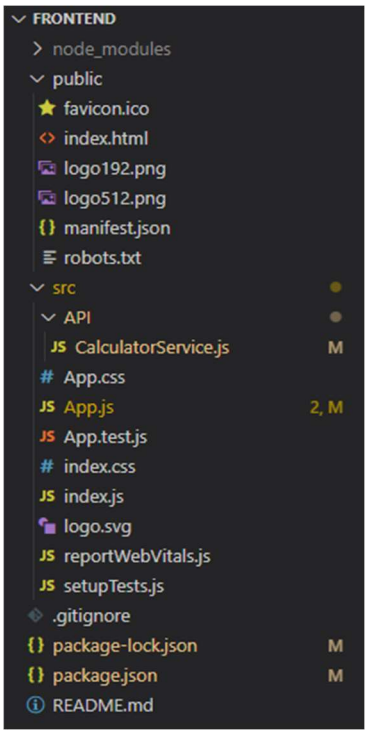
Slika 4.13.: Konačne liste i kod za integriranje po Simpsonovoj

4.3. Izrada frontend dijela

4.3.1 Struktura frontend mape

Pomoću naredbe *npx create-react-app* kreirane su početne mape i datoteke *frontend* dijela aplikacije. *Frontend* aplikacije sastoji se od sljedećih dijelova:

- *node_modules* – Mapa koja sadržava sve instalirane *Node.js* pakete koji su posebno instalirani i one koji su potrebni za pokretanje *React.js* projeka.
- *public* – Mapa koja sadržava slike ili ikone koje su korištene u projektu.
- *src* – Glavna mapa u projektu koja sadrži skripte za pokretanje aplikacije, datoteke koje čine aplikaciju i datoteke u kojima stilizramo elemente aplikacije.
- *.gitignore* – Ukoliko želimo postaviti aplikaciju na platformu *GitHub,* u datoteku *.gitignore* ćemo upisati imena datoteka koje ne želimo postaviti na platformu.
- *package.json* – Datoteka u kojoj se nalaze imena i verzije *npm* paketa korištenih u projektu.



Slika 4.14.: Struktura *frontend* mape

4.3.2 Povezivanje backend i frontend dijela, asinkronizacija

U mapi *src* stvorili smo novu mapu *API*, te kreirali *JavaScript* datoteku *CalculatorService.js.*

```
import axios from "axios";
const baseUrl = 'http://localhost:5000'

class CalculatorService {
    async solveWithAll(equation, downLimit, upperLimit, subintervals) {
        const response = await axios.post(baseUrl + '/solve-with-all', {equation, downLimit, upperLimit, subintervals})
        return response.data
    }

    async convertToExpression(equation) {
        const response = await axios.post(baseUrl + '/convert-expression', {equation})
        return response.data
    }
}

export default new CalculatorService();
```

Slika 4.15.: Sadržaj datoteke *CalculatorService.js*

U datoteku smo prvo ubacili *npm* paket *axios* i definirali varijablu *baseURL* u kojoj se nalazi URL na kojem je pokrenut *backend* aplikacije. Unutar datoteke definirana je klasa *CalculatorService* čija se instanca stvara i izvozi na dnu datoteke. Klasa *CalculatorService* sastoji se od dvije metode. Svaka od

metoda prima određene parametre koje proslijeđuje na *backend* slanjem HTTP POST pomoću paketa *axios*. Nakon izvršavanja poziva kao povratni podatak dobit ćemo objekt sa svojstvom *data* u kojem se nalaze rezultati računskih ili drugih operacija izvršenih na *backend* dijelu aplikacije. Kako bi dobivene podatke mogli koristiti dalje u kodu potrebno je pozvati naredbu *return response.data*. Za izvršavanje HTTP zahtjeva potrebno je duže vrijeme u odnosu na izvršavanje drugih operacija unutar koda. Kako bi osigurali primranje podataka koristimo naredbe *async/await*. Naredbu *async* stavljamo ispred naziva metode odnosno programske funkcije kako bi naznačili da je ona asinkrona, a naredbu *await* ispred linije u kojoj se šalje HTTP zahtjev.

4.3.3 Izrada korisničkog sučelja

Programski kod za korisničko sučelje nalazi se u datoteci *App.js*. HTML elementi napisani unutar *App.js* datoteke stilizirani su pomoću koda napisnog u datoteci *App.css*. Obzirom da korisnik mora imati mogućnost unošenja gornje granice, donje granice, matematičke funkcije i broja podintervala potrebno je definirati *state* za svaki od navedenih parametara. Ukoliko se vrijednost varijable promjeni dio stranice na kojem se koristi varijabla neće se ponovno učitati sa novom vrijednosti varijable, iz tog razloga naše varijable definirano kao *state*.

```
const [equation, setEquation] = useState('')
const [resultTrapez, setResultTrapez] = useState('')
const [resultSimpsons, setResultSimpsons] = useState('')
const [upperLimit, setUpperLimit] = useState(0)
const [downLimit, setDownLimit] = useState(0)
const [subintervals, setSubintervals] = useState(0)
const [expression, setExpression] = useState(null)
```

Slika 4.16.: Definiranje *statea*

Na slici 4.15 prikazano je definiranje *state* – a za svaki parametar koji korisnik treba moći unijeti. Prvi parametar u uglatim zagradama predstavlja varijablu koju će mijenjati programska funkcija definirana u drugom parametru. Nakon znaka jednakosti pozvana je programska funkcija *useState* kojoj predajemo željenu početnu vrijednost varijable. Ukoliko ne pozovemo programsku funkciju *useState* program neće prepoznati da vrijednosti u uglatim zagradama predstavljaju *state*. *HTML* elementi za unos gornje granice, donje granice i podintervala definirani su kao *<input />* element kojem

predajemo minimalnu vrijednost varijable, ime *CSS* klase koja opisuje stil elementa, tip podatka koji treba primati i koju programsku funkciju treba aktivirati nakon što korisnik unese vrijednost.

```
<input
  placeholder="1"
  min={1}
  type="number"
  className='input--number'
  onChange={e => setUpperLimit(e.target.value)}
/>
```

Slika 4.17.: Definiranje *HTML* elementa za unos gornje granice

```
.input--number {
    border: 2px solid black;
    height: 20px;
}
```

Slika 4.18.: *CSS* klasa koja opisuje obrub i visinu elementa sa slike 4.16

Kako bi korisnik unutar preglednika imao tipkovnicu sa matematičkim simbolima potrebnim za upis željene matematičke funkcije korištena je biblioteka *mathlive*. Iz biblioteke *mathlive* uzeli smo komponentu *math-field* koja omogućuje prikaz tipkovnice sa matematičkim simbolima i formatira unesene znakove tako da u korisničkom sučelju izgledaju kao matematička funkcija.

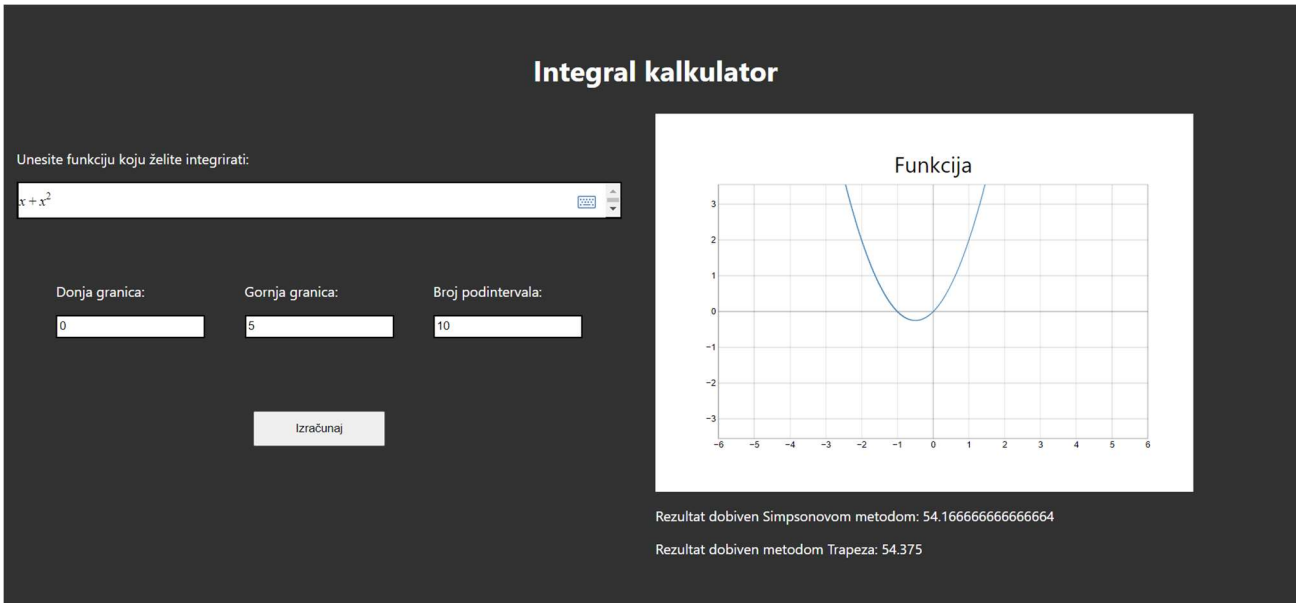
```
x+\frac{x^2}{\sin\left(x\right)}+\sqrt{20}
```

Slika 4.19.: Matematička funkcija u pozadini

Slika 4.20.: Matematička funkcija na korisničkom sučelju

Matematička funkcija sa slike 4.17 napisana je u *Latex* formatu. Graf matematičke funkcije stvara se pomoću programske funckije *functionPlot* iz biblioteke *function-plot*, ali ona ne podržava

matematičke funkcije napisane u *Latex* formatu stoga moramo formatirati trenutno napisanu matematičku funkciju u drugi format. Ukoliko je došlo do pogreške pri izvršavanju zahtjeva aktivirati će se *toast* iz paketa *react-toastify*, te će se u gornjem desnom kutu prikazati notifikacija sa tekstom do koje je pogreške došlo prilikom izvršavanja zahtjeva.



Slika 4.21.: Korisničko sučelje

Pritiskom na gumb *'Izračunaj'* pozvati će se metode iz datoteke *CalculatorService.js*. Metoda *convertToExpression* vratit će unesenu matematičku funkciju formatiranu iz *Latex* formata u obični teksutalni format, a metoda *solveWithAll* vratit će rješenja integriranja matematičke funkcije pomoću Simpsonove i metode Trapeza za trenutne parametre. Nakon primanja podataka u koordinatnom sustavu nacrtat će se graf funkcije, dok će ispod koordinatnog sustava biti ispisana rješenja integriranja.

4.3.4 Testiranje aplikacije

Kako bi testirali rad i točnost aplikacije napravili smo iste funkcije za računanje, ali pomoću vanjske biblioteke *numpy*.

```python
def trapz_numpy(f, down_limit, upper_limit, n=50):
    x = np.linspace(down_limit, upper_limit, n+1)
    y = f(x)
    y_right_endpoint = y[1:]
    y_left_endopint = y[:-1]
    dx = (upper_limit - down_limit)/n
    result = (dx/2) * np.sum(y_right_endpoint + y_left_endopint)
    return str(result)
```

Slika 4.22.: Programska funkcija za rješavanje integrala pomoću metode Trapeza

Na slici 4.20 prikazana je programska funckija za računanje integrala pomoću metode Trapeza napisana pomoću vanjske biblioteke *numpy*. Iz slike možemo vidjeti da je funkcija puno kraća zbog programskih funkcija *linespace* i *sum* koje su definirane u biblioteci *numpy*. Programska funkcija *linespace* korištena je umjesto koda na slici 4.8, a programska funkcija *sum* korištena je umjesto koda sa slike 4.11. Matematičke funkcije smo također riješili nekom od metoda za rješavanje neodređenih integrala i uvrstili granice kako bi usporedili rezultat sa rezultatom dobivenim numeričkim metodama.

Funkcija	Donja granica	Gornja granica	Broj podintervala
$\sin(x) + 5x$	5	25	50
$\frac{1}{x}$	-3	10	24
$\frac{x + 2x^2 + 4x^3}{1 + x + x^4}$	7	40	150
$x + x^2$	1	10	11

Tablica 4.1.: Funkcije i parametri unešeni za testiranje

Metoda Trapeza	Simpsonova metoda	Metoda Trapeza sa *Numpy* bibliotekom	Simpsonova metoda sa *Numpy* bibliotekom	Metoda za rješavanje neodređenih integrala
1499.301918501839595	1499.2923567961395	1499.3019185018397	1499.2923567961395	1505.150141
greška	greška	0.5305750513013507	1.5878947530481295	greška
7.213328992976177	7.21296314125602	7.21332899297617	7.212963141256019	7.212962979
383.5041322314051	greška	383.504132231405	greška	382.5

Tablica 4.2.: Rezultati dobiveni integriranjem pomoću navedenih metoda

Iz tablice 4.2 možemo primjetiti ukoliko u programske funkcije napisane pomoću *numpy* biblioteke unesemo matematičku funkciju koja ima prekid na jednoj od točaka u unesenom intervalu svejedno nećemo dobiti grešku. U ovom slučaju matematička funkcija $\frac{1}{x}$ nije definirana u točki 0, ali s obzirom da se funkcija dijeli na podintervale moguće je da će točka intervala iznositi 0.08 umjesto 0, iz tog razloga program neće vratiti pogrešku. U programskoj funkciji koju smo napisali bez *numpy* biblioteke napravili smo provjeru koja će poslati grešku u navedenom slučaju. Obje funkcije za rješavanje pomoću Simpsonove metode poslati će grešku ukoliko predamo neparan broj podintervala. Između ostalih riješenja nema velikih oscilacija.

5. ZAKLJUČAK

Pretvaranje matematičkog izraza iz teksutalnog tipa podatka u matematičku funkciju koju program razumije predstavlja najveći problem realizacije koda za računanje integrala pomoću Simpsonove metode i metode trapeza. Formule navedenih metoda su realizirane koristeći osnovne programske funkcije *Pythona*. Programske funkcije koje smo pisali bez *Numpy* biblioteke su puno duže, ali omogućavuju nam upravljanje sa svakim rješenjem funkcije i na taj način osiguravaju izbacivanje greške u slučaju unosa intervala na kojem matematička funkcija nije definirana ili u slučaju unosa nepravog integrala. Iako smo uspjeli omogućiti provjeru definicije funkcije u određenoj točki, nismo uspjeli napraviti provjeru konvergira li određeni integral ili divergira. Budući da JavaScript ima iznimno razvijenu podršku za vizualizaciju sadržaja, uspjeli smo napraviti vizualizaciju matematičke funkcije kako bi korisnik mogao vidjeti na kojem je području funkcija definirana i ima li funkcija prekida.

6. LITERATURA

[1] J. G. Brookshear, D. Brylow, Računalna znanost : pregled, Zagreb : Dobar plan, 2016.

[2] G. Crespo, Responzivni Web dizajn uz jQuery : optimizirajte tehnike responzivnog Web dizajna koristeći jQuery, Dobar plan, Zagreb, 2015

[3] Y. Fain, Programiranje Java, Dobar plan, Zagreb, 2011.

[4] P. Gasston, Moderni Web : responzivni Web dizajn uz HTML5, CSS3 i JavaScript, Dobar plan, Zagreb, 2013.

[5] G. Halfacree, BBC micro:bit : službeni priručnik, Dobar plan, Zagreb, 2018.

[6] Z. Hercigonja, Online alati za validaciju web stranica, https://hrcak.srce.hr/clanak/322520, 7.9.2022.

[7] I. Kniewald, Python : osnove programiranja u programskom jeziku Python, Udžbenik.hr, Zagreb, 2016.

[8] M. Miler. Python, Uvod u programiranje za inženjere, https://hrcak.srce.hr/clanak/77785, 7.9.2022.

[9] M. Pilgrim, HTML 5 spreman za upotrebu, Dobar plan, Zagreb, 2010.

[10] V. Sanjković, B. Trstenjak, Visual Studio - univerzalni alat za razvoj aplikacija, https://hrcak.srce.hr/clanak/106234

[11] S. Srbljić , D. Škvorc , M. Popović, Krajnjem korisniku prilagođeni programski jezici za poosobljavanje računalom upravljanih okolina, https://hrcak.srce.hr/clanak/132198, 7.9.2022.

[12] M. D. Stojanović, V. S. Aćimović-Raspopović, Savremene IP mreže : arhitekture, tehnologije i protokoli, Akademska misao, Beograd, 2012.

[13] B. Trogrlić, online editori za web programiranje, https://hrcak.srce.hr/clanak/302144, 7.9.2022.

Sažetak

U radu je opisano rješavanje određenih integrala metodom Trapeza i Simpsonovom metodom putem web aplikacije. Web aplikacija sastoji se od *frontend* i *backend* dijela, odnosno korisničkog sučelja i programirane logike za integriranje. Matematička funkcija za integriranje i parametri unositi će se putem korisničkog sučelja. Opisani su programski jezici i alati korišteni za izradu korisničkog sučelja i programskih funkcija za integriranje. Programske funkcije za integriranje napisane su na dva načina radi uspoređivanja rezultata i točnosti. Opisane su situacije u kojima program ne može izračunati integral. Za izbjegavanje unošenja nepravih integrala i granica u kojima matematička funkcija nije definirana napravljen je vizualni prikaz funkcije u koordinatnom sustavu kako bi korisnik mogao vidjeti u kojem je području funkcija definirana.

Ključne riječi: Python, JavaScript, HTML, integral, Simpsonova metoda, metoda Trapeza

Abstract

The paper describes solving certain integrals using the Trapezoid method and the Simpson method via web application. The web application consists of frontend and backend part, that is, a user interface and programmed logic for integration. The mathematical function for integration and parameters will be entered through the user interface. The programming languages and tools used to create the user interface and programming functions for integration are described. The integration functions are written in two ways in purpose of comparing results and accuracy. Situations in which the program cannot calculate the integral are described. To avoid entering improper integrals and limits where the mathematical function is not defined, a visual display of the function in the coordinate system was made so that the user could see in which area the function is defined.

Keywords: Python, JavaScript, HTML, integral, Simpsons method, Trapez method