ЗАТВЕРДЖЕНО
Вченою радою ЗУІ
Протокол №2 від „28”лютого 2024 р.
Ф-КДМ-3

Міністерство освіти і науки України

Закарпатський угорський інститут ім. Ференца Ракоці ІІ

Кафедра математики та інформатики

Реєстраційний №_____

Кваліфікаційна робота

ОБЧИСЛЕННЯ ВИЗНАЧЕНИХ ІНТЕГРАЛІВ МЕТОДОМ МОНТЕ-КАРЛО

КІРАЛЬ ЄВА ТІБЕРІЇВНА

Студентка IV-го курсу

Освітня програма «Середня освіта (Математика)»

Спеціальність 014 «Середня освіта (Математика)»

Рівень вищої освіти: бакалавр

Тема затверджена на засіданні кафедри

Протокол № 3 / 2024

Науковий керівник: **___Головач Йожеф Ігнацович**

(*доктор технічних наук, професор*)

Завідувач кафедрою математики та інформатики: **Кучінка Каталін Йожефівна**

(к. ф.-м. н, доцент)

Робота захищена на оцінку _____, «___» _____ 202_ року

Протокол № _____ / 202_

ЗАТВЕРДЖЕНО
Вченою радою ЗУІ
Протокол №2 від „28”лютого 2024 р.
Ф-КДМ-3

Міністерство освіти і науки України
Закарпатський угорський інститут ім. Ференца Ракоці II

Кафедра математики та інформатики

Кваліфікаційна робота

ОБЧИСЛЕННЯ ВИЗНАЧЕНИХ ІНТЕГРАЛІВ МЕТОДОМ МОНТЕ-КАРЛО

Рівень вищої освіти: бакалавр

Виконавець: студентка IV-го курсу

Кіраль Єва Тіберіївна

освітня програма «Середня освіта (Математика)»

спеціальність 014 «Середня освіта (Математика)»

Науковий керівник: _____**Головач Йожеф Ігнацович**

(*доктор технічних наук, професор*)

Рецензент: _____ **Стойка Мирослав Вікторович**

(канд. фіз.-мат. наук, доцент)

Берегове
2025

Зміст

ЗАТВЕРДЖЕНО
Вченою радою ЗУІ
Протокол №2 від „28”лютого 2024 р.
Ф-КДМ-3

II. Rákóczi Ferenc Kárpátaljai Magyar Főiskola

Matematika és Informatika Tanszék

HATÁROZOTT INTEGRÁLOK KISZÁMÍTÁSA MONTE-CARLO MÓDSZERREL

Szakdolgozat

Készítette: Király Éva

IV. évfolyamos matematika

szakos hallgató

Témavezető: Holovács József

(műszaki tudományok doktora, professzor)

Recenzens: Sztojka Miroszláv

(fizika és matematika tudományok kandidátusa, docens)

Beregszász – 2025

Tartalomjegyzék

1. fejezet

Bevezetés

1.1. A Monte-Carlo módszer

A Monte-Carlo módszer egy olyan szimulációs módszer, amely a valószínűségszámítás és statisztika elemeit használva numerikusan értékeli ki az eredményt [4, 2]. A módszer a véletlenszerű mintavételen alapul, ennek segítségével pedig elég nagy elemű minta esetén meg lehet becsülni határozott integrálok értékét [3, 16]. Használható továbbá kockázati faktorok becslésére a gazdasági életben [17], illetve számos becsléshez, például a π értékének becsléséhez is [6].

A sztochasztikus szimulációk egy szorosan kapcsolódó kifejezés, ami ugyanazt jelenti, mint a Monte-Carlo szimuláció – nem mellesleg sokkal leíróbb is –, azonban a Monte-Carlo elnevezés szélesebb körben használatos [5].

1.2. Története

A Monte-Carlo módszert egy, a szerencsejátékban fontos helyről, a Monacói Monte-Carlóról nevezték el, mivel a véletlen és véletlenszerűség központi szerepet tölt be a technikában, ahogyan az olyan játékokban is, mint például a rulett vagy a játékgépek [6].

A véletlenszerűség használata egy tudományos probléma megválaszolása során régebbről ered, mint a számítógépek, annak ellenére, hogy a Monte-Carlo módszer a számítógépek fejlesztése idején kezdett kibontakozni. A módszer születése egész régre, az 1700-as évekre vezethető vissza. Szorosan összekapcsolható Buffon nevével, és a π értékének meghatározása körüli problémával [4, 7].

A szimuláció mögött meghúzódó ötlet 1944-ben kapta nevét, felhasználása ekkor kapott jelentős szerepet a kutatásokban. Stanislaw Ulam matematikus fejlesztette ki az első atomfegyver megalkotása során. Ötletét munkatársával, akivel a Manhattan Projecten dolgozott, Neumann Jánossal is megosztotta, akivel együttműködtek a módszer fejlesztésében [1].

A projecten dolgozó tudósok nehéz egyenletekkel dolgoztak annak meghatározására, hogy egy hasadó uránatom neutronja milyen valószínűséggel idézi elő egy másik hasadást. Ezen egyenleteknek vissza kellett adniuk a bomba bonyolult geometriáját, a válasznak pedig helyesnek kellett lennie, mivel ha a kísérlet nem sikerült, hónapokba telt újra elegendő uránhoz jutni az újabb kísérlethez. A meglátás az volt, hogy a szimulált pályák statisztikai tulajdonságai megegyeznek a valós neutronpályákkal, így megbízható válaszokat adhat a problémájukra, csak megfelelő számú pályát kellett szimulálniuk [1].

A Neumann János és Stanislaw Ulam által megalkotott elképzelések szisztematikus kidolgo-

zásával Harris és Herman Kahn 1948-ban foglalkozott. 1948 körül Fermi, Metropolis és Ulam Monte-Carlo becsléseket készítettek a Schrödinger-egyenlet sajátértékeire [1, 4].

1970 körül, az újonnan kialakuló számítási elméletek meggyőző okot adtak a Monte-Carlo módszer alkalmazására. Az elmélet segítségével be tudták azonosítani a problémák egy olyan osztályát, ahol egy probléma pontos megoldására fordított idő exponenciálisan növekszik M-mel. A kérdés az volt, hogy a Monte-Carlo módszer meg tudja-e becsülni a probléma megoldását egy meghatározott statisztikai pontosságon belül. Ezen állítást számos példa támasztja alá [6, 10, 15].

2. fejezet

Numerikus integrálás

Ha egy adott *f* függvény egy $[a, b]$ intervallumon integrálható, és ezen az intervallumon van primitív függvénye ($F' = f$), akkor a függvény határozott integrálját a Newton-Leibniz formula segítségével a következő módon lehet megadni:

$$\int_a^b f(x)dx = F(b) - F(a).$$

Azonban ez a képlet nem alkalmazható közvetlenül minden esetben [8, 10].

A Newton-Leibniz formula nem alkalmazható, ha magát az *f* függvényt nem ismerjük. Ez akkor fordulhat elő, ha az *f* függvény értékei mérésekből származó eredmények. Abban az esetben sem tudjuk alkalmazni, ha az *f* függvény primitív függvényét nem tudjuk megadni zárt alakban. Ebben az esetben a függvény primitívjét vagy nem tudjuk, vagy nem akarjuk explicit módon meghatározni. A $(sinx)/x$ vagy e^{-x^2} függvényeknek például nem tudjuk megadni a primitív függvényeit zárt alakban, annak ellenére, hogy léteznek, hiszen a függvények folytonosak [8, 10, 13].

Vannak esetek, amikor az integrál pontos kiszámítása sokkal komplikáltabb lenne, mint mondani egy jó közelítést. Például a számítógépes program esetében gyakran könnyebb egy jó becslést adni egy függvény adott intervallumon vett integráljára, mint valamilyen kompjuteralgebrai módszerrel szimbolikusan előállítani a függvény primitívjét, és arra alkalmazni a Newton-Leibniz-formulát. Egy tipikus példa erre a differenciálegyenletek megoldása, ahol a megoldás formális előállítása nehéz, ellenben egy numerikus közelítés könnyebben megadható.

Általában, ha egy függvény primitívjét nem tudjuk zárt alakban előállítani, akkor valamilyen hatványsor formájában megadható. Ebben az esetben a hatványsor részletösszegét helyettesítjük be a Newton-Leibniz-képletbe, és annak integrálásával adjuk meg az integrál közelítését.

2.1. Példa. *Határozzuk meg az* $\int_0^1 e^{-x^2}dx$ *integrált* 10^{-6}*-nál kisebb hibával! Mivel*

$$e^x = 1 + x + \frac{x^2}{2!} - \frac{x^3}{3!} + ...,$$

ezért

$$e^{-x^2} = 1 - x^2 + \frac{x^4}{2!} - \frac{x^6}{3!} \pm ...,$$

ez a sor integrálható tagonként, így

$$\int_0^1 e^{-x^2}dx = \Big[x - \frac{x^3}{3} + \frac{x^5}{5 \cdot 2!} - \frac{x^7}{7 \cdot 3!} \pm ...\Big]_0^1 = 1 - \frac{1}{3} + \frac{1}{10} - \frac{1}{42} \pm ... = \sum_{k=0}^{\infty} \frac{(-1)^k}{(2k+1)k!}.$$

Mivel egy Leibniz-sor adja az integrál értékét, az első $k-1$ tag összege 10^{-6}-nál jobban közelíti a sor összegét, ha annak k-adik tagja kisebb 10^{-6}-nál. Ha $k = 9$, akkor

$$\frac{(-1)^k}{(2k+1)k!} = \frac{1}{6894720} < 10^{-6},$$

így

$$\sum_{k=0}^{8} = \frac{1098032417}{1470268800} \approx 0.74682426573970691618$$

így egy megfelelő közelítése az integrál értékének [10].

A fenti módszer a gyakorlatban általában csak néhány konkrét integrál meghatározására alkalmazható. Egy másik lehetőség, ha az integrált a közelítő összegek határértékeként definiáljuk, egy jól megválasztott közelítő összeg megfelelő közelítését tudja megadni a tényleges integrálnak. Ez az úgynevezett interpolációs típusú integrálási módszerek egyike [10].

2.2. Példa. *Legyen adott a következő határozott integrál:*

$$\int_0^1 cos(x)dx = [sinx]_0^1 = sin1.$$

A Newton-Leibniz-tétel alkalmazása nem okoz problémát, azonban ez esetben a végeredményt csak a szinusz függvény $x = 1$ helyen vett Taylor-sorának egy részösszegével tudjuk becsülni [10]:

$$sin1 = 1 - \frac{1}{3!} + \frac{1}{5!} - \frac{1}{7!} \pm ...$$

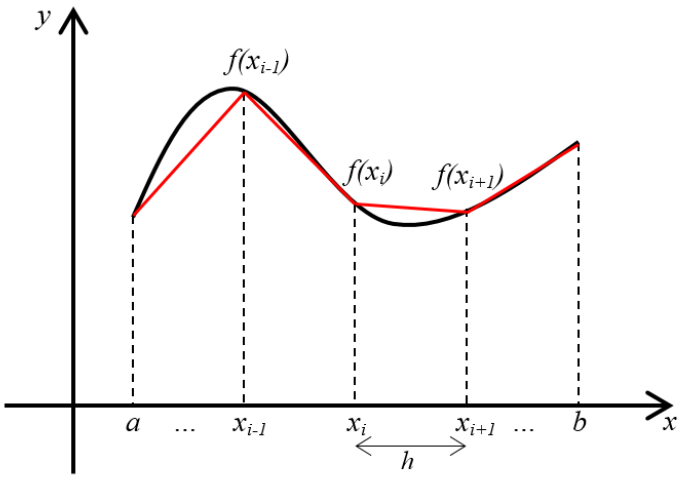
A példa alapján látható, hogy sok integrál esetén lényeges különbség nem állítható fel aközött, hogy a határozott integrál értékének meghatározásakor, vagy pedig a kapott érték számszerű megadásánál használunk közelítést [8, 10].

Numerikus integrálásnak azt az eljárást nevezzük, amikor valamilyen módszerrel közelítjük az integrál értékét[8, 10, 15]. Ezek az eljárások úgynevezett interpolációs módszerek, ilyen például a trapéz-, érintő- és Simpson-formulák, illetve a Gauss-féle integrálformulák [9].

Többdimenziós függvények esetén a numerikus módszerek nehezen programozhatók, illetve egyre lassabban konvergálnak. A véletlen számok generálásán alapuló módszerek az egyszerűbb esetekben lassabban konvergálnak, azonban az összetettebb, többváltozós integrálok esetén is könnyen implementálhatók és konvergencia tulajdonságaikat megtartják [8, 1, 4].

2.1. Trapéz szabály

A trapéz szabály az egyik legismertebb módszer, mely lényege, hogy a két pontot összekötő szakasz alatti trapéz területével becsüljük az adott intervallumon az integrál értékét. Diszkrét pontokra és analitikusan megadott függvényekre vonatkozóan is alkalmazható, ha szimbolikusan nem tudjuk az integrált meghatározni. Utóbbi esetben $n-1$ részre osztva az intervallumot felveszünk n pontot az integrálni kívánt függvényszakaszon, és a kapott pontokra értelmezzük a trapéz szabályt.



2.1. ábra. Trapéz módszer

Egyetlen intervallum esetén:

$$\int_a^b f(x)dx \approx \frac{f(a)+f(b)}{2}(b-a).$$

Több intervallum esetén a felosztás által kapott trapézok területének értékét összegezzük. Ha $N = n-1$ részintervallumunk van, akkor:

$$\int_a^b f(x)dx \approx \frac{1}{2}\sum_{i=1}^{N}(f(x_i)+f(x_{i+1}))(x_{i+1}-x^i).$$

Ez a képlet abban az esetben is működik, ha az intervallumok nem egyenlő hosszúságúak. Ha azonban egyenlő szakaszokra osszuk az adott intervallumot, és azok hossza $h = (x_2 - x_1) = (x_3 - x_2) = ... = (x_n - x_{n-1})$, akkor a szabály egyszerűsíthető, mégpedig a következőképpen:

$$\int_a^b f(x)dx \approx \frac{h}{2}\sum_{i=1}^{N}(f(x_i)+f(x_{i+1})).$$

A fejezet tartalmi része és a képletek a(z) [11] forrás alapján kerültek bemutatásra.

2.2. Simpson-módszer

Míg a trapéz szabály szomszédos pontok közötti egyenesekkel közelíti meg az integrál értékét, más módszerek magasabb rendű, ám könnyen integrálható függvények alkalmazásával teszik ugyanezt. A Simpson-formula az egyik legismertebb ilyen módszer, melynek lényege, hogy másod- vagy harmadfokú polinomokat illeszt a pontokra, és így közelíti a függvényszakaszokat.

Másodfokú Simpson-formula 3 szomszédos pontra illeszt parabolát. Ezt megadhatjuk úgy, hogy a Newton-féle interpolációs polinomot felírjuk 3 pontra vonatkozóan:

$$p(x) = a_1 + a_2(x-x_1) + a_3(x-x_1)(x-x_2).$$

Ha az intervallumot különböző hosszúságú részintervallumokra bontjuk, az együtthatókat a következő módon határozhatjuk meg:

$$a_1 = y_1; \quad a_2 = \frac{y_2-y_1}{x_2-x_1}; \quad a_3 = \frac{\frac{y_3-y_2}{x_3-x_2} - \frac{y_2-y_1}{x_2-x_1}}{x_3-x_1}.$$

Egyenlő h hosszúságú részintervallumokra:

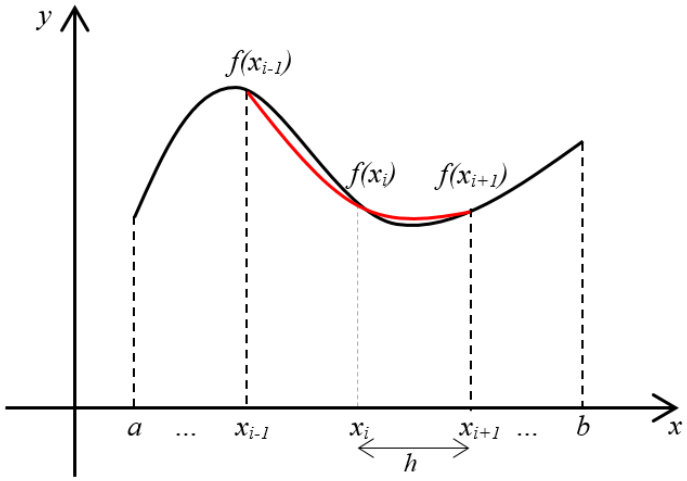
$$a_1 = y_1; \quad a_2 = \frac{y_2 - y_1}{h}; \quad a_3 = \frac{y_3 - 2y_2 + y_1}{2h^2}.$$

Az együtthatókat visszahelyettesítve a polinomba, 3 pontra levezetve a következő képletet kapjuk:

$$\int_{x_1}^{x_3} f(x)dx \approx \int_{x_1}^{x_3} p(x)dx = \frac{h}{3}(f(x_1) + 4f(x_2) + f(x_3)).$$

Általánosan így alkalmazzuk:

$$\int_{x_{i-1}}^{x_{i+1}} f(x)dx \approx \frac{h}{3}(f(x_{i-1}) + 4f(x_i) + f(x_{i+1})).$$



2.2. ábra. Simpson-módszer

Vegyünk n pontot egyenletesen egy adott $[a, b]$ intervallumon, egymástól h távolságra. Legyen $x_1 = a, x_n = b$. Az intervallumot így n pont $N = n - 1$ részintervallumra bontja. A Simpson-szabályhoz 3 pont szükséges, hogy parabolát tudjunk illeszteni, illetve mindig két egymást követő szakaszra kell a függvényt felbontanunk, hogy alkalmazhassuk:

$$\int_a^b f(x)dx = \int_{x_1=a}^{x_3} f(x)dx + \int_{x_3}^{x_5} f(x)dx + ... + \int_{x_{n-2}}^{x_n=b} f(x)dx = \sum_{i=2,4,6...}^{N} \int_{x_{i-1}}^{x_{i+1}} f(x)dx.$$

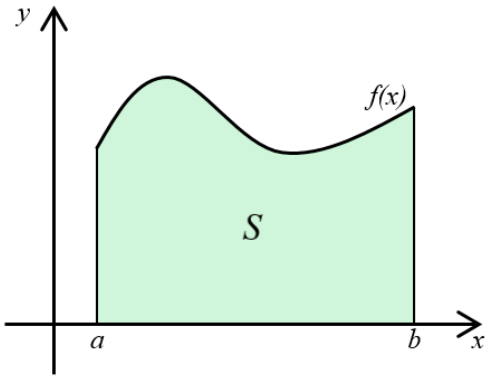
A három pontra kapott egyenlet felhasználásával, n pontra megkaphatjuk a következő formulát:

$$\int_a^b f(x)dx \approx \frac{h}{3}\left[f(a) + 4\sum_{i=2,4,6...}^{n-1} f(x_i) + 2\sum_{j=3,5,7...}^{n-2} f(x_j) + f(b)\right].$$

A fejezet tartalmi része és a képletek a(z) [11] forrás alapján kerültek bemutatásra.

2.3. Több dimenziós integrálok

Egy függvény integráljának meghatározása egy dimenzióban azt jelenti, hogy meghatározzuk egy tetszőleges ív alatti S terület értékét.



2.3. ábra. Függvény integrálja egy változó esetén

A legegyszerűbben ezt a szabályos intervallumonként vett N pont feletti közvetlen összegzés segítségével érhetjük el:

$$S = \sum_{i=1}^{N} f(x_i)\Delta x, \tag{2.1}$$

ahol x_i az $[x_{i-1}, x_i]$ intervallum középpontja:

$$x_i = a + (i - 0.5)\Delta x \quad and \quad \Delta x = \frac{b-a}{N} \tag{2.2}$$

Az (2.1)-es egyenlet átírható a következő módon:

$$S = \frac{b-a}{N} \sum_{i=1}^{N} f(x_i)$$

M dimenzió esetén, az intervallum általánosítva $([a_1, b_1], [a_2, b_2], ..., [a_M, b_M])$, az integrál pedig egy $(M+1)$ dimenziós $V^{(M+1)}$ térfogatot ad:

$$V^{(M+1)} = \frac{(b_1 - a_1)(b_2 - a_2) \cdots (b_M - a_M)}{N_1 N_2 \cdots N_M} \sum_{i_1=1}^{N_1} \sum_{i_2=1}^{N_2} \cdots \sum_{i_M=1}^{N_M} f(x_i),$$

ahol $x_i = (x_{i_1}, x_{i_2}, ..., x_{i_M})$ egy M dimenziós vektor, és minden x_i a (2.2)-es megadás alapján van definiálva.

A fenti egyenlet átírható a következőképpen:

$$V^{(M+1)} = \frac{V^{(M)}}{N} \sum_{i_1=1}^{N_1} \sum_{i_2=1}^{N_2} \cdots \sum_{i_M=1}^{N_M} f(x_i) = V^{(M)} \frac{\sum_{i_1=1}^{N_1} \sum_{i_2=1}^{N_2} \cdots \sum_{i_M=1}^{N_M} f(x_i)}{N}, \tag{2.3}$$

ahol $V^{(M)}$ egy M dimenziós térfogat, az a "terület", ahol a függvény, N pedig a pontok teljes száma. Az egyenlet tört részét értelmezhetjük mint f feletti átlag a kérdéses intervallumon, így az egyenlet az alábbi módon is felírható:

$$V^{(M+1)} = V^{(M)} \langle f \rangle, \tag{2.4}$$

ahol $\langle f \rangle$ a következő átlagot jelöli:

$$\langle f \rangle = \frac{\sum_{i=1}^{N} f(x_i)}{N}.$$

A fejezet képletei és tartalmi része a [2] forrás alapján kerültek bemutatásra.

2.4. Kettős integrálok közelítése trapéz szabállyal és Simpson-módszerrel

A trapéz szabályt és a Simpson-módszert egyaránt lehet alkalmazni kettős integrálok közelítésére is [13, 15]. Először vizsgáljuk meg, hogyan működik a trapéz szabály. Legyen adott:

$$\iint\limits_R f(x,y)dxdy,$$

ahol $R = \{(x,y)|a \leq x \leq b, c \leq y \leq d\}$, a, b, c és d konstansok adják azt a négyszöget, téglalapot, ami felett integrálni szeretnénk az f függvényt. Az integrálást először végezzük el y, majd pedig x szerint:

$$\iint\limits_R f(x,y)dxdy = \int_a^b \left(\int_c^d f(x,y)dy \right) dx,$$

Legyen $k = (d-c)/2$ és $h = (a-b)/2$, és bontsuk mindkét intervallumot két részintervallumra. Alkalmazzuk a trapéz szabályt először a belső integrálra $[c;d]$ intervallumon[13, 15]:

$$\int_c^d f(x,y)dy \approx \frac{k}{2}\left[f(x,c) + f(x,d) + 2f\left(x, \frac{c+d}{2}\right)\right]$$

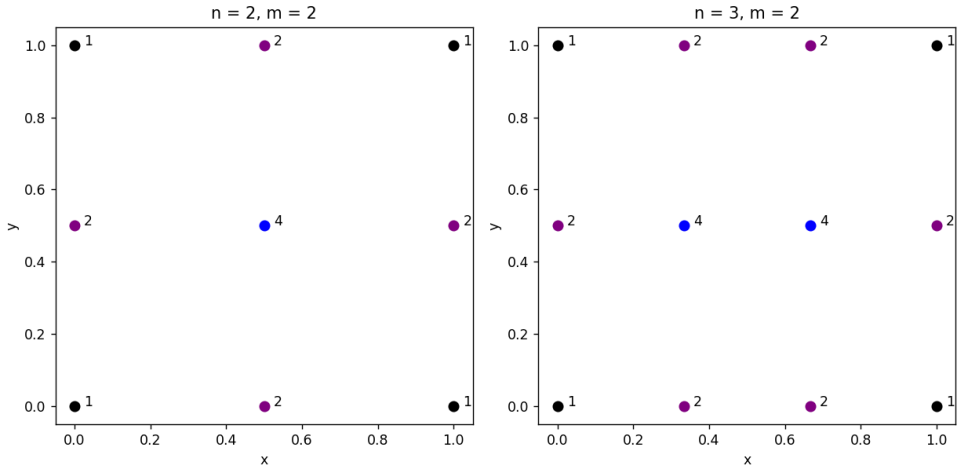
Az integrálunk most a következőképpen néz ki:

$$\int_a^b \left(\int_c^d f(x,y)dy \right) dx \approx \int_a^b \frac{d-c}{4}\left[f(x,c) + f(x,d) + 2f\left(x, \frac{c+d}{2}\right)\right] dx$$

Ha erre alkalmazzuk újra a trapéz szabályt, a következő alakhoz jutunk:

$$\begin{aligned}
\int_a^b \left(\int_c^d f(x,y)\, dy \right) dx &\approx \int_a^b \frac{d-c}{4}\left[f(x,c) + f(x,d) + 2f\left(x, \frac{c+d}{2}\right)\right] dx \\
&= \frac{hk}{4}\Bigg[f(a,c) + f(a,d) + f(b,c) + f(b,d) \\
&\quad + 2\left(f\left(\frac{a+b}{2}, c\right) + f\left(\frac{a+b}{2}, d\right) + f\left(a, \frac{c+d}{2}\right) + f\left(b, \frac{c+d}{2}\right)\right) \\
&\quad + 4f\left(\frac{a+b}{2}, \frac{c+d}{2}\right)\Bigg]
\end{aligned}$$

A formula lényegében azt is mondja és mutatja be, hogy a közelítő összegben az intervallumokra való bontás után a sarkon lévő pontokat 1-es, a szélen lévő, azonban nem sarok pontokat 2-es, a belső pontokat 4-es szorzóval veszi.



2.4. ábra. Pontok súlya a trapéz módszer kettős integrálok közelítésére való alkalmazásakor

A módszer úgy is működik, ha a két intervallumot különböző hosszúságú részintervallumokra bontjuk.

Általánosan, ha az $[a, b]$ intervallumot n, a $[c, d]$ intervallumot pedig m részintervallumra bontjuk [15]:

$$\iint_{[a,b]\times[c,d]} f(x,y)dxdy \approx \frac{hk}{4}\sum_{i=0}^{n}\sum_{j=0}^{m} w_{ij} f(x_i, y_j),$$

ahol $h = (b-a)/n$, $k = (d-c)/m$, $x_0 = a, x_n = b, y_0 = c, y_n = d$ illetve:

$$w_{ij} = \begin{cases} 1, \text{ ha } (i = 0 \text{ vagy } i = n) \text{ és } (j = 0 \text{ vagy } j = n) \\ 2, \text{ ha } (i = 0 \text{ vagy } i = n) \text{ vagy } (j = 0 \text{ vagy } j = n) \\ 4, \text{ egyébként} \end{cases}$$

Alkalmazzuk a Simpson-módszert az f függvényre, hasonlóan, mint a trapéz módszernél, először a $[c, d]$ tartományt osszuk 2 részre és alkalmazzuk a Simpson-módszert [15]:

$$\int_c^d f(x,y)dy \approx \frac{k}{3}\left[f(x,c) + 4f\left(x, \frac{c+d}{2}\right) + f(x,d)\right],$$

ahol $k = (d-c)/2$ Ezt követően alkalmazzuk újra a Simpson-módszert, az $[a, b]$ tartományt két részre osztva:

$$\begin{aligned}\int_a^b \left(\int_c^d f(x,y)\,dy\right)dx &\approx \int_a^b \frac{k}{3}\left[f(x,c) + f(x,d) + 4f\left(x, \frac{c+d}{2}\right)\right]dx \\ &= \frac{hk}{9}\Bigg[f(a,c) + f(a,d) + f(b,c) + f(b,d) \\ &\quad + 4\left(f\left(\frac{a+b}{2}, c\right) + f\left(\frac{a+b}{2}, d\right) + f\left(a, \frac{c+d}{2}\right) + f\left(b, \frac{c+d}{2}\right)\right) \\ &\quad + 16f\left(\frac{a+b}{2}, \frac{c+d}{2}\right)\Bigg],\end{aligned}$$

ahol $h = (b-a)/2$.

Most írjuk fel a képletet általánosan[15]. Először írjuk fel a

$$\int_c^d f(x,y)dy$$

integrálra alkalmazva, ahol a $[c,d]$ intervallumot m részintervallumra osztjuk, illetve $k = (d - c)/m$, $y_0 = c, y_m = d$:

$$\int_c^d f(x,y)dy \approx \frac{k}{3}\left[f(x,y_0) + 4\sum_{j=1,3,5\ldots}^{m-1} f(x,y_j) + 2\sum_{j=2,4,6\ldots}^{m-2} f(x,y_j) + f(x,y_m)\right].$$

Az $[a,b]$ intervallumot osszuk fel n részre, ekkor $h = (b-a)/n$, $x_0 = a$, $x_n = b$:

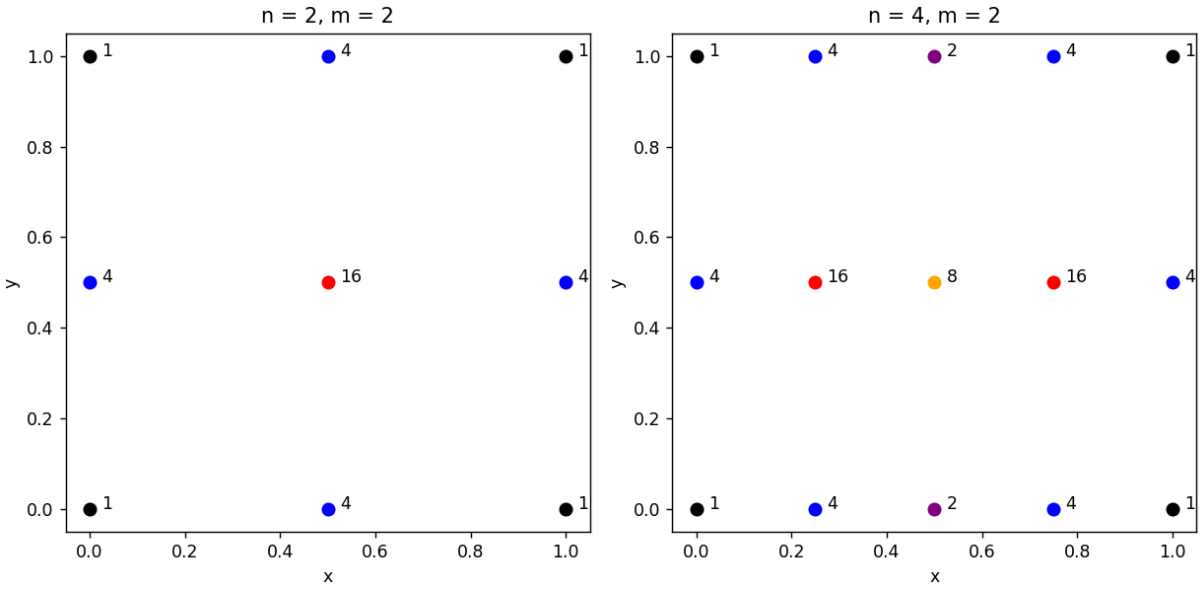
$$\begin{aligned}
\int_a^b \left(\int_c^d f(x,y)\,dy\right)dx \approx \frac{k}{3}\Bigg[&\int_a^b f(x,y_0)dx + 4\sum_{j=1,3,5\ldots}^{m-1}\int_a^b f(x,y_j)dx + \\
&2\sum_{j=2,4,6\ldots}^{m-2}\int_a^b f(x,y_j)dx + \int_a^b f(x,y_m)dx\Bigg] \\
= \frac{hk}{9}\Bigg[&f(x_0,y_0) + 2\sum_{i=2,4,6\ldots}^{n-2} f(x_i,y_0) + 4\sum_{i=1,3,5\ldots}^{n-1} f(x_i,y_0) + f(x_n,y_0)\Bigg] \\
&+ 2\Bigg[\sum_{j=2,4,6\ldots}^{m-2} f(x_0,y_j) + 2\sum_{i=2,4,6\ldots}^{n-2}\sum_{j=2,4,6\ldots}^{m-2} f(x_i,y_j) \\
&+ 4\sum_{i=1,3,5\ldots}^{n-1}\sum_{j=1,3,5\ldots}^{m-1} f(x_i,y_j) + \sum_{j=2,4,6\ldots}^{m-2} f(x_n,y_j)\Bigg] \\
&+ 4\Bigg[\sum_{j=1,3,5\ldots}^{m-1} f(x_0,y_j) + 2\sum_{i=2,4,6\ldots}^{n-2}\sum_{j=1,3,5\ldots}^{m-1} f(x_i,y_j) \\
&+ 4\sum_{i=1,3,5\ldots}^{n-1}\sum_{j=1,3,5\ldots}^{m-1} f(x_i,y_j) + \sum_{j=1,3,5\ldots}^{m-1} f(x_n,y_j)\Bigg] \\
&\Bigg[f(x_0,y_m) + 2\sum_{i=2,4,6\ldots}^{n-2} f(x_i,y_m) + 4\sum_{i=1,3,5\ldots}^{n-1} f(x_i,y_m) + f(x_n,y_m)\Bigg]
\end{aligned}$$

Általánosan felírva:

$$\iint_{[a,b]\times[c,d]} f(x,y)dxdy \approx \frac{hk}{9}\sum_{i=0}^{n}\sum_{j=0}^{m} w_{ij} f(x_i,y_j),$$

ahol a súlyok annak függvényében változnak, hogy hol helyezkednek el a pontok rácshálón:

$$w_{ij} = \begin{cases}
1 & \text{ha } (i = 0 \text{ vagy } i = n) \text{ és } (j = 0 \text{ vagy } j = n) \\
2 & \text{ha } (i = 0 \text{ vagy } i = n \text{ és } 0 < j < n \text{ páros}) \text{ vagy} \\
 & (j = 0 \text{ vagy } j = n \text{ és } 0 < i < n \text{ páros}) \\
4 & \text{ha } (i = 0 \text{ vagy } i = n \text{ és } 0 < j < n \text{ páratlan}) \text{ vagy} \\
 & (j = 0 \text{ vagy } j = n \text{ és } 0 < i < n \text{ páratlan}) \\
8 & \text{ha } 0 < i < n, 0 < j < n \text{ és } i + j \text{ páratlan} \\
16 & \text{ha } 0 < i < n, 0 < j < n \text{ és } i, j \text{ páros.}
\end{cases}$$



2.5. ábra. Pontok súlya a Simpson-módszer kettős integrálok közelítésére való alkalmazásakor

3. fejezet

Integrálás a Monte-Carlo módszerrel

3.1. Mintavételezés

A mintavételezés módszere a Monte-Carlo integrálás során igen hasonló a fentebb kapott (2.1)-es és (2.3)-as egyenletekhez. Rendszeres Δx intervallumonkénti mintavétel helyett vegyünk véletlenszerű pontokat és alkalmazzuk ezen pontok átlagolását [2].

Tegyük fel, hogy kiválasztunk N darab x_i pontot [a, b] intervallumon 1 dimenzióban. Az integrál a következő:

$$S = \frac{b-a}{N} \sum_{i=1}^{N} f(x_i),$$

ami azonos a (2.1)-es egyenlettel.

Általánosítva, M dimenzió esetén

$$x_i = (x_1, x_2, ..., x_M)$$

vektorokat kell választanunk az $([a_1, b_1], [a_2, b_2], ..., [a_M, b_M])$ intervallumon, ami egyszerű a dimenziók egységes véletlen számainak egyidejű használatával. Ha van N ilyen pontunk, akkor a Monte-Carlo becslés az $(M + 1)$ dimenziós térfogathoz M dimenziós függvény felett

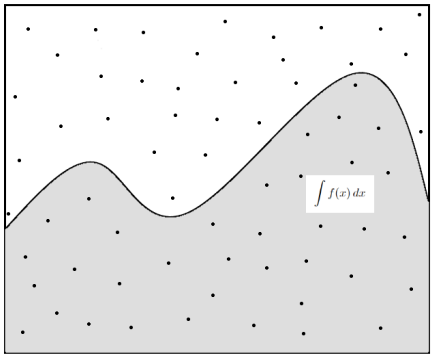
$$V^{(M+1)} \approx V^{(M)} \frac{\sum_{i=1}^{N} f(x_i)}{N} = V^{(M)} \langle f \rangle. \tag{3.1}$$

A (2.3) és (3.1) egyenleteket összehasonlítva könnyen észrevehető a két módszer közötti alapvető különbség: numerikus integrálás esetén szükségünk van M darab különböző összegre, míg a Monte-Carlo integrálás során elegendő egyetlen összeget kiszámolni [13, 15]. Nincs nagy különbség a kettő között 1 dimenzió esetén, könnyen lehet, hogy numerikusan sokkal könnyebb és pontosabb megoldást ad, mint a Monte-Carlo integrál. Azonban a dimenzió növelésével M összeg kiszámítása elég körülményessé válik, azonban a Monte-Carlo becslés, amely során továbbra is csak egy összeg kiszámítására van szükség, sokkal egyszerűbb [2, 3, 1].

3.2. Elutasítási módzser

Létezik egy másik megközelítése is a Monte-Carlo integrálnak, úgy nevezett elutasítási vagy "elfogadás-elutasítás" módszer, ami egyszerűbb, mint a mintavételezés. Lényegében ugyanaz, mint a Neumann-féle elutasítási módszer, amely nem egyenletes eloszlás felett generált véletlen számokat használ [2].

Tegyük fel, hogy egy ismert "térfogatú" rész magába foglalja az általunk keresett integrált. Generáljunk véletlenszerűen pontokat mindenhol ebben az ismert térben és nézzük meg, melyek azok, amelyek eltalálták a kérdéses teret [2].



3.1. ábra. Az elutasítási módszer ábrázolása

A külső rész térfogatát jelölje V_k, f_t pedig a találatok arányát. Ebben az esetben az integrál meghatározása igen egyszerűen írható fel [2]:

$$V = V_k f_t.$$

Tegyük fel, hogy M dimenzióban dolgozunk, a próbapontok száma N, a találatok száma pedig N_t. Ekkor a fenti egyenlet átírható a következő képpen [2]:

$$V = V_k f = V_k \frac{N_t}{N} = V_k \frac{\sum_{i=1}^{N_t} 1}{N}.$$

Ha definiálunk egy M dimenziós függvényt, hogy

$$f(x) = \begin{cases} 1, \text{ ha x a keresett térfogaton belül van} \\ 0, \text{ minden más esetben} \end{cases}$$

akkor az egyenlet úgy fog kinézni, hogy

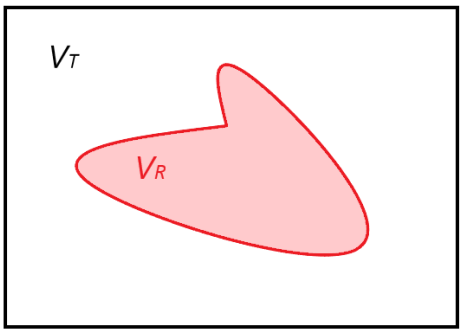
$$V = V_k \frac{\sum_{i=1}^{N} f(x_i)}{N}.$$

Ez pedig megegyezik a (3.1)-s egyenlettel, a különbség annyi, hogy az f dimenzió nélküli, a V és V_k dimenzói pedig megegyeznek. A valóságban az f átlaga a találatok f_t arányt adja, felírhatjuk az egyenletet olyan formában, hogy

$$V = V_k f_t = V_k \langle f \rangle. \tag{3.2}$$

3.3. Szabálytalan idomok területének kiszámítása

Szabálytalan idomok területének meghatározására többféle lehetőségünk is van, a Monte-Carlo módszer előnye azonban abban rejlik, hogy különféle esetekre általánosan alkalmazható [11].



3.2. ábra. Szabálytalan idom

Legyen adva egy $f(x)$ a $x \in V_T$ tartományon értelmezve, és keressük a függvény $V_R \subset V_T$ résztartományon vett határozott integráljának értékét. Az integrál a következő: $\int_{V_R} f(x)dV$ [11].

Integrál alkalmazásával a kérdéses tartományon a függvény átlagértékét így tudjuk megadni:

$$\overline{f}_V = \frac{1}{V_R} \int_{V_R} f(x)dV$$

A V_T tartományon N pont felvételével, a függvényt kiértékelve a pontokban kiszámíthatjuk a függvényértékek átlagát. Ez elég nagy N esetén közelítőleg az integrál alkalmazásával meghatározott értékkel egyezik meg:

$$\overline{f}_V \approx \frac{1}{n} \sum_{i=1}^{n} f(x_i)$$

ahol $x_i \subset V_R$, n a tartományba eső pontok száma [11].

A kapott kifejezéseket egyenlővé téve az integrál a következőképpen fejezhető ki:

$$\int_{V_R} f(x)dV = \approx \frac{V_R}{n} \sum_{i=1}^{n} f(x_i)$$

A V_R tartomány közelítő értékét véletlenszerűen felvett pontokkal is meghatározhatjuk. Ha a pontok egyenletes eloszlást követnek, akkor elég sok pont felvétele után a keresett tartományonban lévő pontok és az összes pont számának aránya a V_R résztartomány nagysága a teljes V_T tartományhoz:

$$\frac{V_R}{V_T} = \frac{n}{N}, \quad V_R = \frac{V_T \cdot n}{N},$$

ahol n a V_R tartományba eső összes pont száma, N az összes pont száma [11]. A keresett integrál közelítő értéke a következő:

$$\int_V f(x)dV \approx \frac{\frac{V_T \cdot n}{N}}{n} \sum_{i=1}^{n} f(x_i) = \frac{V_T}{N} \sum_{i=1}^{n} f(x_i)$$

Tehát a keresett integrál becsülhető a tartományon belül lévő pontokon kiszámolt függvényértékek összegének és a teljes tartomány nagysága/az összes pont arányának szorzatával. A $\frac{V_T}{N}$ ez esetben az egy pontra eső terület/térfogat nagyságát határozza meg[11].

3.4. Hibaelemzés

3.4.1. A mintavételezés hibaelemzése

A mintavételezés hibáját egyszerűen meg lehet kapni. Az integrál értékének kiszámításához az előbbiekben az f függvény feletti $\langle f \rangle$ átlagot számoltunk. Ebből következik, hogy a módszer hibája megegyezik az átlag hibájával a mintákban vett f függvény értékei felett [2, 4].

Az átlag hibájának $\sigma_{\overline{x}}$ meghatározására [2, 3, 15] használt képlet meghatározott x_i pontokon

$$\sigma_{\overline{x}} \approx \frac{\sigma}{\sqrt{N}},$$

ahol a σ^2 szórásnégyzetet a következő egyenletből kapjuk:

$$\sigma^2 = \frac{1}{N-1}\left[\left(\sum_{i=1}^{N} x_i^2\right) - N(\overline{x})^2\right].$$

Ez az egyenlet abban az esetben igaz, ha az adatok szimmetrikusak a 0 körül. Ha ez nem teljesül, akkor:

$$\sigma^2 = \frac{1}{N-1}\left(\sum_{i=1}^{N}(x_i - \overline{x})^2\right).$$

Figyelembe véve a fenti egyenleteket és feltételezve, hogy $N >> 1$, a következő képen írható fel a hiba értéke:

$$\sigma_{\overline{x}} \approx \frac{1}{\sqrt{N}}\sqrt{\frac{\sum_{i=1}^{N} x_i^2}{N} - (\overline{x})^2}.$$

A Monte-Carlo integrál esetében az x_i pontok az f függvény értékei. Használva az átlag jelöléseit:

$$\langle f \rangle = \frac{\sum_{i=1}^{N} f(x_i)}{N} \quad \text{és} \quad \langle f^2 \rangle = \frac{\sum_{i=1}^{N} f^2(x_i)}{N},$$

a Monte-Carlo integrál hibája felírható úgy, mint

$$\sigma_{V\langle f \rangle} \approx V\sqrt{\frac{\langle f^2 \rangle - \langle f \rangle^2}{N}},$$

így a Monte-Carlo integrál egyenlete a következő [2, 3]:

$$\int f dV \approx V\langle f \rangle \pm V\sqrt{\frac{\langle f^2 \rangle - \langle f \rangle^2}{N}}. \tag{3.3}$$

Ez adja az úgynevezett 1 σ hibát. Ez azt jelenti, hogy ha az adatok Gauss-eloszlásban oszlanak el, a lehetősége annak, hogy a valódi érték hibája 1 σ-n belül van, 2/3. Minél szélesebb konfidenciaintervallumot veszünk, 2 σ, 3 σ, stb., annál biztosabb, hogy a valódi érték benne lesz a mért érték hiba környezetében.

Az érték pontossága az $\sqrt{N}$ növekedésével együtt nő, és nem függ a dimenziótól. Ez is azt mutatja, hogy miért érdemes a Monte-Carlo integrál használata [2].

A hibabecsléssel kapcsolatban 2 probléma merülhet fel. Az első lehetséges gond az, hogy semmi nem garantálja, hogy a pontok Gauss-eloszlásban oszlanak el, ezért a (3.3)-es egyenlet által megadott hiba csak hozzávetőleges elképzelésként értelmezhető arra vonatkozóan, hogy milyen szintű bizonytalanság várható [3].

Jobban meg tudjuk becsülni a hiba értékét, ha elegendő $f(x_i)$ pont összegyűjtésével megvizsgáljuk, hogy milyen alakú az eloszlás, és elemezzük az eloszlás hibaértékét.

A második lehetséges probléma, hogyha az adatok Gauss-eloszlásúak is, a (3.3)-es egyenlet nem ad jó megoldást alacsony N esetén. Ha kevés adatpontunk van, az eloszlás átlagának becslése pontatlanná válik, az eltérést és az átlagot összehasonlító egyenlettel együtt.

Egy pontosabb becslés a helytálló hiba értékére, ha aszórás meghatározása során a nevezőben N helyett $(N-1)$-el számolunk:

$$\sigma_{V\langle f\rangle} \approx V\sqrt{\frac{\sum_{i=1}(f_i - \langle f\rangle)^2}{N-1}}.$$

Ezek után nem lehet jobb becslést adni a mögöttes adateloszlás ismerete nélkül. Ha ez ismert, szimulálható a különbség.

Sajnos az f függvény alakja sok esetben ismeretlen, a kiértékelése pedig olyan lassú, hogy csak néhány ponton mintavételezhető, ezért nem lehet meghatározni, milyen eloszlása van. Ilyen esetekben gyakori a (3.3)-es egyenlet használata, annak reményében, hogy célra vezet.

3.4.2. Az elutasítási módszer hibaelemzése

Az elutasítási módszer az integrál értékét a következőként adja meg:

$$V = V_k f = V_k \frac{N_t}{N},$$

ahol N_t a találatok, N pedig a próbapontok száma.

A hiba értékének megállapításához szükséges az az észrevétel, miszerint a "elfogadás-elutasítás" egy változata a mintavételezés módszerének. A (3.2)-es egyenlet [2]

$$V = V_k f_t = V_k \langle f\rangle, \tag{3.4}$$

ahol f a következők szerint definiált függvény:

$$f(x) = \begin{cases} 1, \text{ ha x a keresett térfogaton belül van} \\ 0, \text{ minden más esetben} \end{cases}.$$

Tehát a hiba meghatározására szolgáló képlet:

$$\sigma_{V_k\langle f\rangle} \approx V_k\sqrt{\frac{\langle f^2\rangle - \langle f\rangle^2}{N}},$$

ahol

$$\langle f\rangle = \frac{\sum_{i=1}^{N} f(x_i)}{N} \quad \text{és} \quad \langle f^2\rangle = \frac{\sum_{i=1}^{N} f^2(x_i)}{N}.$$

Az f függvényt figyelembe véve a képlet jelentősen egyszerűsíthető, hiszen $\langle f\rangle = N_t/N$. Ennél fogva a hiba meghatározása a következő képpen fog kinézni:

$$\sigma_{V_k\langle f\rangle} \approx V_k\sqrt{\frac{(N_t/N) - (N_t/N)^2}{N}} = V_k\sqrt{\frac{N_t - N_t^2/N}{N^2}} = V_k\frac{\sqrt{N_t - N_t^2/N}}{N},$$

az integrál képlete pedig felírható úgy, mint:

$$V = V_k\frac{N_t}{N} \pm V_k\frac{\sqrt{N_t - N_t^2/N}}{N}. \tag{3.5}$$

Ha pedig igaz az, hogy $N_t << N$, könnyen kapjuk azt, hogy:

$$V = V_k\frac{N_t}{N} \pm V_k\frac{\sqrt{N_t/N}}{N}, \quad \text{ha} \quad N_t << N.$$

3.5. Fontossági mintavétel

A Monte-Carlo integrál pontosságának javítására lehetőség van a σ^2 szórásnégyzet csökkentésére. Fontos megjegyezni, hogy a σ^2 bármilyen nem állandó adateloszlás esetén valamilyen véges szám, nem nulla, mikor $N \to \infty$, természetesen az N növelésével a hiba értéke tart a 0-hoz [2, 3].

A fontossági mintavétel ötlete, hogy készítsünk a $f(x)$ függvény transzformálásával egy olyan függvényt, amely laposabb. Tegyük fel, hogy $g(x)$ egy olyan [a, b] intervallumon értelmezett függvény, hogy:

$$\frac{f(x)}{g(x)}$$

meglehetősen lapos, és $g(x) > 0$ minden x esetén. Írjuk fel a következő egyenletet, ahol $g(x)$ függvényt integrálva megkapjuk $G(x)$-t:

$$I = \int_a^b f(x)dx = \int_a^b \frac{f(x)}{g(x)} g(x)dx = \int_a^b \frac{f(x)}{g(x)} dG(x), \quad G(x) = \int_a^b g(x)dr.$$

Elvégezve egy $r = G(x)$ változó cserét a képletben, ugyanazt az integrált kapjuk, mint az eredeti, azonban ez sokkal hatékonyabb, mivel az integrálandó függvény laposabb.

$$I = \int_{G(a)}^{G(b)} \frac{f(G^{-1}(r))}{g(G^{-1}(r))} dr.$$

Monte-Carlo módszerrel ez így írható fel, ahol r_i egységes véletlen számok:

$$I = \frac{1}{N} \sum_{i=1}^{N} \frac{f(G^{-1}(r_i))}{g(G^{-1}(r_i))}.$$

Ebben az esetben meg kell határozni a G^{-1}-t. Egy másik lehetőség, hogy valamilyen módon $g(x)$ eloszlású véletlen számokat generálunk. Így az összeg a következőképpen változik, ahol $x_i^{(g)}$ pontok $g(x)$ eloszlású véletlen számok [2]:

$$I = \frac{1}{N} \sum_{i=1}^{N} \frac{f(x_i^{(g)})}{g(x_i^{(g)})}.$$

A használatának egyik előnye, hogy nem kell törődni a dimenzió változásával, és nem kell megkövetelni, hogy $ag(x) > f(x)$, amely néha nehéz vagy inkább lehetetlen, különösen magas dimenziókban.

3.6. Változóvezérlés

A változóvezérlés egy másik szóráscsökkentő eljárás, amely néhány mögöttes információra támaszkodik az f függvényről [2, 3]. Az ötlet hasonló a fontossági mintavételhez. Cseréljük le az integrálandó függvényt egy sokkal laposabb függvényre, így csökken a szórás és vele együtt a hiba is, de osztás helyett vonjuk ki:

$$I = \int_a^b f(x)dx = \int_a^b (f(x) - g(x))dx + \int_a^b g(x)dx.$$

A $g(x)$ függvényt úgy választjuk meg, hogy $(f-g)$ szórása kisebb a f szórásánál és $\int_a^b g(x)dx$ pedig ismert.

Ennek a megközelítésnek több előnye is van a fontossági mintavétellel szemben. Az egyik előnye, hogy nem jelent problémát, ha a $g(x)$ az intervallum valamelyik pontjában 0-t vesz fel vagy éppenséggel negatív. A másik, hogy nem kell ismerni, hogy a véletlen számok eloszlását a $g(x)$ felett.

Az $f-g$ varianciája

$$Var(f-g) = Var(f) + Var(g) - 2Cov(f,g),$$

ahol $Cov(f,g)$ az f és g függvények közötti kovariancia. Annak érdekében, hogy ez használható legyen, igaznak kell lennie a $2Cov(f,g) > Var(g)$ egyenlőtlenségnek, azaz f és g hasonló alakúak [2, 3].

3.7. Kettős integrál

Többváltozós integrálok megoldása sokszor igen nehéz, azonban a Monte-Carlo módszerrel ez nem sokkal nehezebb feladat annál, mintha csak egy változónk lenne [2, 3, 18]. A Monte-Carlo módszer vizsgálatához legyen adott a következő:

$$I = \int_{x_1}^{x_2} \int_{y_1}^{y_2} f(x,y)dydx$$

Ez az integrál felírható az (x_1, x_2) és (y_1, y_2) intervallumok által körbezárt téglalap területének és az $f(x,y)$ függvény értékeiből számolt átlag szorzatával:

$$I = \int_{x_1}^{x_2} \int_{y_1}^{y_2} f(x,y)dydx = S \cdot \overline{f},$$

ahol $S = (x_2 - x_1)(y_2 - y_1)$, és $\overline{f}$ pedig az $f(x,y)$ függvény értékeinek az átlaga.

Tehát az integrál értéke megbecsülhető, ha meghatározzuk a függvény értékeinek átlagát az adott területen. Ehhez alkalmazhatunk véletlen számokat. Véletlenszerűen kiválasztunk N pontot, amelyek az (x_1, x_2) és (y_1, y_2) intervallumok között helyezkednek el. Az $f(x,y)$ függvényt kiértékeljük minden pontban, és meghatározzuk az értékek átlagát:

$$\overline{f} = \frac{1}{N} \sum_{i=1}^{N} f(x_i, y_i).$$

Kettős integrálok számítása során két eset lehetséges. Az első az, amikor az integrált egy téglalap alakú területen becsüljük:

$$\int_{x_1}^{x_2} \int_{y_1}^{y_2} f(x,y)dydx,$$

ahol x_1, x_2, y_1, y_2 konstansok. Ebben az esetben egyszerűbb dolgunk van. Generálunk N pontot ezen a területen, és mind az N pontban kiértékelve a függvényt, kiszámoljuk a függvény értékeinek átlagát. Ezt követően kiszámoljuk a téglalap területét, és meghatározzuk a kapott átlag és a terület szorzatát.

A második eset az, amikor az integrált nem egy téglalap alakú területen vesszük:

$$\iint\limits_R f(x,y)dxdy,$$

ahol R az a terület az $x-y$ síkon, ami felett a függvényt integrálni szeretnénk. Ebben az esetben az R-t elhelyezzük egy téglalapban, és ennek a téglalapnak a területén generálunk pontokat, azonban csak azokban a pontokban értékeljük ki a függvényt, amelyek beleesnek az R-be, és csak ezekből az értékekből számítunk átlagot [12, 18].

Jelölje D az R területet magába foglaló téglalapot. Ha a D területet beszórjuk véletlen pontokkal, nem tudjuk, hogy mennyi pont fog beleesni az R-be. Erre lehet adni egy becslést: $N_R \approx N_D \cdot S_R/S_D$, ahol N_R az R-be eső pontok száma, N_D a D-be eső pontok száma, S_R és S_D az R illetve D területe. Ha tudjuk, hogy körülbelül mennyi pontot szeretnénk az R területen, akkor ahhoz körülbelül $N_R \cdot S_D/S_R$ pontot szükséges generálni a D területén [18].

4. fejezet

Véletlen számgenerálás

A Monte-Carlo módszer központi részét a véletlenszerű számok képezik, ezért fontos, hogy a megfelelő forrás álljon rendelkezésre. [2] A véletlen számokat három nagy kategóriába lehet sorolni:

- valódi véletlen számok –nem lehet tudni, hogy mi lesz a következő szám;
- pszeudovéletlen számok –algoritmikusan előállított számsorozat, amelyek implementációját fel lehet használni szimulációk során;
- kvázirandom számok –véletlen számokként működnek valamilyen szimulációban, de jól vannak rendezve néhány más típusban, céljuk egy N dimenziós tér egyenletes kitöltése.

A Monte Carlo integrál megvalósításához pszeudovéletlen számok és kvázirandom számok is egyaránt használhatóak. Mivel szinte az összes program pszeudovéletlen számokat állít elő, ezért ez az elterjedtebb a gyakorlati életben. Pszeudovéletlen számgenerátorok pl. a lineáris kongruenciagenerátorok, a Mersenne Twister és a Fibonacci generátorok. Azokat a generátorokat nevezhetjük jó generátoroknak, amik bizonyos statisztikai teszteket teljesítenek. A Mersenne Twister és a lineáris kongruenciagenerátorok is ennek elvégzése során hatásosnak bizonyultak.

A számítógépes véletlenszám-generálás tulajdonképpen az egyenletes eloszláson alapszik. Ezek a technikák lényegében egy hosszú sorozatot készítenek, amely matematikai tulajdonságai alapján megfelelő minőségűnek tekinthető. Alapvetőleg a szoftverek az ún. Mersenne Twister-eljárást alkalmazzák. Ez egy igen gyors, matematikai szempontból jó minőségű, pszeudovéletlen számokat előállító algoritmus [2, 8]

A pszeudovéletlen számoknak alapvetően három közös jellemzőjük van:

- kezdeti érték –egy adott kezdeti értékre a számgenerátor ugyanazt a számsort adja vissza;
- periódus –a sorozat egy idő után ismétlődik, ezt a számot nevezzük periódusnak, aminek jóval nagyobbnak kell lennie, mint amilyen hosszú sorozatot akarunk generálni, különben előfordulnak majd ismétlődő elemek;
- intervallum –ez általában $(0,1), (0,1], [0,1)$ vagy $[0,1]$

Leggyakrabban egy 4-byte-os számként reprezentálják a periódust, $2^{32} \approx 4 \cdot 10^9$, így előállítható ugyanennyi véletlen szám is [2].

4.1. Lineáris kongruencia- és Mersenne Twister generátorok

A lineáris kongruenciagenerátor azon legegyszerűbb számgenerátorok közé tartozik, amelyek segítségével pszeudovéletlen számokat hozhatunk létre. [9, 2] Egy N hosszúságú sorozat készítéséhez 4 paraméterre van szükség: $m > N, A, a, b$. Az algoritmus a következőképpen néz ki:

$$y_1 \equiv A \pmod{m} \quad 0 \leq y_1 \leq m,$$
$$y_{n+1} \equiv a \cdot y_n \pmod{m} \quad 0 \leq y_{n+1} \leq m,$$
$$n = 1, 2, ...N - 1$$

Továbbá legyen $i = 1, 2, ...N$ esetén $x_i = \frac{y_i}{m}$, a bizonyítottan pszeudovéletlen sorozatnak tekinthető a $[0, 1)$ intervallumon. [9]

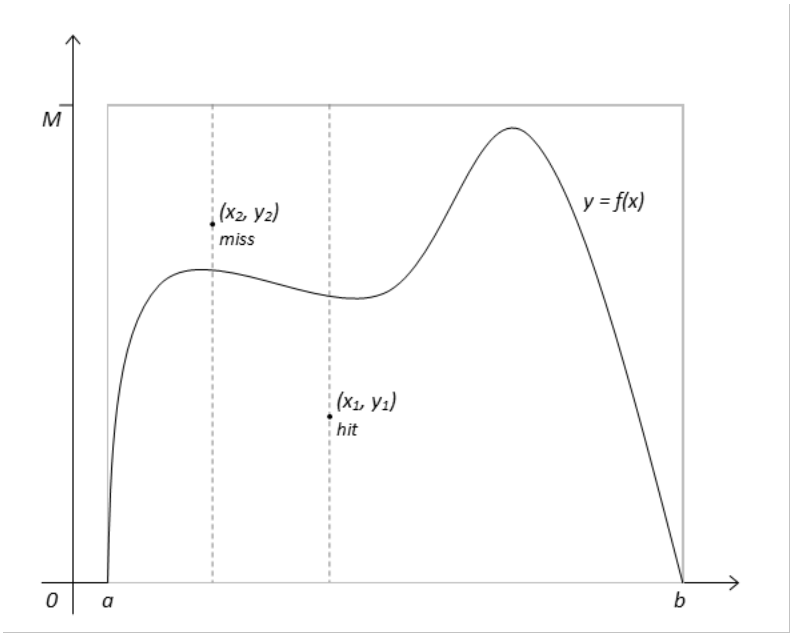
A Mersenne Twister számgenerátor a periódusáról kapta a nevét, ami egy Mersenne-prím $(2^{19937} - 1)$ [2]. Ez a leggyakrabban használt véletlen számgenerátor, mivel jelenleg beépített függvényként használja a legtöbb program. A generátor periódusa egy körülbelül 6000 számjegyből álló szám, ami azt jelenti, hogy ha másodpercenként generálunk egymilliárd számot, akkor 5985 évig tart, mire a számsor újrakezdődik [2, 9].

4.2. Neumann-féle elutasítási módszerrel

A véletlen számok bármilyen, nem egyenletes eloszlás esetén szinte mindig 0 és 1 közötti egyenletes eloszlású véletlen számokból indulnak ki. Két megközelítése van a nem egységes eloszlású véletlen számok generálásának: az analitikus és a Neumann-féle elutasítási módszer.

A Neumann-féle elutasítási módszer egy tisztán numerikus megközelítése a véletlen számok létrehozásának, és bármely véges intervallumon értelmezett véges értékű függvényre működik, függetlenül attól, hogy a függvény invertálható vagy integrálható [2, 4].

Legyen $f(x)$ függvény definiálva egy véges $[a, b]$ intervallumon, $\forall x \in [a, b]$. Továbbá legyen M egy olyan valós szám, hogy $M \geq f(x)$, minden x-re az adott intervallumon.



4.1. ábra. Elutasítási módszer

Az algoritmus a következő:

1. generáljunk egyenletes eloszlású x számot az $[a, b]$ intervallumon;
2. generáljunk egyenletes eloszlású y számot a $[0, M]$ intervallumon;
3. ha $y > f(x)$, akkor ez nem talált, a visszatérési érték hamis,
4. ellenkező esetben ez egy találat, a visszatérési érték pedig x.

Az így kapott x véletlen számok az $f(x)$ szerint oszlanak el, az y viszont csak az ellenőrzés szempontjából lényeges, így nem kapjuk meg visszatérési értékként [2].

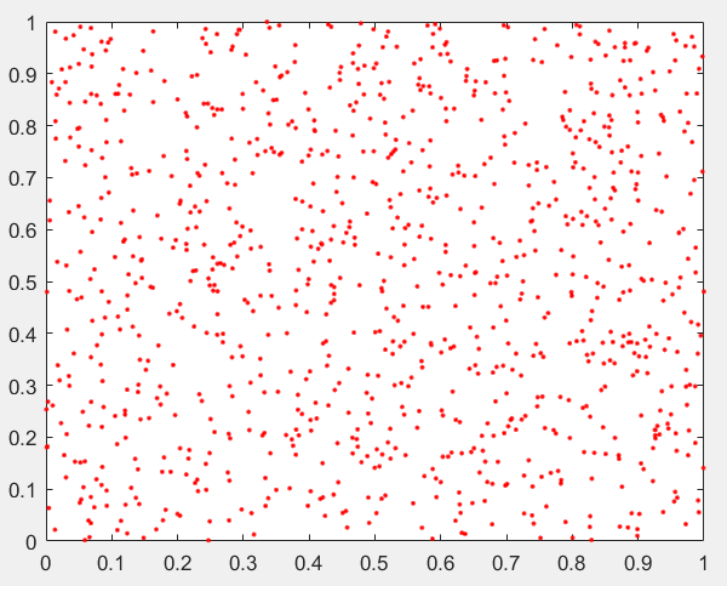
Ez a módszer elég egyszerűnek és könnyűnek tűnik, azonban a háttérben felesleges munkát is végzünk, mivel az elutasított értékeket hiába generáltuk. Annak a valószínűsége, hogy a generált szám talált:

$$P_t = \frac{\int_a^b f(x)dx}{M(b-a)} = \frac{1}{M(b-a)}.$$

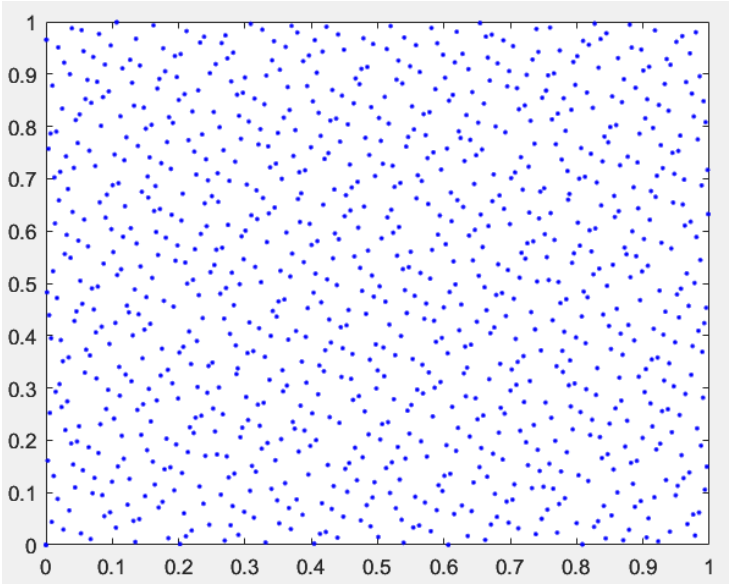
Egyes függvények esetén, mint például ha az nagyon csúcsos, ez a módszer nem épp a legalkalmasabb, mivel könnyen lehet, hogy a találatok száma igen kicsi lesz.

4.3. Pszeudovéletlen pontok és Halton-pontok

A peszeudovéletlen számok esetén gyakran alakulhatnak ki csomósodási pontok a generálás során. Halton-pontok alkalmazásával egy egyenletesen elosztott ponthalmazt kaphatunk egy adott tartományon belül. A Halton-pontok a van der Corput-sorozatokon alapulnak. [9, 2]



4.2. ábra. Pszeudovéletlen pontok



4.3. ábra. Halton-pontok

Az ábrákon jól megfigyelhető a pszeudovéletlen és a Halton pontok közötti különbség. Látható, hogy az utóbbi egyenletesebb eloszlású pontokat adott, így számítások során is jól alkalmazható.

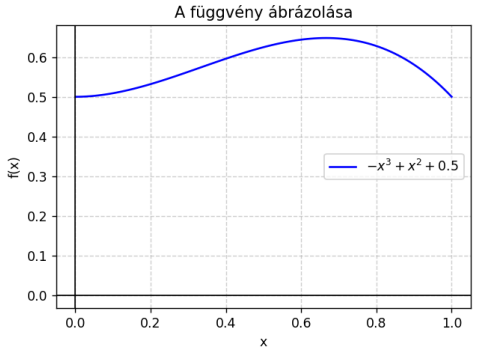
5. fejezet

Integrálás numerikus és Monte-Carlo módszerekkel a gyakorlatban

5.1. A trapéz szabály alkalmazása egyváltozós határozott integrálok közelítésére

Az előző fejezetekben már tárgyaltuk, hogy hogyan működik a trapéz szabály: két pontra illesztett szakaszok alatti trapézok területének összegével ad becslést a határozott integrál értékére. A továbbiakban egy példán keresztül mutatom be a folyamatot abban az esetben, ha az intervallumot egyenlő hosszúságú részintervallumokra bontjuk.

Legyen adott az $f(x) = -x^3 + x^2 + 0.5$, integráljuk a $[0; 1]$ intervallumon.

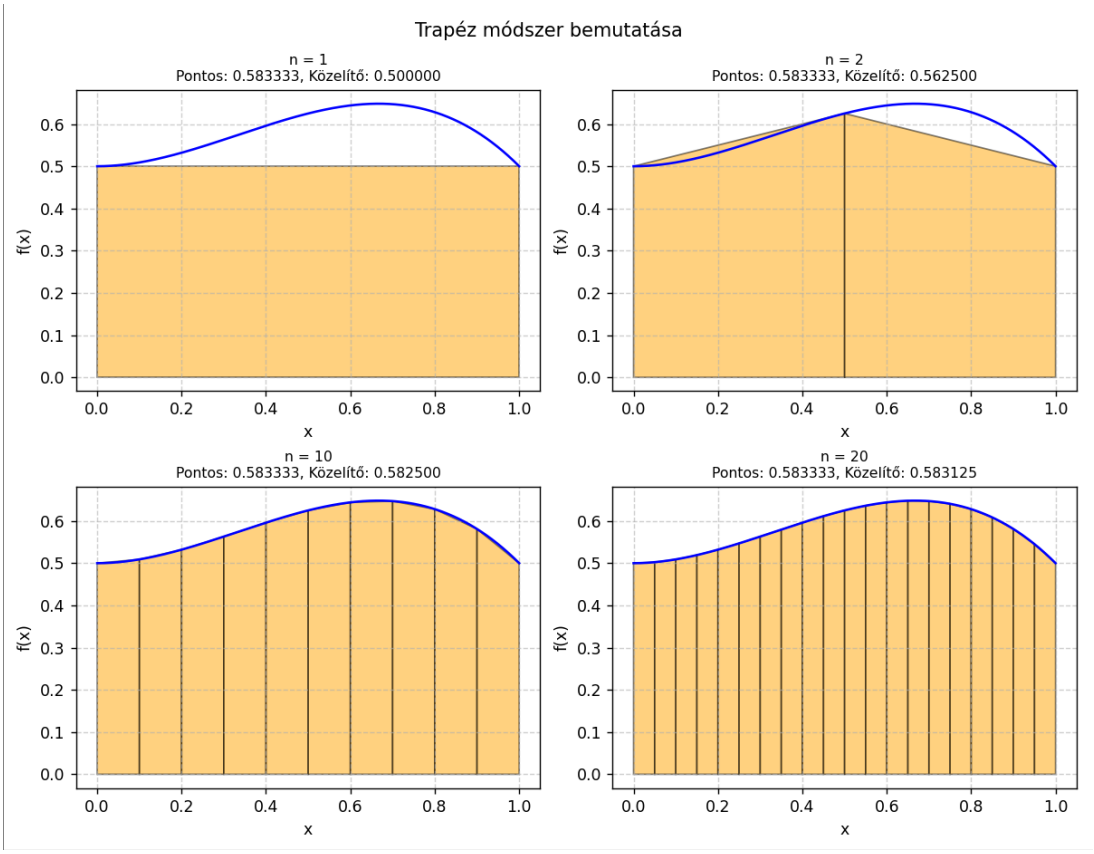


5.1. ábra. Az $f(x) = -x^3 + x^2 + 0.5$ a $[0; 1]$ intervallumon

Az f függvényt szimbolikusan is tudjuk integrálni, és alkalmazhatjuk a Newton-Leibniz formulát:

$$\begin{aligned}\int_0^1 f(x)dx &= \int_0^1 (-x^3 + x^2 + 0.5)dx = \int_0^1 (-x^3)dx + \int_0^1 x^2 dx + \int_0^1 0.5dx \\ &= -\int_0^1 (x^3)dx + \int_0^1 x^2 dx + \int_0^1 0.5dx = -\left[\frac{x^4}{4}\right]_0^1 + \left[\frac{x^3}{3}\right]_0^1 + [0.5x]_0^1 \\ &= -\left(\frac{1^4}{4} - \frac{0^4}{4}\right) + \left(\frac{1^3}{3} - \frac{0^3}{3}\right) + (0.5 \cdot 1 - 0.5 \cdot 0) = -\frac{1}{4} + \frac{1}{3} + 0.5 \\ &\approx 0.583333\end{aligned}$$

Az (5.2)-es ábrán jól megfigyelhető, hogy minél több részintervallumra bontjuk az intervallumot, a közelítő érték annál inkább konvergál a pontos értékhez.



5.2. ábra. Az $f(x) = -x^3 + x^2 + 0.5$ függvény határozott integrál értékének $[0; 1]$ intervallumon való közelítése a trapéz módszerrel, n - a részintervallumok száma

Bemutatás és értelmezés szempontjából, ha az intervallumot nem osszuk fel részintervallumokra, akkor a trapéz szabály elég egyszerű, és rendkívül pontatlan. Ebben az esetben a közelítés a következő:

$$\int_0^1 f(x)dx \approx \frac{1-0}{2} \cdot (f(0) + f(1)) = \frac{1}{2} \cdot (0.5 + 0.5) = \frac{1}{2} \cdot 1 = 0.5$$

A két pontot itt összekötve egy téglalapot kaptunk, és az integrál értékének közelítésére ennek a téglalapnak a területét kaptuk. Nézzük meg abban az esetben, ha a $[0; 1]$ intervallumot két részintervallumra bontjuk. Ebben az esetben két trapéz területének összegével ad becslést, a három osztópont pedig $x_0 = 0, \quad x_1 = 0.5, \quad x_2 = 1$. Az intervallumok hossza $h = \frac{1-0}{2} = = 0.5$.

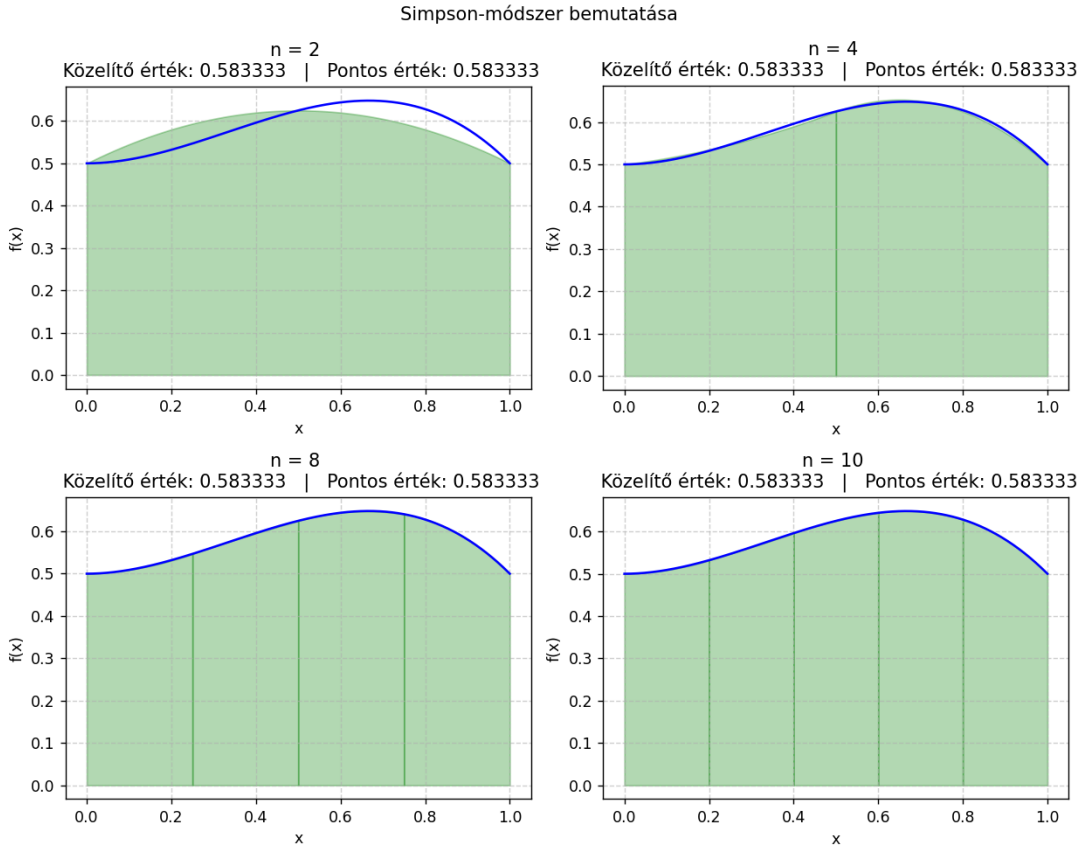
$$\begin{aligned}
\int_0^1 f(x)dx &\approx \frac{0.5}{2}(f(0) + f(0.5)) + \frac{0.5}{2}(f(0.5) + f(1)) \\
&= \frac{0.5}{2}(f(0) + f(0.5) + f(0.5) + f(1)) \\
&= 0.25\,(f(0) + 2f(0.5) + f(1)) \\
&= 0.25\left(0.5 + 2 \cdot \frac{5}{8} + 0.5\right) = 0.5625
\end{aligned}$$

A közelítés ebben az esetben is csak egy tizedesjegy pontossággal adott becslést, azonban az (5.2)-es ábrán jól megfigyelhető, hogy az intervallum felbontásának finomításával pontosabb eredményt kapunk. A példa alapján azt is megfigyelhetjük, hogy a közelítő összegben a szélső pontokat 1-es, míg a belső pontokat 2-es súllyal veszi.

5.2. A Simpson-módszer alkalmazása egyváltozós határozott integrálok közelítésére

A második fejezetben már szó esett részletesebben a Simpson-módszerről, amely másod- vagy harmadfokú polinomokkal közelíti a határozott integrál értékét. Vizsgáljuk meg a módszer működését az előző példa alapján, abban az esetben, ha az intervallumot egyenlő hosszúságú részintervallumokra bontjuk. Legyen:

$$f(x) = -x^3 + x^2 + 0.5, \quad \int_0^1 f(x)dx \approx 0.583333$$



5.3. ábra. Az $f(x) = -x^3 + x^2 + 0.5$ függvény határozott integrál értékének $[0; 1]$ intervallumon való közelítése a Simpson-módszerrel, n - a részintervallumok száma

Mivel másodfokú polinom illesztéséhez 3 pontra van szükség, ezért a Simpson-módszer alkalmazásához minden esetben páros számú felosztást kell megadni! Vizsgáljuk meg a módszert $n = 2$ esetén, és alkalmazzuk az általános képletet 3 pontra, ahol $h = (1 - 0)/2 = 0.5$:

$$\int_0^1 f(x)dx \approx \frac{0.5}{3}\left(f(0) + 4f(0{,}5) + f(1)\right) = \frac{0.5}{3}\left(0.5 + 4 \cdot 0.625 + 0.5\right) \approx 0.583333$$

Ebben az esetben már $n = 2$ esetén is pontos értéket kaptunk, ami nem meglepő, mivel az f függvény is egy hatványfüggvény, és a közelítő függvény is az. Így az (5.3)-as ábra azt szemlélteti, hogy a felbontások finomításával a pontokra illesztett függvények hogyan simulnak az eredeti függvényhez. Az általános képlet megértése érdekében alkalmazzuk $n = 4$ esetén is. Az értéke nem fog változni, de az algoritmus jól szemléltethető. Az osztópontok a következők: $x_0 = 0, \quad x_1 = 0.25, \quad x_2 = 0.5, \quad x_3 = 0.75, \quad x_4 = 1, h = (1-0)/4 = 0.25.$

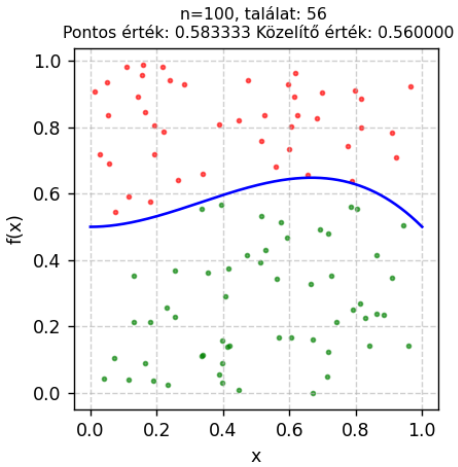
$$\begin{aligned}\int_0^1 f(x)dx &\approx \frac{0.25}{3}\left(f(0) + 4f(0.25) + f(0.5) + f(0.5) + 4f(0.75) + f(1)\right) \\ &= \frac{0.25}{3}\left(f(0) + 4\left(f(0.25) + f(0.75)\right) + 2f(0.5) + f(1)\right) \\ &= \frac{0.25}{3}\left(0.5 + 4\left(0.546875 + 0.640625\right) + 2 \cdot 0{,}625 + 0.5\right) \\ &\approx 0.583333\end{aligned}$$

Itt tulajdonképpen először az x_0, x_1 és x_2 osztópontok használatával illeszt görbét, majd az x_2, x_3, x_4 osztópontokkal, ezt követően pedig összeadja a kapott eredményeket.

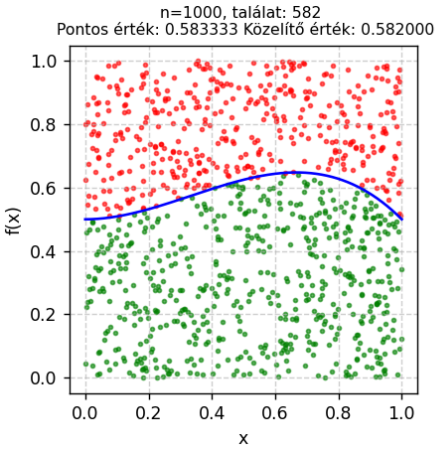
5.3. A Monte-Carlo módszerek alkalmazása egyváltozós határozott integrálok közelítésére

A Monte-Carlo módszerek közül az elutasítási módszert és a mintavételezést fogom bemutatni. Az elutasítási módszer lényege, hogy azon a területen (térfogaton) mely magába foglalja az integrálandó függvényt, véletlenszerűen pontokat veszünk fel rajta, és megszámoljuk, hogy hány pont találta el a keresett területet, és hány nem. Ez követően a találatok számának és az összes pontnak az arányát megszorozzuk a terület értékével, amelyen a véletlen pontokat felvettük.

Példaként vegyük a már előzőekben is használt $f = -x^3 + x^2 + 0.5$ függvényt, és a $[0, 1]$ intervallumon adjunk becslést rá. A pontok generálásához és az érték közelítéséhez egy Python programot fogunk használni. A függvényt tartalmazza egy $[0, 1] \times [0, 1]$-es négyzet, melynek területe 1.



5.4. ábra. Monte-Carlo elutasítási módszer 100 ponttal



5.5. ábra. Monte-Carlo elutasítási módszer 1000 ponttal

Ha a területet 100 ponttal szórjuk be, mely közül 56 pont esik a függvény alá, tehát eltalálta

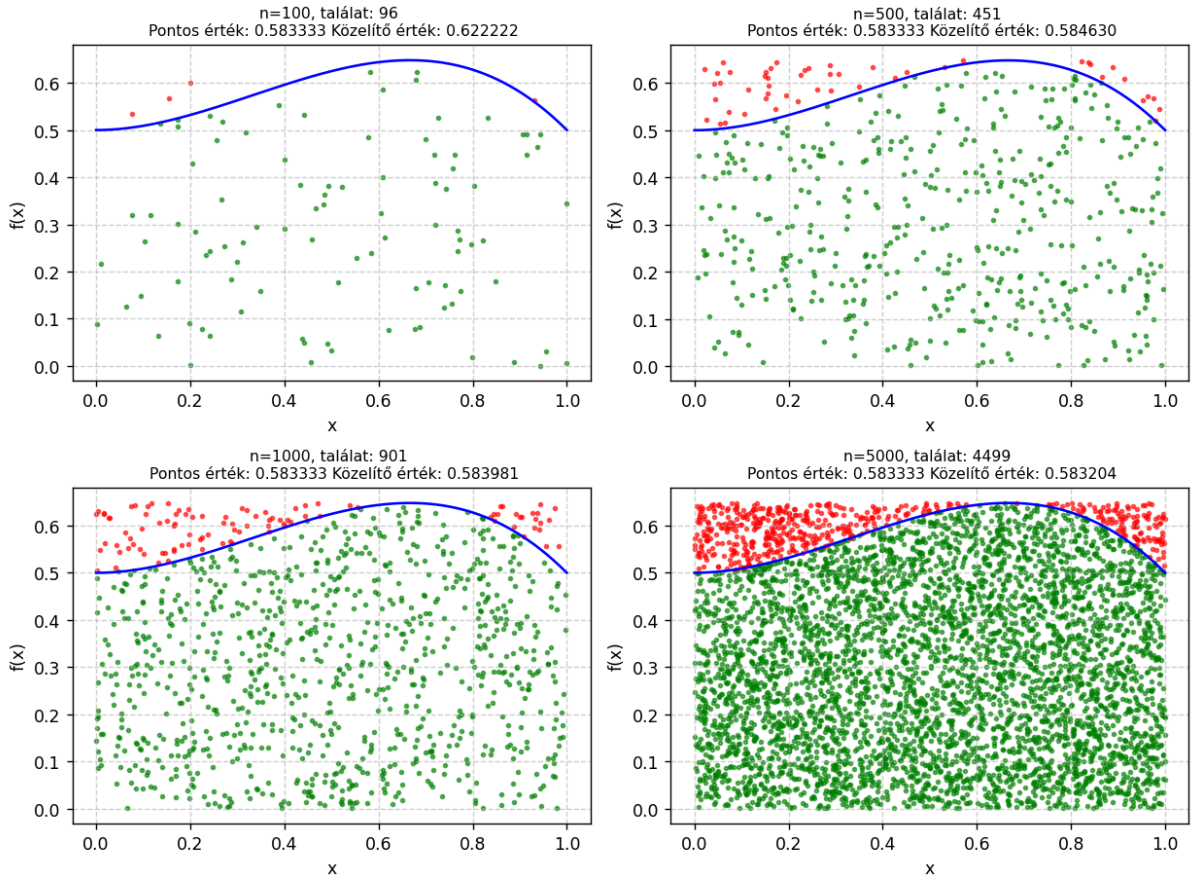
a keresett területet, akkor a közelítő érték a következő:

$$\int_0^1 f(x) \approx \frac{56}{100} \cdot 1 = 0.56.$$

Ha ugyanezt a területet beszórjuk 1000 ponttal, melyből 582 esik a keresett tartományba, akkor

$$\int_0^1 f(x) \approx \frac{582}{100} \cdot 1 = 0.582.$$

Ezt szemlélteti az (5.4) és (5.5)-s ábra. Jól látható, hogy a függvényt egy kisebb tartományban is el lehet helyezni. Ehhez megkereshetjük a függvény y_{max} maximumát, a terület pedig, melyen a pontokat generáljuk $[0, 1] \times [0, y_{max}]$. Az (5.6)-os ábra ezt az esetet szemlélteti.



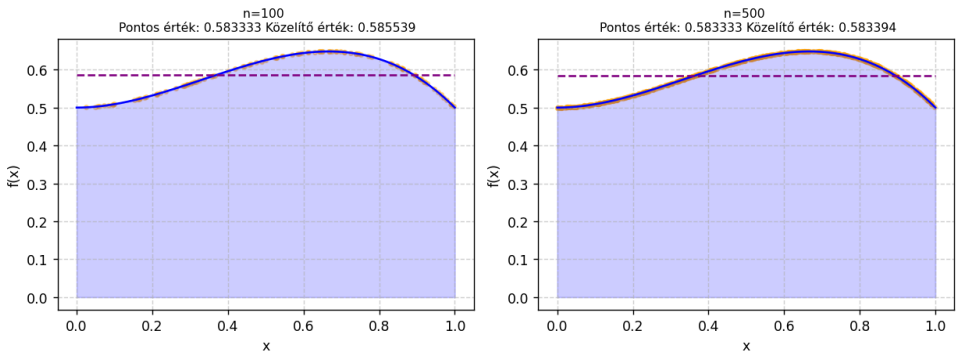
5.6. ábra. Az $f(x) = -x^3 + x^2 + 0.5$ függvény határozott integrál értékének $[0; 1]$ intervallumon való közelítése a Monte-Carlo elutasítási módszerrel, n - a generált véletlen pontok száma

A program segítségével megkeressük a függvény maximumát: $y_{max} \approx 0.648148$. Ebben az esetben a terület $(1 - 0) \cdot (0.648148 - 0) = 0.648148$. Ha n az összes generált pont, n_t pedig a

találatok száma:

$$\int_0^1 f(x) \approx \frac{96}{100} \cdot 0.648148 \approx 0.622222, \quad n = 100, \quad n_t = 96$$
$$\int_0^1 f(x) \approx \frac{451}{500} \cdot 0.648148 \approx 0.584630, \quad n = 500, \quad n_t = 451$$
$$\int_0^1 f(x) \approx \frac{901}{1000} \cdot 0.648148 \approx 0.583981, \quad n = 1000, \quad n_t = 901$$
$$\int_0^1 f(x) \approx \frac{4499}{5000} \cdot 0.648148 \approx 0.583204, \quad n = 5000, \quad n_t = 4499$$

A mintavételezés módszer alkalmazásához elegendő csak az x intervallumán véletlenszerű számokat generálni. A f függvényt kiértékeljük ezekben a pontokban, átlagot számolunk belőlük, a kapott átlagot beszorozzuk az intervallum hosszával, melyen a számokat generáltuk. Ez esetben az intervallum hossza $1 - 0 = 1$, így a kapott érték egyben az átlag értékével egyezik meg



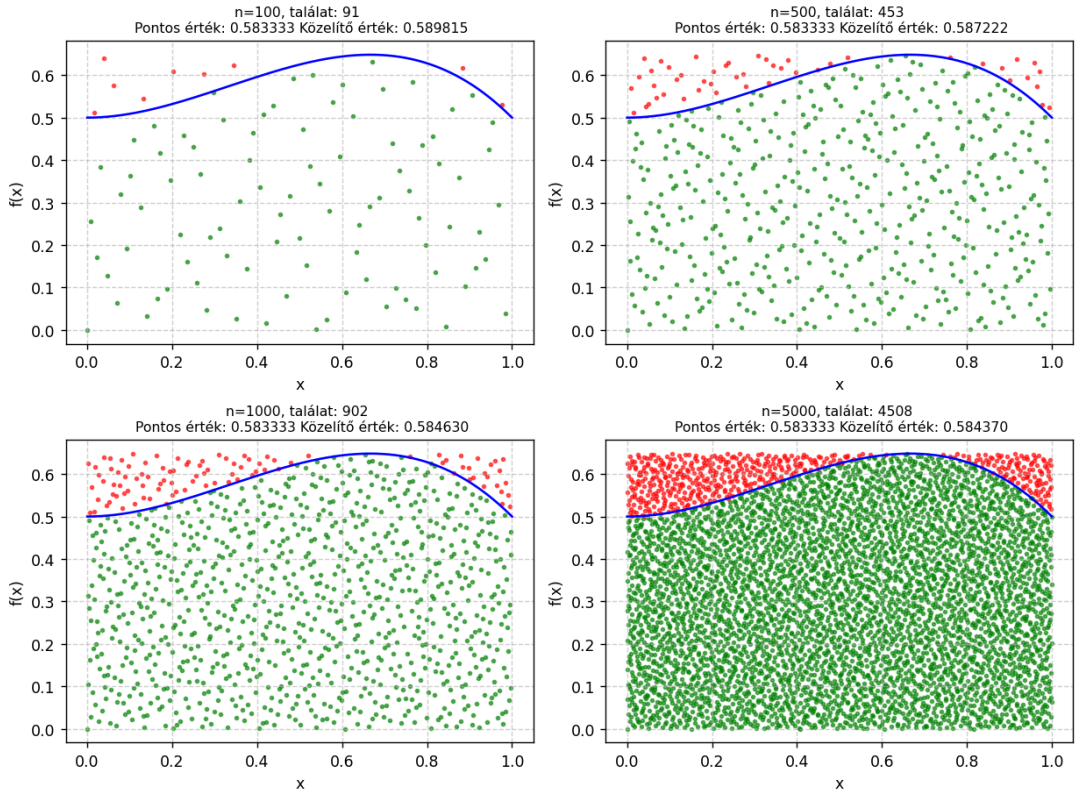
5.7. ábra. Az $f(x) = -x^3 + x^2 + 0.5$ függvény határozott integrál értékének $[0; 1]$ intervallumon való közelítése a Monte-Carlo mintavételezés módszerrel, n - a generált véletlen számok darabszáma

Az (5.7)-es ábrán látható, hogy már 500 pont esetén közelebbi tudott adni, mint az elutasítási módszer 5000 pont esetén. Fontos azonban megjegyezni, hogy túl kevés vagy túl sok pont generálása torzíthatja is az eredményt, így nem vezet pontosabb közelítéshez.

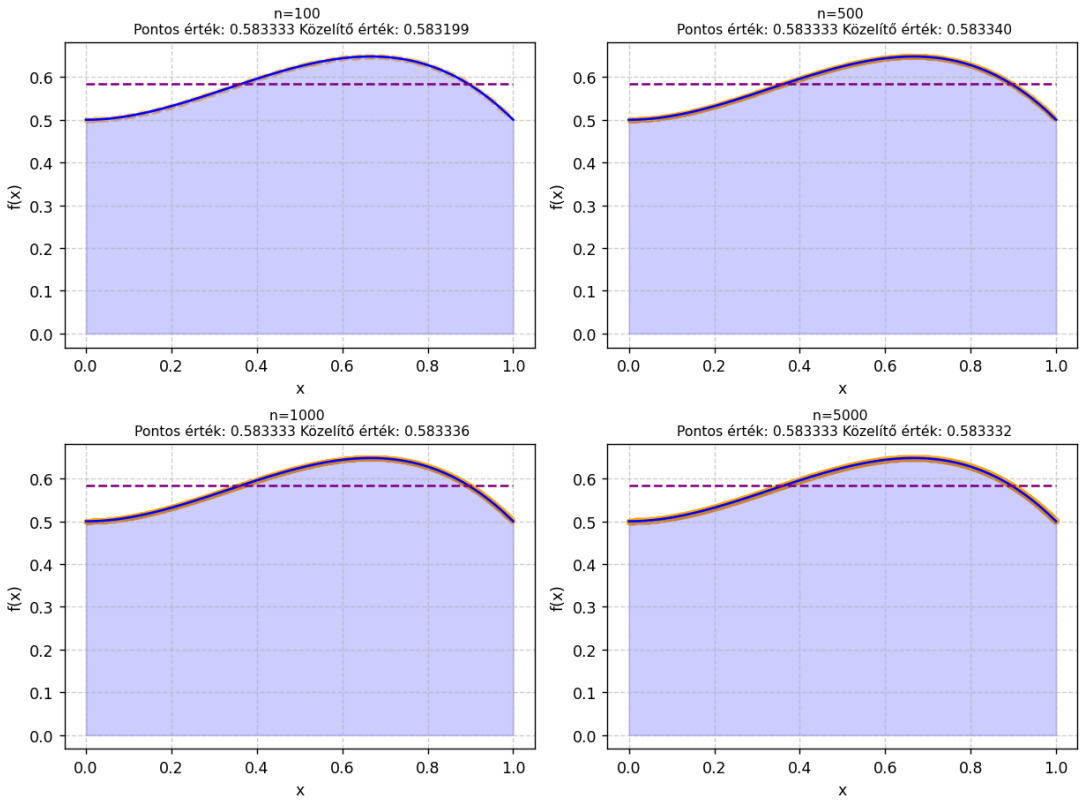
5.4. A Monte-Carlo módszer alkalmazása Halton-pontokkal

Korábbi fejezetekben már említettük, hogy mivel a Monte-Carlo módszer alapja a véletlenszerűség, így az eredményt befolyásolja az, hogy a generált pontokat milyen módszerrel állítjuk elő. Az előző példában a pontok generálásához a Python beépített és alapesetben használt Mersenne Twister generátort használtuk pszeudovéletlen pontok előállítására. Most nézzük meg, milyen eredményt adnak a Monte-Carlo módszerek Halton-pontokkal. Ehhez vizsgáljuk a már előzőekben használt függvényt.

Az (5.8)-as és (5.9)-es ábrákon látható eredményekből az látszik, hogy az elutasítási módszer ezzel nem lett pontosabb, azonban a mintavételezési módszer majdnem pontos eredményt adott.



5.8. ábra. Az $\int_0^1 f(x) = -x^3 + x^2 + 0.5$ integrál értékének közelítése a Monte-Carlo elutasítási módszerrel, Halton-pontokon, n - a generált véletlen pontok száma

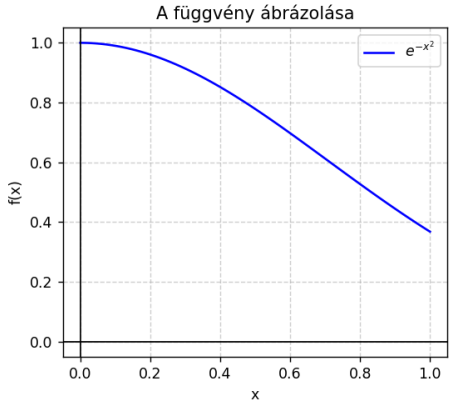


5.9. ábra. Az $\int_0^1 f(x) = -x^3 + x^2 + 0.5$ integrál értékének közelítése a Monte-Carlo mintavételezés módszerrel, Halton-pontokon, n - a generált véletlen pontok száma

5.5. A numerikus és Monte-Carlo módszerek összehasonlítása különböző egyváltozós határozott integrélok közelítésére

Ebben a fejezetben összehasonlítjuk a numerikus és Monte-Carlo módszereket két példán keresztül.

5.1. példa Először figyeljük meg a módszerek működését a (2.1)-es példában már látott $\int_0^1 e^{-x^2} dx$ integrál segítségével. A példában megkaptuk, hogy $\int_0^1 e^{-x^2} dx \approx 0.746824$.

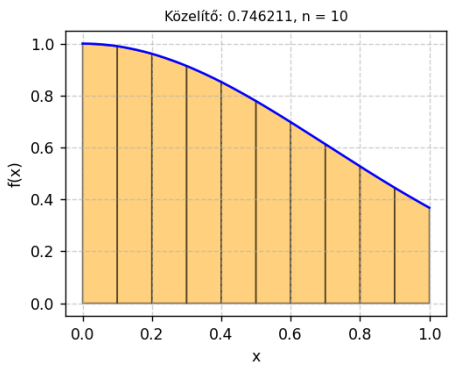


5.10. ábra. A e^{-x^2} függvény a $[0; 1]$ intervallumon

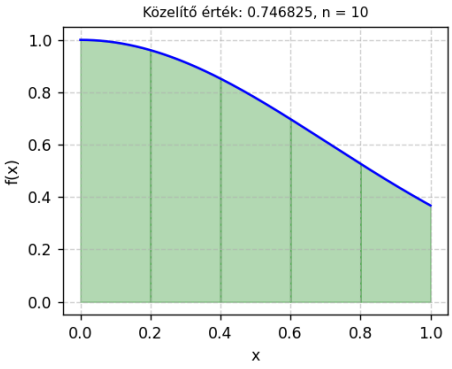
Az 5.1-es táblázat a trapéz szabály és Simpson-módszer alkalmazása által kapott eredményeket mutatja be. Látható, hogy a trapéz szabály elég lassan konvergál, és 100 részintervallum esetén is csak 10^{-4}-en pontossággal adott közelítést. A Simpson-módszer sokkal gyorsabban konvergált, már 20 részintervallumra való osztáskor 10^{-6}-on pontosságú közelítést adott.

n	Pontos érték	Közelítő érték trapéz szabállyal	Közelítő érték Simpson-módszerrel
2	0.746824	0.731370	0,747180
10	0.746824	0.746211	0,746825
20	0.746824	0.746671	0,746824
50	0.746824	0.746800	-
100	0.746824	0.746818	-

5.1. táblázat. Az trapéz szabály és Simpson-módszer összehasonlítása



5.11. ábra. A trapéz szabály alkalmazása



5.12. ábra. A Simpson-módszer alkalmazása

Alkalmazzuk a Monte-Carlo módszereket különböző pontgenerálási eljárásokkal. A terület, mely magába foglalja az integrálandó függvényt egy $[0;1] \times [0;1]$ négyzet, melynek területe $1n.e.$. Az 5.2-es táblázat az elutasítási módszer alkalmazásával kapott eredményeket foglalja össze különböző módszerekkel generált pontok esetén, ahol n az összes generált pont, n_t pedig a találatok száma:

n	Pontos érték	Pszeudovéletlen pontok		Halton-pontok	
		n_t	közelítés	n_t	közelítés
500	0.746824	357	0,714000	376	0,752000
1000	0.746824	743	0,743000	751	0,751000
5000	0.746824	3724	0,744800	3738	0,747600
10000	0.746824	7474	0,747400	7467	0,746700
20000	0.746824	14917	0,745850	14932	0,746600
100000	0.746824	74687	0,746870	74676	0,746760
200000	0.746824	–	–	149366	0,746830

5.2. táblázat. Elutasítási módszer eredményei pszeudovéletlen és Halton pontokkal

Az eredményekben látható, hogy a pontok növelésével egyre közelebbi eredményt kaptunk, azonban pszeudovéletlen pontok esetén 100000 pont beszórásával is 10^{-4} pontosságú közelítést adott. Halton-pontokkal ugyanazt a pontosságot csak 200000 pont beszórásával értük el, így lassabban konvergál a pontos értékhez.

Most nézzük meg, milyen eredményeket ad a mintavételezés a pszeudovéletlen számok és Halton-pontok esetén különböző n-re:

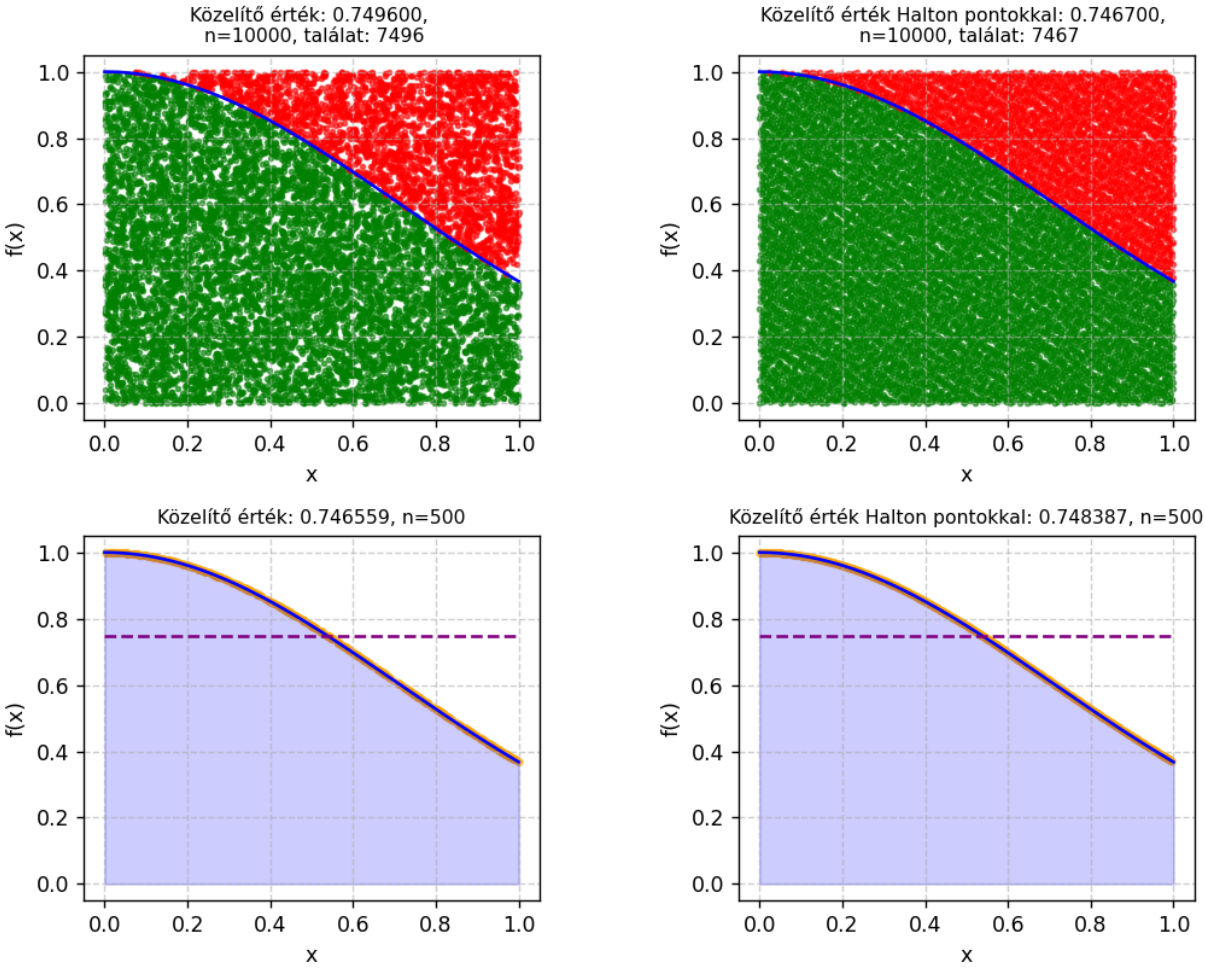
Pontos érték	Közelítő érték	n
1000	0.746824	0,748191
5000	0.746824	0,747188
10000	0.746824	0,746863
20000	0.746824	0,746796

5.3. táblázat. Mintavételezés Monte-Carlo módszer pszeudovéletlen számokkal

Ez a kísérlet érdekes eredményeket adott, mivel a módszer már 10000 pont beszórásával 10^{-4} pontossággal adott becslést. 10000-nél több pont beszórásával bár lassult a konvergencia, mégis elmondható, hogy egyre pontosabb volt. Ugyanezt a módszert Halton-pontokra alkalmazva azt látjuk, hogy bár lassan konvergál, mégis jól megfigyelhető, hogy a generált számok növelésével a pontosság is növekszik:

n	Pontos érték	Közelítő érték
1000	0.746824	0.747602
5000	0.746824	0.747039
10000	0.746824	0.746932
20000	0.746824	0.746878
50000	0.746824	0.746851
100000	0.746824	0.746837
200000	0.746824	0.746831
300000	0.746824	0.746829

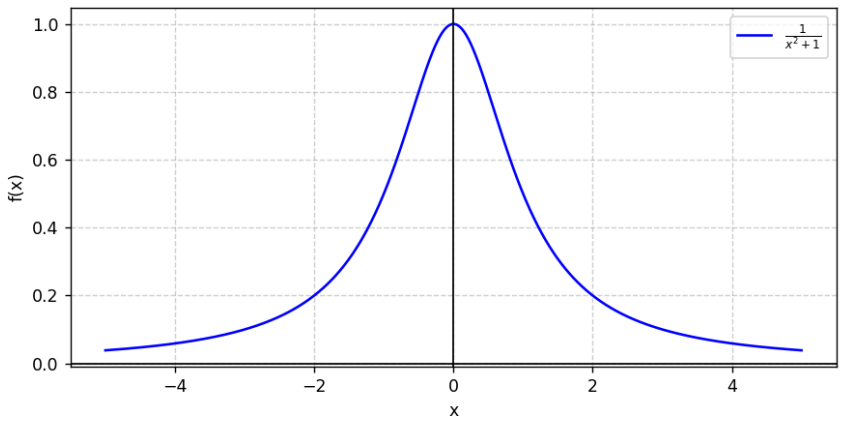
5.4. táblázat. Halton-pontokra alkalmazott mintavételezéses Monte-Carlo közelítések



5.13. ábra. A Monte-Carlo módszerek alkalmazása

5.2. példa Figyeljük meg a módszerek működését egy másik példán. Adjunk becslést a következő határozott integrál értékére:

$$\int_{-5}^{5} f(x)dx = \int_{-5}^{5} \frac{1}{1+x^2}dx$$



5.14. ábra. Az f függvény ábrázolása

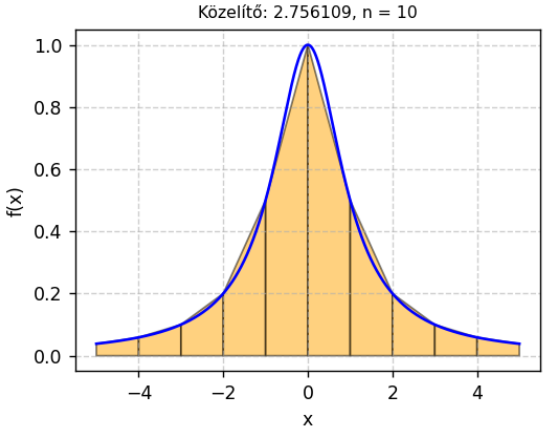
Mivel ez egy táblázati integrál, tudjuk integrálni, és alkalmazhatjuk a Newton-Leibniz formulát:

$$\int_{-5}^{5} \frac{1}{1+x^2} dx = [arctg(x)]_{-5}^{5} = arctg(5) - arctg(-5) = 2artcg(5) \approx 2.746802$$

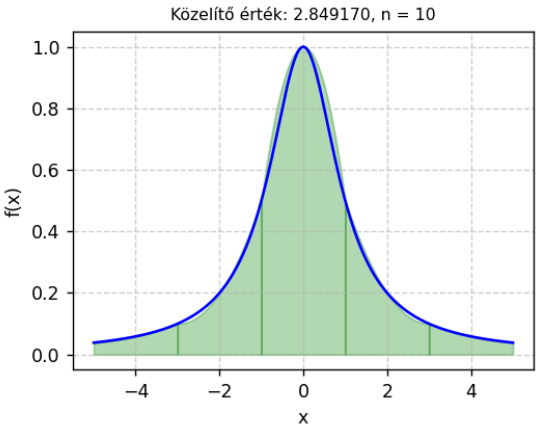
Alkalmazzuk a trapéz szabályt és a Simpson-módszert az integrál közelítésére. A kapott eredményeket a 5.5.-ös táblázat tartalmazza. A táblázatban látható, hogy a Simpson-módszer gyorsabban közelít a pontos értékhez, és ad pontos közelítést, míg a trapéz szabály 100 részintervallumra való osztás esetén is távolabb áll a pontos közelítéstől.

n	Pontos érték	Közelítő érték trapéz szabállyal	Közelítő érték Simpson-módszerrel
2	2.746802	5.192308	6.794872
10	2.746802	2.756109	2.849170
20	2.746802	2.746208	2.742098
30	2.746802	2.746528	2.746799
40	2.746802	2.746648	2.746802
50	2.746802	2.746703	-
100	2.746802	2.746777	-

5.5. táblázat. Trapézszabály és Simpson-módszer összehasonlítása a $\int_{-5}^{5} 1/(1+x^2)dx$ határozott integrál közelítésére



5.15. ábra. A trapéz szabály alkalmazása



5.16. ábra. A Simpson-módszer alkalmazása

Alkalmazzum az elutasítási Monte-Carlo módszert pszeudovéletlen és Halton pontokra. A négyszög, amit most beszórunk pontokkal a $[-5;5] \times [0;1]$, mivel a $[-5;5]$ intervallumon értelmezzük a függvényt, és a legmagasabb érték, amit felvesz, az az 1. A terület nagysága így $(5-(-5)) \cdot (1-0) = 10 n.e.$. Az eredményekből azt tudjuk következtetésként levonni, hogy a pszeudovéletlen pontokra alkalmazott elutasítási néhol bár jobbnak bizonyul, mintha a Halton-pontokra alkalmaznánk, összességében a Halton-pontokkal közelebbi becslést kapunk.

n	Pontos érték	Pszeudovéletlen pontok		Halton-pontok	
		n_t	közelítés	n_t	közelítés
1000	2.746802	275	2.750000	276	2.760000
5000	2.746802	1373	2.746000	1372	2.744000
10000	2.746802	2745	2.745000	2755	2.755000
50000	2.746802	13737	2.746600	13738	2.747000
100000	2.746802	27471	2.747100	27481	2.748100
500000	2.746802	137335	2.746700	137335	2.746700
1000000	2.746802	274667	2.746670	274667	2.746700
2000000	2.746802	549370	2.746850	549362	2.746810

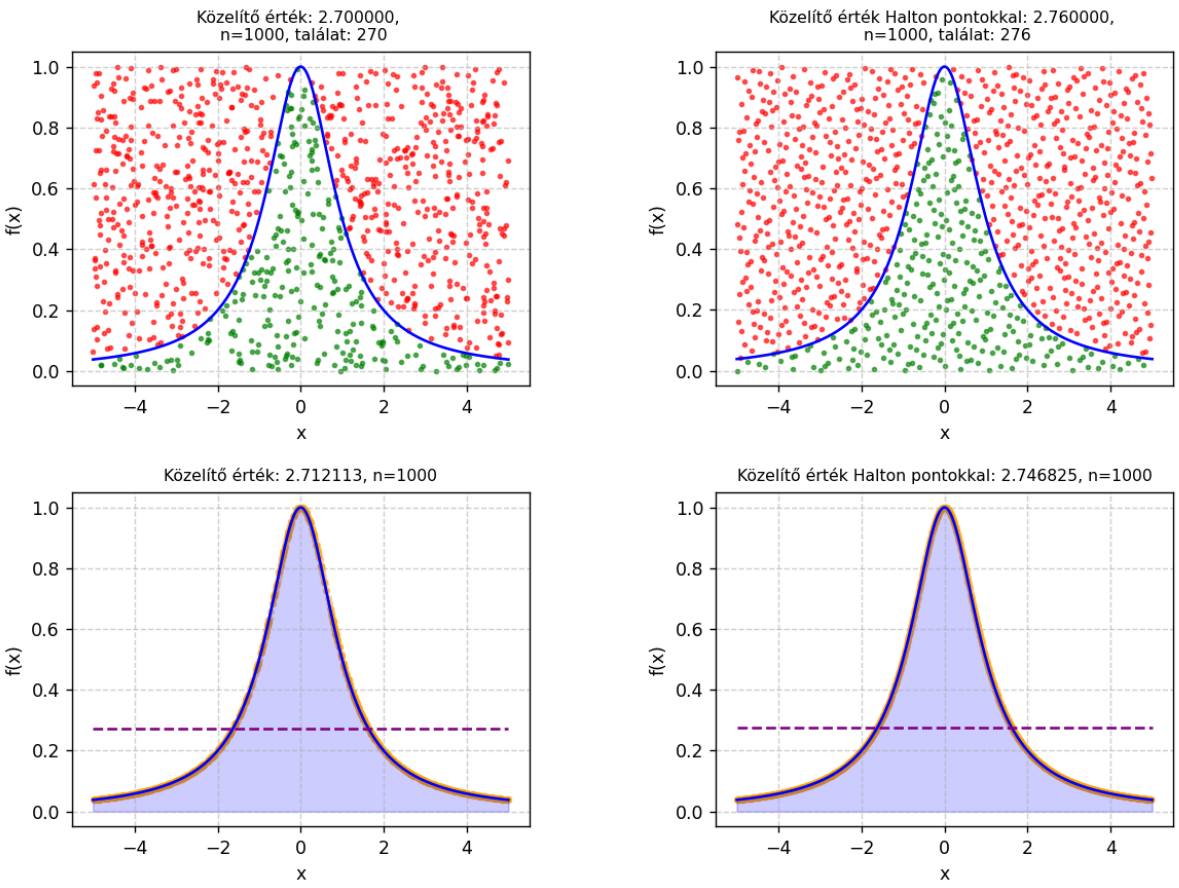
5.6. táblázat. Elutasítási módszer eredményei pszeudovéletlen és Halton-pontokkal

Alkalmazzuk a mintavételezési módszert a közelítéshez szintén pszeudovéletlen és Halton-pontokkal.

n	Pontos érték	Pszeudovéletlen számok	Halton-pontok
1000	2.746802	2.746779	2.746825
5000	2.746802	2.752086	2.746860
10000	2.746802	2.748308	2.746801
50000	2.746802	2.746353	2.746802
100000	2.746802	2.745194	-
500000	2.746802	2.746980	-
1000000	2.746802	2.746809	-

5.7. táblázat. Mintavéletezési módszer alkalmazása pszeudovéletlen és Halton-pontokkal

Pszeudovéletlen pontokra alkalmazva a módszert 1 000 000 szám generálásával kaptuk a legközelebbi becslést, azonban Halton-pontokat alkalmazva már 50 000-nél pontos becslést kapunk.

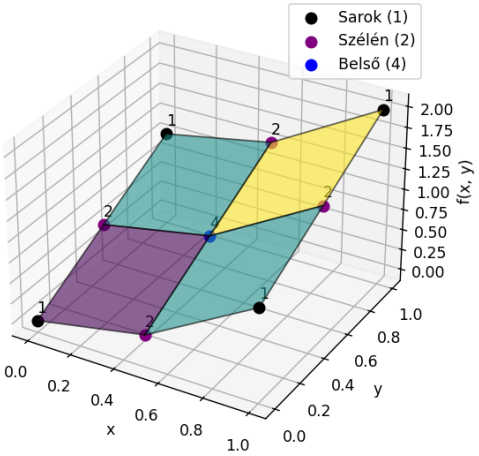


5.17. ábra. A Monte-Carlo módszerek alkalmazása

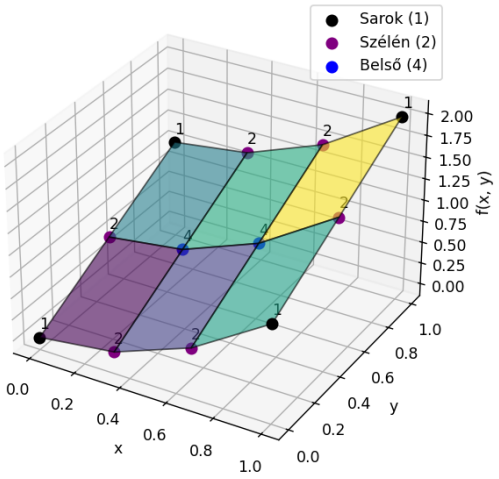
5.6. A trapéz szabály alkalmazása kettős integrálok becslésére

A második fejezetben már beszéltünk arról, hogy hogyan lehet alkalmazni a trapéz szabályt kettős integrálok esetén. Abban különbözik az egyváltozós integrálközelítéstől, hogy a trapéz szabályt kétszer alkalmazzuk egymás után, először az egyik, azt követően pedig a másik változó szerint.

Beszéltünk a közelítő összeg súlyozásáról is. A sarkon lévő pontok csak egy-egy hasáb térfogatának meghatározásában játszanak szerepet, így a közelítő összegben is csak egyes szorzóval szerepelnek. A szélen lévő, de nem sarok kövek hasonló okból kettes, a belső pontok pedig négyes szorzóval lép fel. Mint azt már korábban említettük, a két intervallumot, melyeken a függvényt értelmezzük, különböző számú részintervallumokra is felbonthatjuk. Mivel egymás után alkalmazzuk egyszer az egyik intervallumra, azt követve pedig a másikra, az összeget egy $hk/2$ törttel kell megszorozni, ahol h és k a részintervallumok hossza.



5.18. ábra. Pontok súlya trapéz módszer alkalmazásakor 2x2-es felosztás esetén



5.19. ábra. Pontok súlya trapéz módszer alkalmazásakor 3x2-es felosztás esetén

Vegyük példának az $f = x^2 + y$ függvényt a $[0;1] \times [0;1]$ tartományon. Ezt a függvényt analitikusan is könnyen integrálhatjuk:

$$\int_0^1 \int_0^1 f(x,y)dydx = \int_0^1 \int_0^1 (x^2+y)dydx = \int_0^1 \left[x^2 y + \frac{y^2}{2}\right]_0^1 dx$$
$$= \int_0^1 \left(x^2 + \frac{1}{2}\right) dx = \left[\frac{x^3}{3} + \frac{1}{2}x\right]_0^1 = \frac{1}{3} + \frac{1}{2} = \frac{5}{6} \approx 0.833333$$

A trapéz szabályt alkalmazva adjunk becslést az integrál értékére. Először osszuk fel az x és y értelmezési tartományait két-két részre, így az osztópontok mindkét esetben a 0, 0.5 és 1 számok, a részintervallumok hossza $h = 0.5$. Először alkalmazzuk a belső integrálra a trapéz szabályt, úgy tekintve a függvényre, mint egy egyváltozós függvényre, ahol y a változó, x pedig konstans. Két pontra a trapéz szabály:

$$\int_0^1 f(x,y)dy \approx \frac{0.5}{2}\left(f(x,0) + 2f(x,0.5) + f(x,1)\right)$$

Most az $\int_0^1 f(x,y)dy$ helyére pótoljuk az imént kapott kifejezést:

$$\int_0^1 \int_0^1 f(x,y)dydx \approx \int_0^1 \left[\frac{0.5}{2}\left(f(x,0) + 2f(x,0.5) + f(x,1)\right)\right] dx$$
$$= \frac{0.5}{2}\left(\int_0^1 \left(f(x,0) + 2f(x,0.5) + f(x,1)\right) dx\right)$$
$$= \frac{0.5}{2}\left(\int_0^1 f(x,0)dx + 2\int_0^1 f(x,0.5)dx + \int_0^1 f(x,1)dx\right)$$

Most alkalmazzuk újra a trapéz szabályt a három x változó szerinti integrálra:

$$\int_0^1 f(x,0)dx \approx \frac{0.5}{2}\left(f(0,0) + 2f(0.5,0) + f(1,0)\right)$$
$$\int_0^1 f(x,0.5)dx \approx \frac{0.5}{2}\left(f(0,0.5) + 2f(0.5,0.5) + f(1,0.5)\right)$$
$$\int_0^1 f(x,1)dx \approx \frac{0.5}{2}\left(f(0,1) + 2f(0.5,1) + f(1,1)\right)$$

Ezeket visszahelyettesítve:

$$\frac{0.5}{2}\left(\int_0^1 f(x,0)dx + 2\int_0^1 f(x,0.5)dx + \int_0^1 f(x,1)dx\right) \approx$$

$$\frac{0.5}{2}\left(\frac{0.5}{2}\left(f(0,0)+2f(0.5,0)+f(1,0)\right) + 2\cdot\frac{0.5}{2}\left(f(0,0.5)+2f(0.5,0.5)+f(1,0.5)\right) + \right.$$

$$\left. + \frac{0.5}{2}\left(f(0,1)+2f(0.5,1)+f(1,1)\right)\right) = \frac{0.5}{2}\cdot\frac{0.5}{2}\Bigg(f(0,0)+2f(0.5,0)+f(1,0)$$

$$+2f(0,0.5)+4f(0.5,0.5)+2f(1,0.5)+f(0,1)+2f(0.5,1)+f(1,1)\Bigg)$$

$$\approx 0.875$$

A módszer hasonlóképp működik nagyobb felosztás esetén is, vagy abban az esetben, ha a két változó tartományát különböző számú részintervallumokra bontjuk. Bontsuk az x változó értelmezési tartományát 3, az y változó értelmezési tartományát 2 részintervallumra. A részintervallumok hossza $h = 1/3$ és $k = 0.5$, az osztópontok pedig $0,\quad 1/3,\quad 2/3,\quad 1$ és $0,\quad 0.5,\quad 1$. Írjuk fel a trapéz szabályt először y szerint. Ez ugyanaz lesz, mint az előző esetben.

$$\int_0^1 f(x,y)dy \approx \frac{0.5}{2}\left(f(x,0)+2f(x,0.5)+f(x,1)\right)$$

A levezetés az előzőhöz hasonló.

$$\int_0^1\int_0^1 f(x,y)dydx \approx \int_0^1 \left[\frac{0.5}{2}\left(f(x,0)+2f(x,0.5)+f(x,1)\right)\right]dx$$

$$= \frac{0.5}{2}\left(\int_0^1 f(x,0)dx + 2\int_0^1 f(x,0.5)dx + \int_0^1 f(x,1)dx\right)$$

Alkalmazzuk a trapéz szabályt az x változó szerint:

$$\int_0^1 f(x,0)dx \approx \tfrac{\frac{1}{3}}{2}\left(f(0,0)+2f\left(\frac{1}{3},0\right)+2f\left(\frac{2}{3},0\right)+f(1,0)\right)$$

$$\int_0^1 f(x,0.5)dx \approx \tfrac{\frac{1}{3}}{2}\left(f(0,0.5)+2f\left(\frac{1}{3},0.5\right)+2f\left(\frac{2}{3},0.5\right)+f(1,0.5)\right)$$

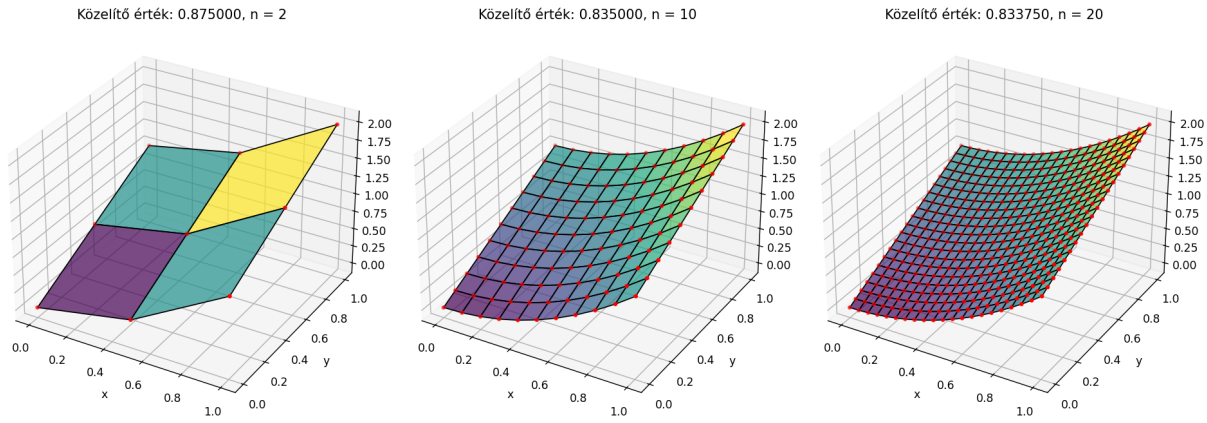
$$\int_0^1 f(x,1)dx \approx \tfrac{\frac{1}{3}}{2}\left(f(0,1)+2f\left(\frac{1}{3},1\right)+2f\left(\frac{2}{3},1\right)+f(1,1)\right)$$

Ha az integrálok helyeire a kapott kifejezéseket visszahelyettesítjük, és felbontjuk a zárójeleket, megfigyelhetjük, hogy melyik pontokat milyen súlyokkal számítja be.

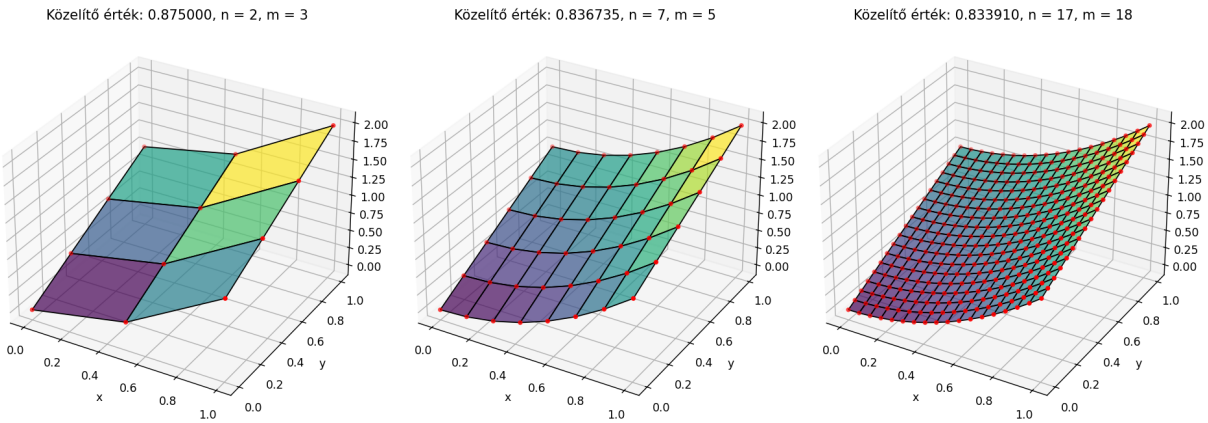
$$\frac{0.5}{2}\left(\int_0^1 f(x,0)dx + 2\int_0^1 f(x,0.5)dx + \int_0^1 f(x,1)dx\right) \approx$$

$$\frac{0.5}{2}\left(\frac{1}{6}\left(f(0,0)+2f\left(\frac{1}{3},0\right)+2f\left(\frac{2}{3},0\right)+f(1,0)\right)\right.$$
$$+2\cdot\frac{1}{6}\left(f(0,0.5)+2f\left(\frac{1}{3},0.5\right)+2f\left(\frac{2}{3},0.5\right)+f(1,0.5)\right)$$
$$\left.+\frac{1}{6}\left(f(0,1)+2f\left(\frac{1}{3},1\right)+2f\left(\frac{2}{3},1\right)+f(1,1)\right)\right)$$
$$=\frac{0.5}{2}\cdot\frac{1}{6}\left(f(0,0)+f(0,1)+f(1,0)+f(1,1)+2f(0,0.5)+\right.$$
$$2f\left(\frac{1}{3},0\right)+2f\left(\frac{2}{3},0\right)+2f\left(\frac{1}{3},1\right)+2f\left(\frac{2}{3},1\right)+2f(1,0.5)$$
$$\left.+4f\left(\frac{1}{3},0.5\right)+4f\left(\frac{2}{3},0.5\right)\right)\approx 0.875$$

Az 5.20-as és 5.21-es ábrán látható, hogy különböző felbontások esetén milyen közelítést ad a trapéz szabály.



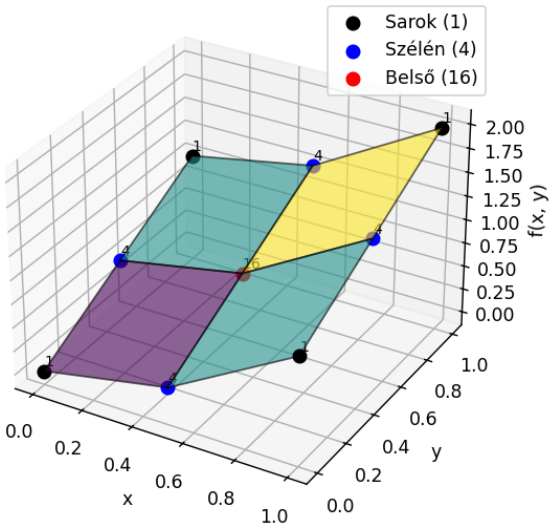
5.20. ábra. A trapéz szabály alkalmazása egyforma felosztással



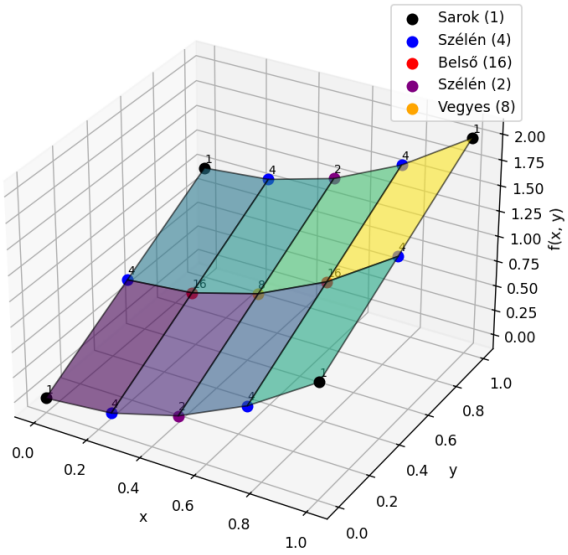
5.21. ábra. A trapéz szabály alkalmazása különböző felosztással

5.7. A Simpson-módszer alkalmazása kettős integrálok becslésére

Kettős integrálokra a Simpson-módszert hasonlóan alkalmazzuk, mint a trapéz szabályt. Először az egyik változó szerint, azután a másik változó szerint. A második fejezetben erről is beszéltünk, és megmutattuk, hogy a pontokat milyen súllyal számítja be a közelítő értékbe. A Simpson-módszert is lehet alkalmazni úgy, hogy a két változó értelmezési tartományát különböző számú részintervallumokra bontjuk, azonban figyelni kell arra,hogy a módszer csak akkor működik, ha a felosztást páros számra végezzük el.



5.22. ábra. Pontok súlya a Simpson-módszer alkalmazásakor 2x2-es felosztás esetén



5.23. ábra. Pontok súlya Simpson-módszer alkalmazásakor 4x2-es felosztás esetén

A Simpson-módszer alkalmazásához vegyük alapul a trapéz szabály bemutatásánál is használt példát:

$$\int_0^1 \int_0^1 f(x,y)dydx = \int_0^1 \int_0^1 (x^2+y)dydx \approx 0.833333$$

Először bontsuk fel mindkét intervallumot két-két részintervallumra. A részintervallumok hossza $h = (1-0)/2 = 0.5$ mindkét esetben, és az osztópontok szintén mindkét esetben a $0, 0.5, 1$ lesznek. Alkalmazzuk a Simpson-módszert a belső integrálra.

$$\int_0^1 f(x,y)dy \approx \frac{0.5}{3}\left(f(x,0) + 4f(x,0.5) + f(x,1)\right)$$

Most pedig a belső integrál helyett alkalmazzuk a kapott kifejezést:

$$\int_0^1 \int_0^1 f(x,y)dydx \approx \int_0^1 \frac{0.5}{3}\left(f(x,0) + 4f(x,0.5) + f(x,1)\right)dx$$
$$= \frac{0.5}{3}\int_0^1 f(x,0)dx + \frac{0.5}{3}\cdot 4\int_0^1 f(x,0.5)dx + \frac{0.5}{3}\int_0^1 f(x,1)dx$$
$$\frac{0.5}{3}\left(\int_0^1 f(x,0)dx + 4\int_0^1 f(x,0.5)dx + \int_0^1 f(x,1)dx\right)$$

Az integrálokra alkalmazzuk újra a Simpson-módszert:

$$\int_0^1 f(x,0)dx \approx \frac{0.5}{3}\left(f(0,0)+4f(0.5,0)+f(1,0)\right)$$
$$\int_0^1 f(x,0.5)dx \approx \frac{0.5}{3}\left(f(0,0.5)+4f(0.5,0.5)+f(1,0.5)\right)$$
$$\int_0^1 f(x,1)dx \approx \frac{0.5}{3}\left(f(0,1)+4f(0.5,1)+f(1,1)\right)$$

Helyettesítsük vissza a kapott kifejezéseket:

$$\int_0^1\int_0^1 f(x,y)dydx \approx \frac{0.5}{3}\left(\int_0^1 f(x,0)dx+4\int_0^1 f(x,0.5)dx+\int_0^1 f(x,1)dx\right)$$
$$\approx \frac{0.5}{3}\left(\frac{0.5}{3}\left(f(0,0)+4f(0.5,0)+f(0.5,0)\right)+4\left(\frac{0.5}{3}\left(f(0,0.5)+4f(0.5,0.5)+f(1,0.5)\right)\right)\right.$$
$$\left.+\frac{0.5}{3}\left(f(0,1)+4f(0.5,1)+f(1,1)\right)\right)=\frac{0.5}{3}\cdot\frac{0.5}{3}\left(f(0,0)+4f(0.5,0)+f(0.5,0)\right.$$
$$\left.+4f(0,0.5)+16f(0.5,0.5)+4f(1,0.5)+f(0,1)+4f(0.5,1)+f(1,1)\right)\approx 0.833333$$

Mivel egy másodfokú függvényre alkalmaztuk a módszert, így el is várható, hogy gyorsan konvergáljon a pontos értékhez. Most alkalmazzuk úgy a módszert, hogy az x értelmezési tartományát 4 részintervallumra bontjuk. Legyen $k=(1-0)/4=0.25$ a részintervallumok hossza, az osztópontok pedig 0, 0.25, 0.5, 0.75 és 1. Az y szerinti integrál továbbra is ugyanaz marad, viszont az x szerinti integrálban történnek változások:

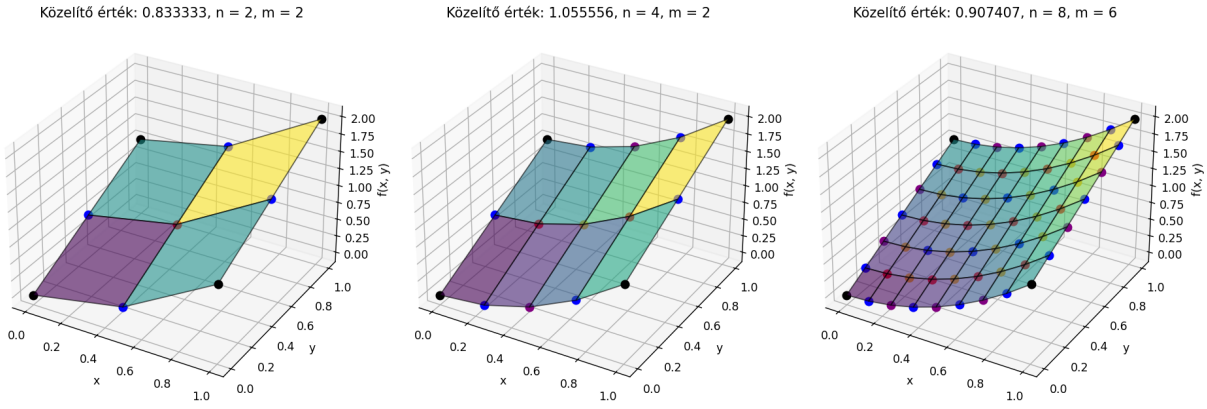
$$\int_0^1 f(x,0)dx \approx \frac{0.25}{3}\left(f(0,0)+4f(0.25,0)+2f(0.5,0)+4f(0.75,0)+f(1,0)\right)$$
$$\int_0^1 f(x,0.5)dx \approx \frac{0.25}{3}\left(f(0,0.5)+4f(0.25,0.5)+2f(0.5,0.5)+4f(0.75,0.5)+f(1,0.5)\right)$$
$$\int_0^1 f(x,1)dx \approx \frac{0.25}{3}\left(f(0,1)+4f(0.25,1)+2f(0.5,1)+4f(0.75,1)+f(1,1)\right)$$

A kapott kifejezéseket visszahelyettesítjük:

$$\int_0^1\int_0^1 f(x,y)dydx \approx \frac{0.5}{3}\left(\int_0^1 f(x,0)dx+4\int_0^1 f(x,0.5)dx+\int_0^1 f(x,1)dx\right)$$
$$\approx \frac{0.5}{3}\left(\frac{0.25}{3}\left(f(0,0)+4f(0.25,0)+2f(0.5,0)+4f(0.75,0)+f(1,0)\right)\right.$$
$$+4\cdot\frac{0.25}{3}\left(f(0,0.5)+4f(0.25,0.5)+2f(0.5,0.5)+4f(0.75,0.5)+f(1,0.5)\right)$$
$$\left.+\frac{0.25}{3}\left(f(0,1)+4f(0.25,1)+2f(0.5,1)+4f(0.75,1)+f(1,1)\right)\right)$$

$$= \frac{0.5}{3} \cdot \frac{0.25}{3} \Bigg(f(0,0) + 4f(0.25,0) + 2f(0.5,0) + 4f(0.75,0) + f(1,0)$$
$$+4f(0,0.5) + 16f(0.25,0.5) + 8f(0.5,0.5) + 16f(0.75,0.5) + 4f(1,0.5)$$
$$+f(0,1) + 4f(0.25,1) + 2f(0.5,1) + 4f(0.75,1) + f(1,1) \Bigg) \approx 1.055556$$

A zárójelek felbontása során megfigyelhetjük, hogy melyik pontot milyen súllyal számítjuk be, és miért. Az eredmény sokkal rosszabb, mint a 2×2-es felosztás esetén, elég távol áll a valós értéktől. Ha az egyik irányban kevés pont van, a Simpson-súlyozás torzul. Az y irányban 3 pont (2 szakasz) túl kevés, így nem tudja jól kompenzálni az x iránybeli pontosabb felbontást, és rosszabb eredményt kapunk, mint egy „durvább, de szimmetrikus” felosztással.



5.24. ábra. A Simpson-módszer alkalmazása különböző felosztással

5.8. A Monte-Carlo módszerek alkalmazása kettős integrálok becslésére

A Monte-Carlo módszerek kinézetre sokkal barátságosabbak többdimenziós integrálok esetén, mint a numerikus módszerek. A módszer alkalmazása során a műveletek száma a dimenziótól függetlenül nem változik. Elutasítási módszer alkalmazása során a pontokat nem egy sík területen, hanem a térben generáljuk. Meghatározzuk annak a térbeli tartománynak a térfogatát, mely magába foglalja az integrálandó függvényt, és azt beszorozzuk a találatok és az összes pont számának arányával.

Példának itt is vegyük a már korábbiakban is használt integrált a módszerek bemutatásához és kipróbálásához.

$$\int_0^1 \int_0^1 f(x,y)dydx = \int_0^1 \int_0^1 (x^2 + y)dydx \approx 0.833333$$

Az $f(x,y)$ függvény értékei a $[0;1] \times [0;1]$ tartományon legfeljebb 2-ig terjednek, így a teljes felület a $[0;1] \times [0;1] \times [0;2]$ téglatestben helyezkedik el. A téglatest térfogata $(1-0) \cdot (1-0) \cdot (2-0) = 2k.e.$. A téglatestet szórjuk be különböző számú pontokkal, és nézzük meg az eredményt.

$$\int_0^1 \int_0^1 f(x,y)dydx \approx 2 \cdot \frac{417}{1000} = 0.834000, \quad n = 1000, \quad n_t = 417$$
$$\int_0^1 \int_0^1 f(x,y)dydx \approx 2 \cdot \frac{2083}{5000} = 0.833200, \quad n = 5000, \quad n_t = 2083$$
$$\int_0^1 \int_0^1 f(x,y)dydx \approx 2 \cdot \frac{4166}{10000} = 0.833200, \quad n = 10000, \quad n_t = 4166$$
$$\int_0^1 \int_0^1 f(x,y)dydx \approx 2 \cdot \frac{20838}{50000} = 0.833520, \quad n = 50000, \quad n_t = 20838$$
$$\int_0^1 \int_0^1 f(x,y)dydx \approx 2 \cdot \frac{41643}{100000} = 0.832860, \quad n = 100000, \quad n_t = 41643$$
$$\int_0^1 \int_0^1 f(x,y)dydx \approx 2 \cdot \frac{208339}{500000} = 0.833356, \quad n = 500000, \quad n_t = 208339$$

Ezeket az eredményeket pszeudovéletlen pontok alkalmazásával kaptuk. Most alkalmazzunk Halton-pontokat.

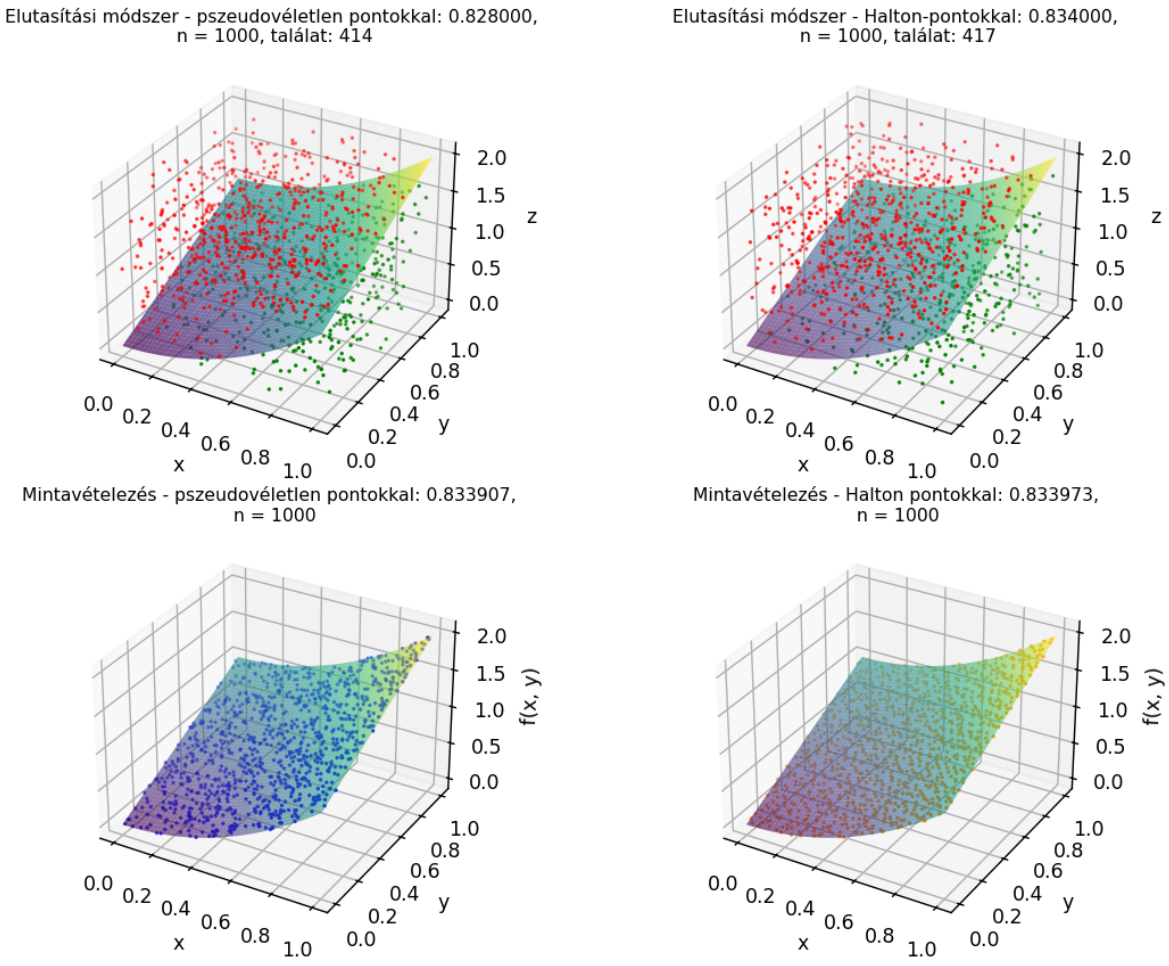
$$\int_0^1 \int_0^1 f(x,y)dydx \approx 2 \cdot \frac{417}{1000} = 0.834000, \quad n = 1000, \quad n_t = 417$$
$$\int_0^1 \int_0^1 f(x,y)dydx \approx 2 \cdot \frac{2083}{5000} = 0.833200, \quad n = 5000, \quad n_t = 2083$$
$$\int_0^1 \int_0^1 f(x,y)dydx \approx 2 \cdot \frac{4168}{10000} = 0.833600, \quad n = 10000, \quad n_t = 4168$$
$$\int_0^1 \int_0^1 f(x,y)dydx \approx 2 \cdot \frac{20833}{50000} = 0.833320, \quad n = 50000, \quad n_t = 20833$$
$$\int_0^1 \int_0^1 f(x,y)dydx \approx 2 \cdot \frac{41666}{100000} = 0.833320, \quad n = 100000, \quad n_t = 41666$$
$$\int_0^1 \int_0^1 f(x,y)dydx \approx 2 \cdot \frac{208335}{500000} = 0.833340, \quad n = 500000, \quad n_t = 208335$$

Halton-pontokat alkalmazva nagyobb pontszám esetén pontosabb értéket kapunk, mint pszeudovéletlen pontokkal.

A mintavételezési módszer is hasonlóan működik, mint egyváltozós integrálok esetén. A különbség csupán annyi, hogy most nem csak egy szakaszon, az egyik változó értelmezési tartományán generálunk számokat, hanem az x és y változók értelmezési tartományán, tehát a $[0;1][0;1]$ területen. A generált pontokon kiértékeljük a függvényt, átlagot számolunk belőlük, és beszorozzuk a terület nagyságával, ami ez esetben $(1-0)\cdot(1-0) = 1$. Ebben az esetben a kapott közelítések lényegében a számolt átlag értékével egyezik meg. Egy táblázatban foglaljuk össze mind a pszeudovéletlen, mind pedig a Helton-pontok alkalmazásával kapott eredményeket.

n	Pszeudovéletlen pontok	Halton-pontok
1000	0.831666	0.833334
5000	0.832878	0.833329
10000	0.833599	0.833352
50000	0.833098	0.833331
100000	0.833284	0.833333
500000	0.833419	-

5.8. táblázat. Mintavéletezési módszer alkalmazása kettős integrál közelítésére pszeudovéletlen és Halton-pontokkal



5.25. ábra. A Monte-Carlo módszerek alkalmazásának egy véletlenszerű esete

5.9. A numerikus és Monte-Carlo módszerek összehasonlítása kettős integrálok közelítésére

Az összehasonlítás során a trapéz szabály és Simpson-módszer esetében mindkét változó értelmezési tartományára megegyező felbontást fogunk alkalmazni.

5.3. példa Először vegyünk példának egy egyszerűen integrálható, hirtelen csökkenő függvényt:

$$\int_0^2 \int_{0.1}^4 \frac{x}{y^2} dydx.$$

Integráljuk az f függvényt az adott tartományon, és határozzuk meg az értékét.

$$\int_{0.1}^4 \frac{x}{y^2} dy = x \int_{0.1}^4 \frac{1}{y^2} dy = x \left[-\frac{1}{y} \right]_{0.1}^4 = x \left[-\frac{1}{4} + \frac{1}{0.1} \right] = x \cdot \frac{39}{4} = \frac{39x}{4}$$

$$\int_0^2 \int_{0.1}^4 \frac{x}{y^2} dydx = \int_0^2 \frac{39x}{4} dx = \frac{39}{4} \int_0^2 x dx = \frac{39}{4} \cdot \left[\frac{x^2}{2} \right]_0^1 = \frac{39}{4} \cdot \left(\frac{2^2}{2} - \frac{0^2}{2} \right) = \frac{39}{4} \cdot 2 = 19.5$$

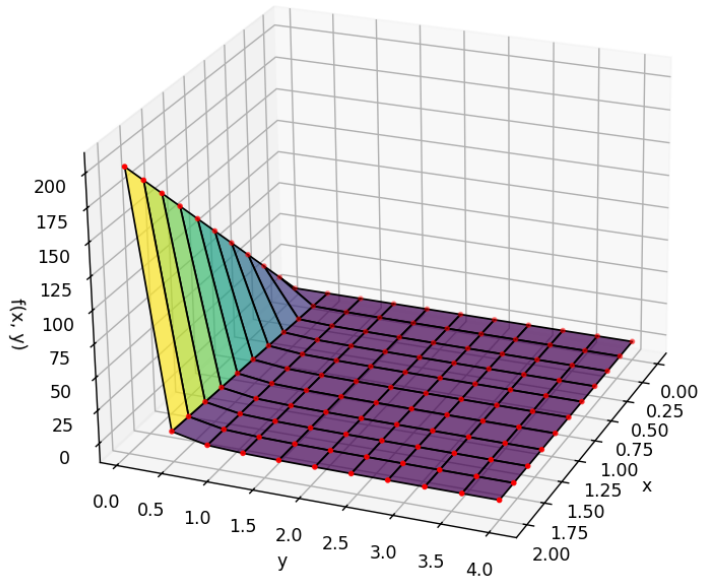
Ennek a határozott integrálnak könnyen és egyszerűen meg tudtuk határozni az értékét. Most figyeljük meg, milyen közelítést adnak a numerikus és Monte-Carlo módszerek.

Először alkalmazzuk a trapéz szabályt és a Simpson-módszert, és nézzük meg, különböző felosztásokra milyen közelítést adnak rá.

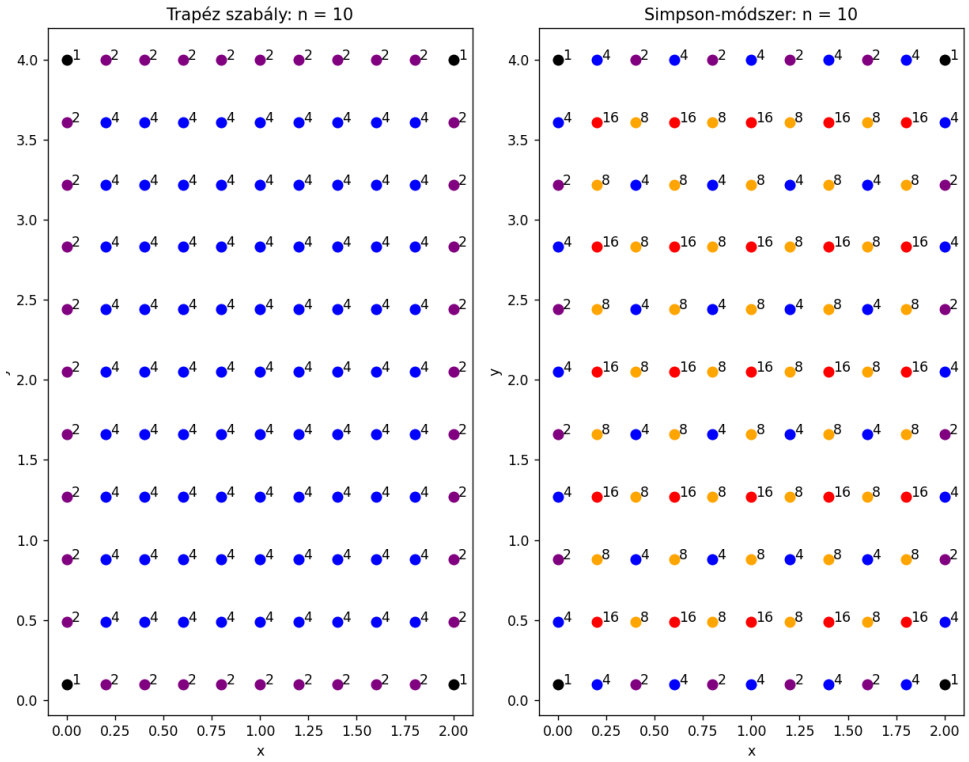
Pontos érték	**n**	**Trapéz szabály**	**Simpson-módszer**
19.500000	10	44.596003	32.447399
19.500000	20	28.479712	23.107615
19.500000	50	21.345041	19.877731
19.500000	100	19.992973	19.542283
19.500000	150	19.722418	19.503421
19.500000	200	19.625809	19.500141
19.500000	250	19.580731	19.500072
19.500000	300	19.556144	19.500030
19.500000	350	19.541263	19.500015
19.500000	400	19.531627	19.500007
19.500000	450	19.525445	19.500002
19.500000	500	19.520255	19.500001

5.9. táblázat. A közelítő értékek összehasonlítása a Trapéz szabály és a Simpson-módszer esetén

A numerikus módszerek nehezen kezelik, ha a függvény értéke hirtelen változik, ezért láthatjuk azt, hogy a módszerek lassan konvergálnak a pontos értékhez finomabb felbontás esetén is. Az 5.26.-os ábrán látható is, hogy milyen hirtelen változik a függvény értéke.



5.26. ábra. A függvényhez illesztett rácspontok a numerikus módszerek alkalmazása esetén 10×10-es felbontásban

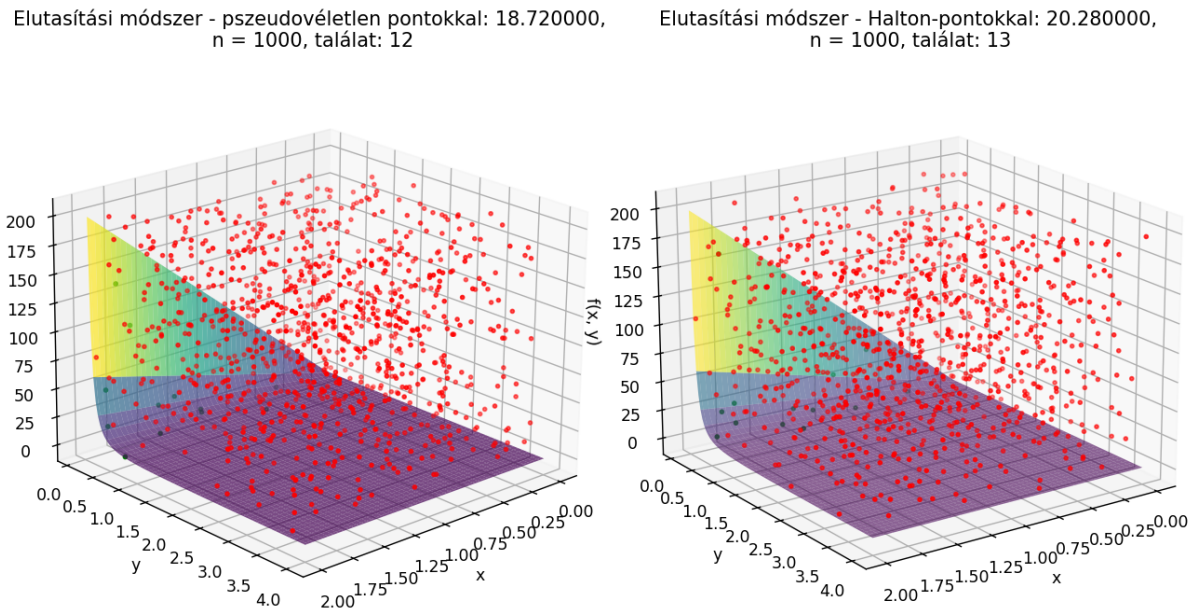


5.27. ábra. A függvényhez illesztett rácspontok súlyai numerikus módszerek alkalmazása esetén 10×10-es felbontásban

Alkalmazzuk a Monte-Carlo elutasítási módszert. A téglatest térfogata, melyben a függvény elhelyezhető $(2-0) \cdot (4-0.1) \cdot (200-0) = 1560 k.e.$. Az eredményeket az 5.10.-es táblázat mutatja be.

Pontos érték	n	Pszeudovéletlen pontok		Halton-pontok	
		n_t	közelítés	n_t	közelítés
19.500000	1000	12	18.720000	13	20.280000
19.500000	5000	64	19.968000	63	19.656000
19.500000	10000	125	19.500000	126	19.656000
19.500000	50000	627	19.562400	624	19.468800
19.500000	100000	1249	19.484400	1250	19.500000
19.500000	500000	6247	19.190640	6250	19.500000
19.500000	1000000	12498	19.496880	-	-
19.500000	2000000	24999	19.499220	-	-
19.500000	3000000	37504	19.502080	-	-

5.10. táblázat. Elutasítási módszer által kapott eredmények pszeudovéletlen és Halton-pontokkal

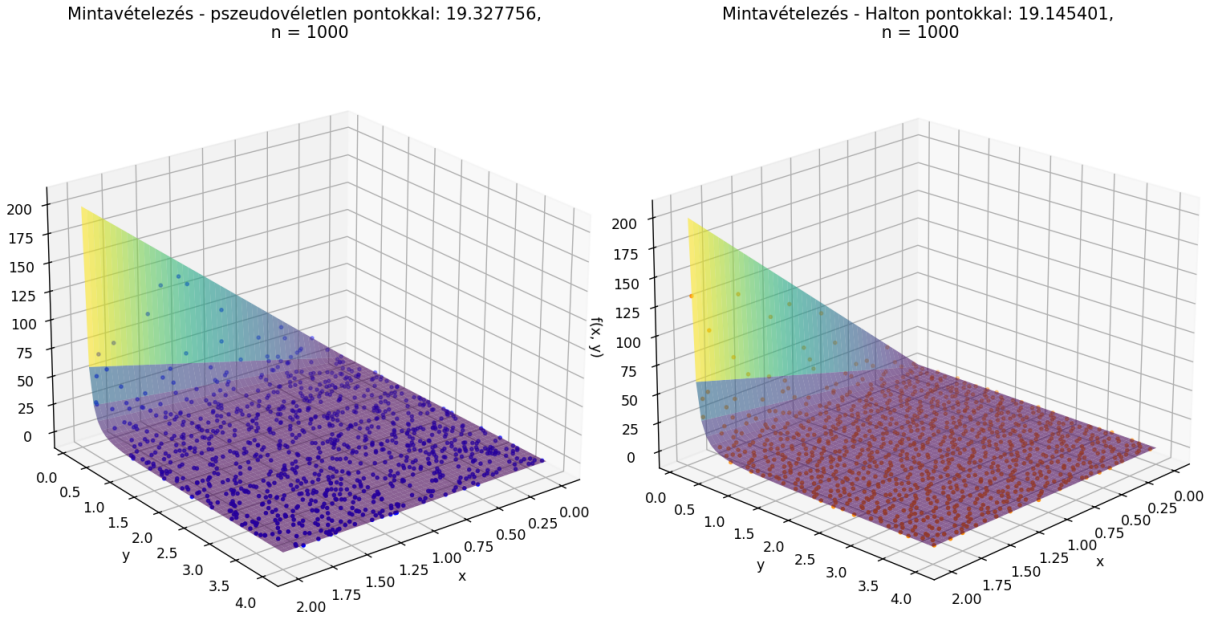


5.28. ábra. Az elutasítási módszer alkalmazásának egy véletlenszerű esete

Alkalmazzuk a mintavételezési módszert különböző pontokra. A terület, melyen pontokat generálunk $(2-0)\cdot(4-0.1) = 7.8n.e.$. A kapott eredményeket az 5.11.-es táblázat szemlélteti.

Pontos érték	n	Pszeudovéletlen pontok	Halton-pontok
19.500000	1000	19.231028	19.515057
19.500000	5000	19.510415	19.502888
19.500000	10000	19.611896	19.499729
19.500000	50000	19.462835	19.499342
19.500000	100000	19.497015	19.500691
19.500000	500000	19.501342	19.500002
19.500000	1000000	19.496810	19.500006
19.500000	2000000	19.504429	19.488883
19.500000	3000000	19.498295	19.500002

5.11. táblázat. A mintavételezési módszer által kapott eredmények pszeudovéletlen és Halton-pontokkal



5.29. ábra. Az mintavételezési módszer alkalmazásának egy véletlenszerű esete

Ebben a példában már jobban látható a Monte-Carlo módszer hatékonysága, mivel az elutasítási módszer Halton-pontokra való alkalmazása esetén már a tér 100000 pont generálásával is pontos értéket kaptunk a legtöbb esetben. A trapéz módszer alkalmazásával még 500 részintervallum esetén is rosszabb eredményt kaptunk, mint a Monte-Carlo módszerekkel, és a Simpson-módszer is lassan konvergált a pontos értékhez.

5.4. példa A következő példában nézzük meg a következő határozott integrált:

$$\int_{-1}^{1}\int_{-1}^{1} f(x,y)dydx = \int_{-1}^{1}\int_{-1}^{1} \sqrt{4-x^2-y^2}dydx$$

Ennek a függvénynek az integrálja eléggé összetett, nehezen integrálható a gyök miatt. Ilyen esetben egy numerikus becslést gyorsabban lehet adni. Erre a függvényre a Python is egy numerikus közelítést ad:

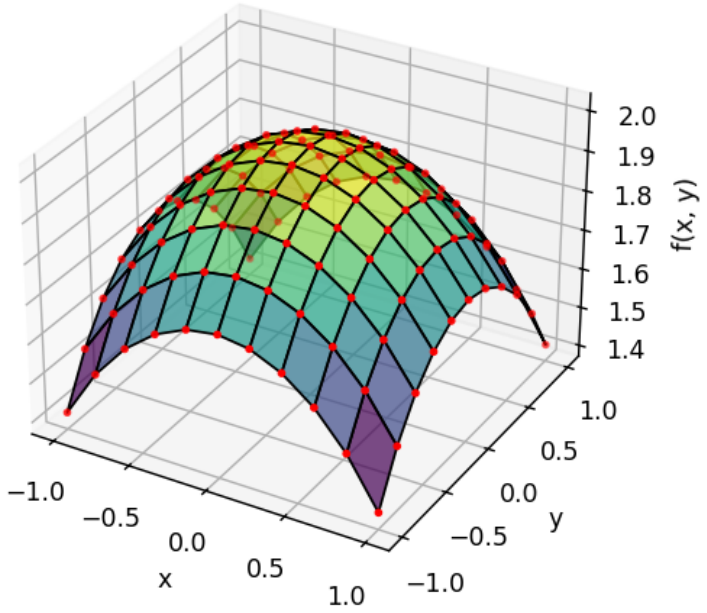
$$\int_{-1}^{1}\int_{-1}^{1} \sqrt{4-x^2-y^2}dydx \approx 7.287727$$

Nézzük meg a trapéz szabályt és a Simpson-módszert. Az eredményt az 5.12.-es táblázat foglalja össze.

Pontos érték	n	Trapéz szabály	Simpson-módszer
7.287727	10	7.271316	7.287660
7.287727	20	7.283624	7.287722
7.287727	50	7.287070	7.287727
7.287727	100	7.287563	-
7.287727	150	7.287654	-
7.287727	200	7.287686	-
7.287727	250	7.287701	-
7.287727	300	7.287709	-
7.287727	350	7.287713	-
7.287727	400	7.287717	-
7.287727	450	7.287719	-
7.287727	500	7.287720	-

5.12. táblázat. A közelítő értékek összehasonlítása a Trapéz szabály és a Simpson-módszer esetén

Látható, hogy míg a trapéz szabály csak 500 részintervallum felett ad pontos közelítést, addig a Simpson-módszer már 50-nél pontos közelítést adott.



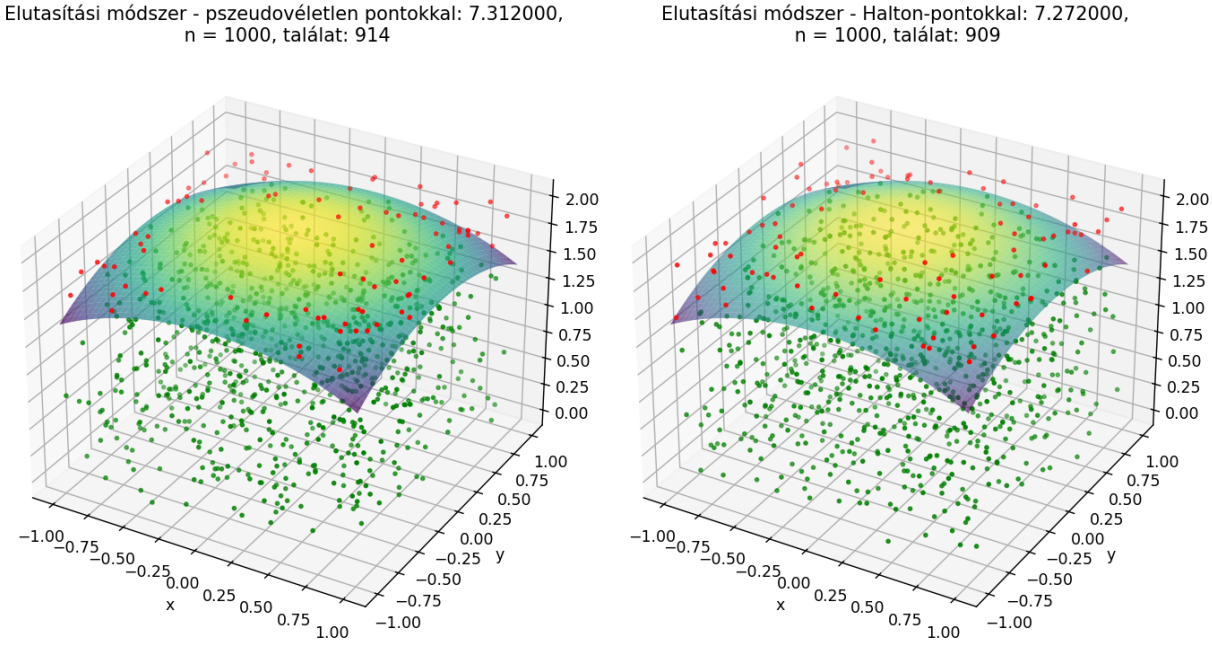
5.30. ábra. A függvényhez illesztett rácspontok a numerikus módszerek alkalmazása esetén 10×10-es felbontásban

Most alkalmazzuk az elutasítási módszert pszeudovéletlen és Halton-pontokkal. A térfogata a téglatestnek, melyen belül a pontokat generáljuk $(1-(-1)) \cdot (1-(-1)) \cdot (2-0) = 8 k.e.$.

Pontos érték	n	Pszeudovéletlen pontok		Halton-pontok	
		n_t	közelítés	n_t	közelítés
7.287727	1000	911	7.287998	911	7.287998
7.287727	5000	4554	7.286398	4555	7.287998
7.287727	10000	9110	7.287998	9110	7.287998
7.287727	50000	45546	7.287358	45548	7.287678
7.287727	100000	91097	7.287758	91097	7.287758
7.287727	500000	455473	7.287566	455481	7.287694
7.287727	1000000	910922	7.287374	910971	7.287766
7.287727	2000000	1821955	7.287818	1821930	7.287718
7.287727	3000000	2732909	7.287756	2732902	7.287737

5.13. táblázat. Elutasítási módszer által kapott eredmények pszeudovéletlen és Halton-pontokkal

Bár a pontos értéket nem sikerült előállítani, elég közeli becsléseket kaptunk.



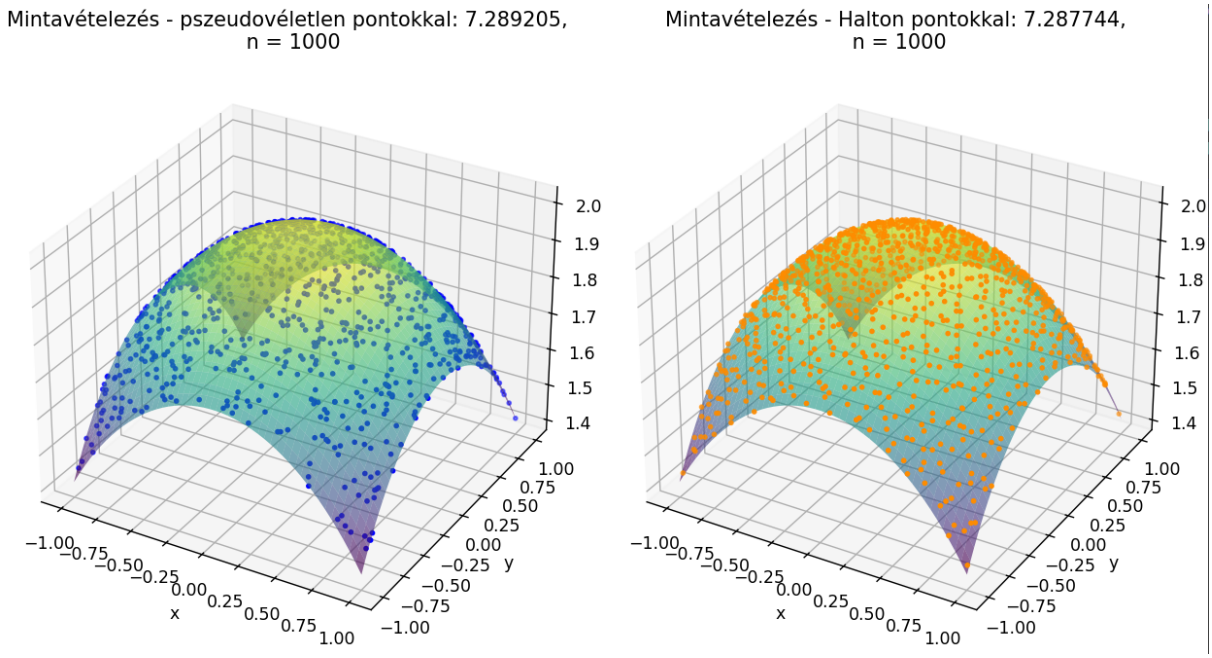
5.31. ábra. Az elutasítási módszer alkalmazásának egy véletlenszerű esete

Alkalmazzuk a mintavételezési módszert. Az eredményeket az 5.14.-es táblázat tartalmazza.

Pontos érték	n	Pszeudovéletlen pontok	Halton-pontok
19.500000	1000	7.287558	7.287689
19.500000	5000	7.287725	7.287730
19.500000	10000	7.287714	7.287718
19.500000	50000	7.287790	7.287727
19.500000	100000	7.287784	7.287727
19.500000	500000	7.287692	-
19.500000	1000000	7.287746	-
19.500000	2000000	7.287642	-
19.500000	3000000	7.287732	-

5.14. táblázat. A mintavételezési módszer által kapott eredmények pszeudovéletlen és Halton-pontokkal

Amint azt láthatjuk, Halton-pontokat használva már 50000 pontnál megkaptuk a pontos értéket, de pszeudovéletlen pontokkal is közeli becsléseket kaphatunk.



5.32. ábra. Az mintavételezési módszer alkalmazásának egy véletlenszerű esete

6. fejezet

Integrálok oktatása a középiskolákban

Az ukrán középiskolai oktatásban a matematika oktatás terén három különböző oktatási szintet határoznak meg: standard szint (alapszint), profil szint (emelt szint), illetve mélyített vagy haladó szint. A szintek közötti eltérés az elméleti háttér részleteibe való betekintés, az alkalmazások típusa és a tanulói tevékenységek komplexitása alapján különül el [19].

Alap szinten az integrálszámítás bevezetése szemléletes és intuitív módon történik. A cél elsősorban a tanulók szemléletformálása: az integrál mértani és fizikai értelmezésének megértése, illetve egyszerű alkalmazások bemutatása és alkalmazása. Az oktatás a határozott integrál fogalmára, a Newton–Leibniz-formulára, valamint a síkidomok területének meghatározására fókuszál, a feladatok e köré épülnek. A tanulók a táblázati integrálok és az alapvető integrálási szabályok alkalmazásával keresik a függvények primitív integrálját és oldanak meg feladatokat.

Emelt szinten a tanulók már részletesebben foglalkoznak az integrál fogalmának elméleti alapjaival és gyakorlati alkalmazásaival. A tananyag kitér a határozatlan integrál fogalmára, és nagyobb hangsúlyt fektet az integrálási szabályokra, s kerülnek a különféle transzformációk. A tanulók a síkidomok területének meghatározása mellett a forgástestek térfogatának kiszámításával is megismerkednek. Kulcsfontosságú szerepet kap az integrál alkalmazása a valós problémák modellezésében.

Mélyített, azaz haladó szinten a tanulók az integrál fogalmát mélyebben érintik az elmélet és az alkalmazási lehetőségek szempontjából is. Az oktatás célja nem az integrálási rutin fejlesztése, hanem a matematikai modellezésre való képesség megalapozása. Ennek megfelelően a tananyag magába foglalja a valós alkalmazási problémák megoldását is, emellett különféle módszerekkel primitív függvényeket határoznak meg, a határozott integrál mértani, fizikai értelmezésének, a síkidomok területének és a testek térfogatának kiszámításával foglalkoznak.

6.1. Numerikus és Monte-Carlo módszerek a középiskolában

Bár elsőre egy nehéz témának tűnik, mégis némely módszer egyszerűen magyarázható középiskolások számára is. Ebben a fejezetben azt fogjuk megnézni, hogy hogyan.

A trapéz szabály alkalmazásához mindössze néhány alapvető, a középiskolai oktatásban már korábban elsajátított fogalomra van szükség: a tanulóknak érteniük kell, hogyan számítható ki két pont távolsága a számegyenesen vagy a síkban, képesnek kell lenniük egy függvényt adott pontokban kiértékelni, valamint ismerniük kell a trapéz területének kiszámítási módját. Ezekre a fogalmakra támaszkodva a trapéz szabály működése szemléletesen és fokozatosan felépíthető, akár konkrét példákon keresztül is, így az eljárás nem igényel előzetes integrálszámítási ismereteket sem. A módszer jól szemléltethető, geometriailag értelmezhető, technikailag nem túl bonyolult. A módszer lehetőséget biztosít az analitikus és numerikus módszerek összehasonlí-

tására, valamint arra, hogy a diákok lássák: nem minden problémára van egzakt megoldás, néha más lehetőséghez kell folyamodnunk, és a számítógépekre bízni a munkát.

Hasonlóan az 5.1-es fejezetben bemutatott példához, a trapéz szabály középiskolás diákok számára is szemléletesen bevezethető. Egy egyszerű, számukra is integrálható példán keresztül – például egy lineáris vagy másodfokú függvény esetén – először úgy vizsgáljuk a közelítő integrál értékét, hogy az intervallumot nem bontjuk fel, hanem csak a két végpontban értékeljük ki a függvényt. Ezután az intervallumot két, majd három részre osztva is alkalmazzuk a trapéz szabályt, és összehasonlítjuk az eredményeket. Ennek során a diákok fokozatosan felismerhetik az eljárás lényegét, és közösen is megfogalmazhatják a szabály általános alakját.

A Simpson-módszer alkalmazása középiskolai szinten több szempontból is problémás. Először is, a módszer alapötlete - hogy másodfokú (parabolikus) függvényeket illesztünk három egymást követő ponton keresztül a közelítendő függvény helyett - nehezen értelmezhető azon diákok számára, akik nem rendelkeznek a polinomközelítés vagy az interpoláció fogalmának ismeretével. A középiskolai matematika tanterv általában nem tér ki ezekre a témákra, így a módszer elméleti megalapozása hiányzik a tanulók eszköztárából.

Másodszor, a Simpson-formula maga is viszonylag bonyolult:

$$\int_a^b f(x)\,dx \approx \frac{b-a}{3n}\left[f(x_0) + 4f(x_1) + 2f(x_2) + \cdots + 2f(x_{n-2}) + 4f(x_{n-1}) + f(x_n)\right],$$

ahol n páros természetes szám. A képlet súlyozási mintázata $(1, 4, 2, 4, \ldots, 2, 4, 1)$ nem intuitív, és a használatát gyakran gépies alkalmazás kíséri, a megértés helyett. Ez különösen veszélyes abban az oktatási környezetben, ahol a cél a matematikai fogalmak mélyebb szemléltetése és megértése. Összességében a Simpson-módszer tanítása általános középiskolai szinten nem célszerű.

A Monte-Carlo elutasítási módszer középiskolás tanulók számára is könnyen érthető lehet, mivel nem igényel magasabb szintű tudást. Emellett lehetőséget ad a valószínűségszámítás és a geometriai ismeretek összekapcsolására.

Az elutasítási módszer lényege, hogy a területet sok véletlenszerűen generált pont segítségével közelítjük. Ha egy egyszerűbb, jól ismert terület (például egy téglalap) belsejébe generálunk véletlenszerű pontokat, majd megszámoljuk, hány pont esik az adott görbe alá. A becsült terület az alábbi képlettel adható meg:

$$T_{\text{becsült}} = T_{\text{téglalap}} \cdot \frac{N_{\text{görbe alatt}}}{N_{\text{összes pont}}}.$$

Ez a megközelítés szemléletesen és interaktívan bemutatható akár rajzolt koordinátarendszerrel, papíron is, vagy digitálisan (pl. GeoGebra, Python, Excel), és jól kapcsolható a valószínűségi eseményekhez.

Az egyszerű mintavételezéses módszer esetén az $[a, b]$ intervallumon egyenletesen véletlenszerűen választunk ki n darab pontot. Minden pont esetén kiszámítjuk az $f(x_i)$ értékét, majd ezeket átlagoljuk:

$$\int_a^b f(x)\,dx \approx (b-a) \cdot \frac{1}{n}\sum_{i=1}^{n} f(x_i).$$

A módszer alkalmazása során nincs szükség az $f(x)$ maximumának ismeretére. Inkább az átlagszámításra és a számegyenes fogalmára épül, amelyek jól ismertek a középiskolai tananyagból.

Mindkét eljárás alkalmas arra, hogy játékos, interaktív módon, például fizikai vagy digitális szimulációk segítségével (mint a céltábla, darts vagy Python-programok) a tanulók jobban

megértsék a területközelítés lényegét, így motiválva őket a valószínűség és az analízis közötti kapcsolatok felfedezésére, felismerésére és megértésére.

A két Monte-Carlo eljárás középiskolai bemutatása számos pedagógiai előnnyel jár, ugyanakkor néhány korlátot is figyelembe kell venni. Az elutasítási módszer vizuálisan jól szemléltethető, és természetes módon kapcsolható a valószínűségszámítás fogalmához, így a tanulók könnyen megérthetik, hogy a találatok és az összes véletlenszerű pont számának aránya a keresett terület értékét közelíti. A mintavételezéses módszer egyszerű logikájával (átlagolás és intervallumszélesség szorzata) jól illeszkedik a korábban tanult matematikai fogalmakhoz. Ugyanakkor egyik módszer sem ad determinisztikus eredményt. A közelítés pontossága nagymértékben függ attól, hány mintát használunk. Ez a bizonytalanság nehezen emészthető lehet, és akadályt jelenthet a tanulók számára abban az esetben, ha kevés pontból becsülnek, és az eredmények nagy szórást mutatnak. Továbbá a szumma jelölése, a sztochasztikus gondolkodásmód, vagy a közelítés mibenléte – különösen elméleti háttér nélkül – kihívást jelenthet egyes tanulók számára.

Összességében, a módszerek megfelelő példák, illusztrációk és tanári támogatás mellett középiskolás diákok számára is érthetőek lehetnek. Segíthetnek abban, hogy jobban megértsék a függvény, a terület, az átlag és a valószínűség fogalmát, és lehetőséget adnak a matematikai modellalkotás kezdeti lépéseinek megértésére is.

A kettős integrálok bevezetése a középiskolai matematika kerettantervben nem szerepel, mivel olyan új fogalmakat és elméleti hátteret igényel, amelyek meghaladják a középiskolás tanulók számára elvárható szintet. A síkidomok területének és egyszerű testek térfogatának számítása még geometriai eszközökkel történik, az integrálszámítás csúcspontját pedig az egyváltozós függvények határozott integráljának alkalmazása jelenti. A térgeometria témakörében a tanulók megismerkednek a térbeli alakzatokkal és azok alapvető tulajdonságaival, például felszín- és térfogatszámítással, azonban a többváltozós függvények és a többdimenziós koordinátarendszerek nem szerepelnek a középiskolai tananyagban. A többváltozós függvényekkel, valamint az ehhez tartozó térbeli tartományokkal kapcsolatos számítások – így a kettős integrál – már a felsőoktatáshoz tartozik, ahol megfelelő háttérismeretekkel rendelkeznek a hallgatók.

Ennek megfelelően a numerikus vagy Monte-Carlo módszerek többváltozós változatai sem illeszthetők be közvetlenül a középiskolai tananyagba. Bár a Monte-Carlo módszerek elvi működése érdekes lehet, például bonyolult alakzatok térfogatának közelítésére, ezek inkább motivációs célból, figyelemfelkeltő elemként alkalmazhatók. Egy-egy látványos példával (például szabálytalan testek térfogatának szimulációs becslésével) lehet szemléltetni a módszer erejét, de rendszeres oktatása vagy elmélyített alkalmazása nem várható el a középiskolás diákoktól.

Összefoglaló

Munkám folyamán egyváltozós és kétváltozós határozott integrálok becslésével foglalkoztam. Erre különböző numerikus és Monte-Carlo módszereket vizsgáltam elméleti és gyakorlati szempontból egyaránt. A dolgozatom megírása folyamán tanulmányoztam az integrálok középiskolai oktatását, illetve az ukrajnai középiskolai matematikaoktatás alapján mérlegelni próbáltam, tanítható vagy esetleg bemutatható-e valamely módszer a tanulók számára.

Az első fejezetben a Monte-Carlo módszer alapgondolatával és történelmi hátterével ismerkedtem meg. A második fejezetben numerikus módszerekkel foglalkoztam. Tanulmányoztam annak szükségességét, és kiemelten vizsgáltam közülük kettőt: a trapéz szabályt és a Simpson-módszert. Ezek működését egyváltozós és kétváltozós határozott integrálokra is megismertem és alkalmaztam.

A harmadik fejezetben részletesen foglalkoztam a Monte-Carlo módszerek alkalmazásaival szintén egyváltozós és kettős határozott integrálok becslése esetén. Megvizsgáltam, hogyan lehet alkalmazni szabálytalan idomok területének becslésére is. Elméleti szinten hibaelemzéssel és olyan fejlettebb Monte-Carlo módszerekkel foglalkoztam, mint a fontossági mintavételezés és változóvezérlés.

A negyedik fejezetben ismertettem a véletlen számgenerálás alapjait és módszereit, mivel a véletlen mintavételezés kulcsfontosságú szerepet játszik a Monte-Carlo módszerek működésében. Itt a pszeudovéletlen és Halton-pontok közötti különbségeket is megfigyeltem.

Az ötödik fejezetben gyakorlati szempontból foglalkoztam a különböző módszerekkel. Először egyváltozós határozott integrálok becslésére írtam le részletesen a módszerek működését, majd két példán keresztül összehasonlítottam őket. Hasonlóan kettős integrálok esetén is egy példán keresztül részletesen szemléltettem, majd további két példán a módszerek hatékonyságát mutattam be. A módszerek bemutatásához python kódokat írtam, melyek szemléletes ábrákat készítenek, és közelítő értékeket számolnak adott példákra. Arra a következtetésre jutottam, hogy egyváltozós integrálok esetén egyszerűbb egy jó numerikus becslést adni, mint a Monte-Carlo módszert alkalmazni. Kettős integrálok esetén már a Monte-Carlo módszer is sokkal hatékonyabbnak bizonyult, bár a benne rejlő lehetőségek itt még nem minden esetben mutatkoznak meg. Az, hogy melyik módszert érdemes egy esetben alkalmazni, a függvény tulajdonságai mondják meg.

A dolgozat utolsó fejezetében foglalkoztam azzal a kérdéssel, hogy a numerikus és Monte-Carlo módszerek miként jelenhetnek meg a középiskolai oktatásban. Azt vizsgáltam meg, hogy milyen módszertani lehetőségek állnak rendelkezésre ahhoz, hogy a diákok számára érthető és élményszerű módon lehessen a módszereket átadni.

Irodalomjegyzék

[1] N. Metropolis, *The Beginning of the Monte Carlo Method*, Los Alamos Science, No. 15, pp. 125–130, 1987.

[2] Kai Nordlund, *Basics of Monte Carlo Simulations*, Helsinki, 2006.

[3] Wojciech Jarosz, *Efficient Monte Carlo Methods for Light Transport in Scattering Media*, UC San Diego, September 2008.

[4] James E. Gentle, *Random Number Generation and Monte Carlo Methods*, Springer-Verlag, New York, 2003.

[5] Greg Kochanski, *Monte Carlo Simulation*, 2005.

[6] Sabri Pllana, *History of Monte Carlo Methods*.

[7] I. M. Szobol, *A Monte-Carlo módszerek alapjai*, Budapest, 1981.

[8] Kehl Dániel, *Mont-Carlo módszerek a statisztikában*, 2012.

[9] Harald Niederreiter, *On the distribution of pseudo-random numbers generated by the linear congruential method I–III*, Springer Verlag, 1985.

[10] Faragó István, Horváth Róbert, *Numerikus Módszerek*, 2011.

[11] Laky Piroska, *Numerikus módszerek építőmérnököknek Matlab-bal*, Akadémiai Kiadó, 2020. ISBN: 978-963-454-506-4, https://mersz.hu/hivatkozas/m703nmem_137_p1#m703nmem_137_p1

[12] *International Journal of Scientific Research in Multidisciplinary Studies*, Vol. 9, Issue 3, pp. 06–10, March 2023.

[13] Cheney, Ward, and David Kincaid, *Numerical Mathematics and Computing*, International Thomson Publishing, 1998.

[14] Bárczy Barnabás, *Integrálszámítás*, 6. kiadás, Műszaki Könyvkiadó, Budapest, 1992.

[15] Burden, Richard L., and J. Douglas Faires, *Numerical Analysis* (9th edition), Thomson Brooks/Cole, 2010.

[16] Cameron McElfresh, *Monte Carlo Integration*, Medium, https://cameron-mcelfresh.medium.com/monte-carlo-integration-313b37157852.

[17] *Monte Carlo Simulation*, Investopedia, https://www.investopedia.com/terms/m/montecarlosimulation.asp.

[18] *Double Integral via Monte Carlo*, SAS Blogs, https://blogs.sas.com/content/iml/2021/04/07/double-integral-monte-carlo.html.

[19] Ukrajna Oktatási és Tudományos Minisztériuma, *Matematika tantárgyi tanterv az 5–11. évfolyam számára*, https://mon.gov.ua/osvita-2/zagalna-serednya-osvita/osvitni-programi/navchalni-programi-dlya-10-11-klasiv

Ábrák jegyzéke

Táblázatok jegyzéke

Mellékletek

trapez_pelda1.py

A `trapez_pelda1.py` a $\int_0^1(-x^3+x^2+0.5)dx$ határozott integrál értékére ad becslést a trapéz szabály alkalmazásával, emellett ábrákkal szemlélteti a módszer működését.

```
import numpy as np
import matplotlib.pyplot as plt
import sympy as sp

def plot_trapezoidal_subplot(ax, f_lambdified, a, b, n):
    x_vals = np.linspace(a, b, 1000)
    y_vals = f_lambdified(x_vals)
    exact_integral = sp.integrate(f, (x, a, b))

    x_trap = np.linspace(a, b, n + 1)
    y_trap = f_lambdified(x_trap)

    h = (b - a) / n
    approx_integral = (h / 2) * (y_trap[0] + 2 *
    ↪  np.sum(y_trap[1:-1]) + y_trap[-1])

    ax.plot(x_vals, y_vals, label="f(x)", color='blue')

    for i in range(n):
        xs = [x_trap[i], x_trap[i], x_trap[i+1], x_trap[i+1]]
        ys = [0, y_trap[i], y_trap[i+1], 0]
        ax.fill(xs, ys, color='orange', edgecolor='black', alpha=0.5)

    ax.set_title(f"n = {n}\nPontos: {float(exact_integral):.6f},
    ↪  Kzelt: {float(approx_integral):.6f}", fontsize=9)
    ax.set_xlabel("x")
    ax.set_ylabel("f(x)")
    ax.grid(True, linestyle='--', alpha=0.6)

x = sp.Symbol('x')
f = -(x**3) + x**2 + 0.5
f_lambdified = sp.lambdify(x, f, modules=['numpy'])

x_vals = np.linspace(0, 1, 100)

y_vals = f_lambdified(x_vals)
```

```
plt.figure(figsize=(5, 3))
plt.plot(x_vals, y_vals, label=fr"${sp.latex(f)}$", color='b')

plt.axhline(0, color='black', linewidth=1)
plt.axvline(0, color='black', linewidth=1)
plt.grid(True, linestyle='--', alpha=0.6)
plt.xlabel("x")
plt.ylabel("f(x)")
plt.legend()
plt.title("A fggvny brzolsa")
plt.show()

fig, axes = plt.subplots(2, 2, figsize=(10, 8))
fig.suptitle("Trapz mdszer bemutatsa", fontsize=12)
fig.tight_layout(rect=[0, 0, 1, 0.95])
n_values = [1, 2, 10, 20]

axes_flat = axes.flatten()

for ax, n in zip(axes_flat, n_values):
    ax.set_aspect("equal", adjustable='box')
    plot_trapezoidal_subplot(ax, f_lambdified, 0, 1, n)

plt.tight_layout()
plt.show()
```

simpson_pelda1.py

A `simpson_pelda1.py` az $\int_0^1(-x^3+x^2+0.5)dx$ határozott integrál értékére ad becslést a Simpson-módszer alkalmazásával, emellett ábrákkal szemlélteti a módszer működését.

```
import numpy as np
import matplotlib.pyplot as plt
import sympy as sp

def plot_simpson(ax, f, f_lambdified, a, b, n):
    if n % 2 != 0:
        raise ValueError("A Simpson-mdszerhez n-nek prosnak kell
        ↪  lennie.")

    x = sp.Symbol('x')
    exact_integral = sp.integrate(f, (x, a, b))

    x_vals = np.linspace(a, b, 1000)
    y_vals = f_lambdified(x_vals)

    x_simp = np.linspace(a, b, n + 1)
    y_simp = f_lambdified(x_simp)
    h = (b - a) / n
```

```
    approx_integral = (h / 3) * (
        y_simp[0] +
        4 * np.sum(y_simp[1:n:2]) +
        2 * np.sum(y_simp[2:n-1:2]) +
        y_simp[n]
    )

    ax.plot(x_vals, y_vals, label="f(x)", color='blue')

    for i in range(0, n, 2):
        xi = x_simp[i:i+3]
        yi = y_simp[i:i+3]
        coeffs = np.polyfit(xi, yi, 2)
        parabola = np.poly1d(coeffs)
        x_par = np.linspace(xi[0], xi[-1], 100)
        y_par = parabola(x_par)
        ax.fill_between(x_par, y_par, color='green', alpha=0.3)

    ax.set_title(f"n = {n} \n Kzelt rtk: {approx_integral:.6f}   |
    ↪  Pontos rtk: {float(exact_integral):.6f}")
    ax.set_xlabel("x")
    ax.set_ylabel("f(x)")
    ax.grid(True, linestyle='--', alpha=0.6)


x = sp.Symbol('x')
f = -(x**3) + x**2 + 0.5
f_lambdified = sp.lambdify(x, f, modules=['numpy'])

x_vals = np.linspace(0, 1, 100)
y_vals = f_lambdified(x_vals)

plt.figure(figsize=(2, 2))
plt.plot(x_vals, y_vals, label=fr"${sp.latex(f)}$", color='b')

plt.axhline(0, color='black', linewidth=1)
plt.axvline(0, color='black', linewidth=1)
plt.grid(True, linestyle='--', alpha=0.6)
plt.xlabel("x")
plt.ylabel("f(x)")
plt.legend()
plt.title("A fggvny brzolsa")
plt.show()

fig, axes = plt.subplots(2, 2, figsize=(10, 8))
fig.suptitle("Simpson-mdszer bemutatsa", fontsize=12)
fig.tight_layout(rect=[0, 0, 1, 0.95])  # Hagy helyet a cmnek
axes_flat = axes.flatten()

n_values = [2, 4, 8, 10]
for ax, n in zip(axes_flat, n_values):
```

```
    ax.set_aspect("equal", adjustable='box')
    plot_simpson(ax, f, f_lambdified, 0, 1, n)

plt.tight_layout()
plt.show()
```

monte_carlo_pelda1_hit_and_miss.py

A `monte_carlo_pelda1_hit_and_miss.py` az $\int_0^1(-x^3+x^2+0.5)dx$ határozott integrál értékére ad becslést a Monte-Carlo elutasítási módszer alkalmazásával, emellett ábrákkal szemlélteti a módszer működését.

```
import numpy as np
import matplotlib.pyplot as plt
import sympy as sp

def plot_monte_carlo_hit_and_miss(ax, f_lambdified, a, b,
↪  num_points):
    x_vals = np.linspace(a, b, 10000)
    y_vals = f_lambdified(x_vals)
    y_min = 0
    y_max = max(y_vals)

    x_rand = np.random.uniform(a, b, num_points)
    y_rand = np.random.uniform(y_min, y_max, num_points)
    f_rand = f_lambdified(x_rand)

    under_mask = y_rand <= f_rand
    over_mask = ~under_mask
    hits = np.sum(under_mask)

    approx_integral = (b - a) * y_max * hits / num_points
    exact_integral = float(sp.integrate(f, (x, a, b)))

    ax.plot(x_vals, y_vals, color='blue', label="f(x)")
    ax.scatter(x_rand[under_mask], y_rand[under_mask],
    ↪  color='green', s=5, alpha=0.6, label='Alatta')
    ax.scatter(x_rand[over_mask], y_rand[over_mask], color='red',
    ↪  s=5, alpha=0.6, label='Fltte')

    ax.set_title(f"n={num_points}, tallat: {hits} \n Pontos rtk:
    ↪  {float(exact_integral):.6f} Kzelt rtk:
    ↪  {float(approx_integral):.6f}", fontsize=9)
    ax.set_xlabel("x")
    ax.set_ylabel("f(x)")
    ax.grid(True, linestyle='--', alpha=0.6)


x = sp.Symbol('x')
f = -(x**3) + x**2 + 0.5
```

```
f_lambdified = sp.lambdify(x, f, modules=['numpy'])

x_vals = np.linspace(0, 1, 100)
y_vals = f_lambdified(x_vals)

plt.figure(figsize=(2, 2))
plt.plot(x_vals, y_vals, label=fr"${sp.latex(f)}$", color='b')

plt.axhline(0, color='black', linewidth=1)
plt.axvline(0, color='black', linewidth=1)
plt.grid(True, linestyle='--', alpha=0.6)

plt.xlabel("x")
plt.ylabel("f(x)")
plt.legend()
plt.title("A fggvny brzolsa")
plt.show()

fig, axes = plt.subplots(2, 2, figsize=(9, 9))
fig.suptitle("Monte-Carlo elutastsi mdszer alkalmazsa", fontsize=12)
fig.tight_layout(rect=[0, 0, 1, 0.95])
n_values = [100, 500, 1000, 5000]

axes_flat = axes.flatten()

for ax, n in zip(axes_flat, n_values):
    ax.set_aspect("equal", adjustable='box')
    plot_monte_carlo_hit_and_miss(ax, f_lambdified, 0, 1, n)

plt.tight_layout()
plt.show()
```

monte_carlo_pelda1_sampling.py

A `monte_carlo_pelda1_sampling.py` az $\int_0^1(-x^3+x^2+0.5)dx$ határozott integrál értékére ad becslést a Monte-Carlo mintavételezési módszer alkalmazásával, emellett ábrákkal szemlélteti a módszer működését.

```
import numpy as np
import matplotlib.pyplot as plt
import sympy as sp

def plot_monte_carlo_sampling(ax, f_lambdified, a, b, num_points):
    x_vals = np.linspace(a, b, 1000)
    y_vals = f_lambdified(x_vals)
    exact_integral = float(sp.integrate(f, (x, a, b)))

    x_rand = np.random.uniform(a, b, num_points)
    y_rand = f_lambdified(x_rand)
```

```
    avg_value = np.mean(y_rand)
    approx_integral = (b - a) * avg_value

    ax.plot(x_vals, y_vals, label="f(x)", color='blue')
    ax.hlines(avg_value, a, b, color='purple', linestyle='--',
    ↪  label='tlag f(x)')

    ax.scatter(x_rand, y_rand, color='orange', s=10, alpha=0.4,
    ↪  label='mintk')
    ax.fill_between(x_vals, 0, y_vals, alpha=0.2, color='blue')

    ax.set_title(f"n={num_points} \n Pontos rtk:
    ↪  {exact_integral:.6f} Kzelt rtk: {approx_integral:.6f}",
    ↪  fontsize=9)
    ax.set_xlabel("x")
    ax.set_ylabel("f(x)")
    ax.grid(True, linestyle='--', alpha=0.6)

x = sp.Symbol('x')
f = -(x**3) + x**2 + 0.5
f_lambdified = sp.lambdify(x, f, modules=['numpy'])

x_vals = np.linspace(0, 1, 100)
y_vals = f_lambdified(x_vals)

plt.figure(figsize=(2, 2))
plt.plot(x_vals, y_vals, label=fr"${sp.latex(f)}$", color='b')

plt.axhline(0, color='black', linewidth=1)
plt.axvline(0, color='black', linewidth=1)
plt.grid(True, linestyle='--', alpha=0.6)

plt.xlabel("x")
plt.ylabel("f(x)")
plt.legend()
plt.title("A fggvny brzolsa")

plt.show()

fig, axes = plt.subplots(2, 2, figsize=(10, 10))
fig.suptitle("Monte Carlo mdszer bemutatsa (Sampling method)",
↪  fontsize=12)
fig.tight_layout(rect=[0, 0, 1, 0.95])
n_values = [100, 500, 1000, 2000]

axes_flat = axes.flatten()

for ax, n in zip(axes_flat, n_values):
    ax.set_aspect("equal", adjustable='box')
    plot_monte_carlo_sampling(ax, f_lambdified, 0, 1, n)
```

```
plt.tight_layout()
plt.show()
```

monte_carlo_with_halton_pelda1.py

A `monte_carlo_with_halton_pelda1.py` az $\int_0^1(-x^3+x^2+0.5)dx$ határozott integrál értékére ad becslést a Monte-Carlo módszerek Halton-pontokra való alkalmazásával, emellett ábrákkal szemlélteti a módszer működését.

```
import numpy as np
import matplotlib.pyplot as plt
import sympy as sp
from scipy.stats.qmc import Halton

def plot_monte_carlo_hit_and_miss(ax, f_lambdified, a, b,
 ↪  num_points):
    x_vals = np.linspace(a, b, 10000)
    y_vals = f_lambdified(x_vals)
    y_min = 0
    y_max = max(y_vals)

    halton_engine = Halton(d=2, scramble=False)
    halton_points = halton_engine.random(n=num_points)

    x_rand = a + (b - a) * halton_points[:, 0]
    y_rand = y_min + (y_max - y_min) * halton_points[:, 1]
    f_rand = f_lambdified(x_rand)

    under_mask = y_rand <= f_rand
    over_mask = ~under_mask
    hits = np.sum(under_mask)

    approx_integral = (b - a) * y_max * hits / num_points

    exact_integral = float(sp.integrate(f, (x, a, b)))

    ax.plot(x_vals, y_vals, color='blue', label="f(x)")
    ax.scatter(x_rand[under_mask], y_rand[under_mask],
     ↪  color='green', s=5, alpha=0.6, label='Alatta')
    ax.scatter(x_rand[over_mask], y_rand[over_mask], color='red',
     ↪  s=5, alpha=0.6, label='Fltte')

    ax.set_title(
        f"n={num_points}, tallat: {hits} \n Pontos rtk:
         ↪  {float(exact_integral):.6f} Kzelt rtk:
         ↪  {float(approx_integral):.6f}",
        fontsize=9
    )
    ax.set_xlabel("x")
    ax.set_ylabel("f(x)")
```

```
    ax.grid(True, linestyle='--', alpha=0.6)

def plot_monte_carlo_sampling(ax, f_lambdified, a, b, num_points):
    x_vals = np.linspace(a, b, 1000)
    y_vals = f_lambdified(x_vals)
    exact_integral = float(sp.integrate(f, (x, a, b)))

    halton_engine = Halton(d=1, scramble=False)
    halton_points = halton_engine.random(n=num_points)
    x_rand = a + (b - a) * halton_points.flatten()

    y_rand = f_lambdified(x_rand)
    avg_value = np.mean(y_rand)
    approx_integral = (b - a) * avg_value

    ax.plot(x_vals, y_vals, label="f(x)", color='blue')
    ax.hlines(avg_value, a, b, color='purple', linestyle='--',
    ↪  label='tlag f(x)')

    ax.scatter(x_rand, y_rand, color='orange', s=10, alpha=0.4,
    ↪  label='mintk')
    ax.fill_between(x_vals, 0, y_vals, alpha=0.2, color='blue')

    ax.set_title(
        f"n={num_points} \n Pontos rtk: {exact_integral:.6f} Kzelt
        ↪  rtk: {approx_integral:.6f}",
        fontsize=9
    )
    ax.set_xlabel("x")
    ax.set_ylabel("f(x)")
    ax.grid(True, linestyle='--', alpha=0.6)


x = sp.Symbol('x')
f = -(x**3) + x**2 + 0.5

f_lambdified = sp.lambdify(x, f, modules=['numpy'])

x_vals = np.linspace(0, 1, 100)
y_vals = f_lambdified(x_vals)

plt.figure(figsize=(2, 2))
plt.plot(x_vals, y_vals, label=fr"${sp.latex(f)}$", color='b')

plt.axhline(0, color='black', linewidth=1)
plt.axvline(0, color='black', linewidth=1)
plt.grid(True, linestyle='--', alpha=0.6)

plt.xlabel("x")
plt.ylabel("f(x)")
plt.legend()
```

```
plt.title("A fggvny brzolsa")

plt.show()

fig1, axes1 = plt.subplots(2, 2, figsize=(10, 10))
fig1.suptitle("Monte-Carlo mdszer - Hit and Miss -
↪  Halton-pontokkal", fontsize=12)
fig1.tight_layout(rect=[0, 0, 1, 0.95])
n_values = [100, 500, 1000, 5000]

axes_flat = axes1.flatten()

for ax, n in zip(axes_flat, n_values):
    ax.set_aspect("equal", adjustable='box')
    plot_monte_carlo_hit_and_miss(ax, f_lambdified, 0, 1, n)

plt.tight_layout()
plt.show()

fig2, axes2 = plt.subplots(2, 2, figsize=(10, 10))
fig2.suptitle("Monte-Carlo mdszer - Sampling method - Halton
↪  pontokkal", fontsize=12)
fig2.tight_layout(rect=[0, 0, 1, 0.95])
n_values = [100, 500, 1000, 5000]

axes_flat = axes2.flatten()

for ax, n in zip(axes_flat, n_values):
    ax.set_aspect("equal", adjustable='box')
    plot_monte_carlo_sampling(ax, f_lambdified, 0, 1, n)

plt.tight_layout()
plt.show()
```

osszehasonlitas_egyvaltozos_integralok1.py

A `osszehasonlitas_egyvaltozos_integralok1.py` a $\int_0^1 e^{-x^2} dx$ határozott integrál értékére ad becslést a numerikus és Monte-Carlo módszerek alkalmazásával, emellett ábrákkal szemlélteti a módszerek működését.

```
import numpy as np
import matplotlib.pyplot as plt
import sympy as sp
from scipy.stats.qmc import Halton
from numpy.random import default_rng
rng = default_rng()

def approx_trapezodal(f_lambdified, a, b, n):
    x_trap = np.linspace(a, b, n + 1)
    y_trap = f_lambdified(x_trap)
```

```
    h = (b - a) / n
    approx_integral = (h / 2) * (y_trap[0] + 2 *
    ↪  np.sum(y_trap[1:-1]) + y_trap[-1])
    print(f"\tKzelt rtk: {approx_integral:.6f}, n = {n}")

def approx_simpson(f_lambdified, a, b, n):
    if n % 2 != 0:
        raise ValueError("A Simpson-mdszerhez n-nek prosnak kell
        ↪  lennie.")

    x_simp = np.linspace(a, b, n + 1)
    y_simp = f_lambdified(x_simp)
    h = (b - a) / n
    approx_integral = (h / 3) * (
        y_simp[0] +
        4 * np.sum(y_simp[1:n:2]) +
        2 * np.sum(y_simp[2:n-1:2]) +
        y_simp[n]
    )
    print(f"\tKzelt rtk: {approx_integral:.6f}, n = {n}")

def monte_carlo_hit_and_miss(f_lambdified, a, b, num_points):
    y_min = 0
    y_max = 1

    x_rand = rng.uniform(a, b, num_points)
    y_rand = rng.uniform(y_min, y_max, num_points)

    f_rand = f_lambdified(x_rand)

    under_mask = y_rand <= f_rand
    hits = np.sum(under_mask)
    approx_integral = (b - a) * y_max * hits / num_points
    print(f"\tKzelt rtk: {approx_integral:.6f}, n = {num_points}, nt
    ↪  = {hits}")


def monte_carlo_hit_and_miss_halton(f_lambdified, a,  b, num_points):
    y_min = 0
    y_max = 1

    halton_engine = Halton(d=2, scramble=False)
    halton_points = halton_engine.random(n=num_points)

    x_rand = a + (b - a) * halton_points[:, 0]
    y_rand = y_min + (y_max - y_min) * halton_points[:, 1]

    f_rand = f_lambdified(x_rand)
    under_mask = y_rand <= f_rand
    hits = np.sum(under_mask)
```

```
    approx_integral = (b - a) * y_max * hits / num_points
    print(f"\tKzelt rtk: {approx_integral:.6f}, n = {num_points}, nt
    ↪  = {hits}")


def monte_carlo_sampling(f_lambdified, a, b, num_points):
    x_rand = np.random.uniform(a, b, num_points)
    y_rand = f_lambdified(x_rand)
    avg_value = np.mean(y_rand)
    approx_integral = (b - a) * avg_value
    print(f"\tKzelt rtk: {approx_integral:.6f}, n = {num_points}")


def monte_carlo_sampling_halton(f_lambdified, a, b, num_points):
    halton_engine = Halton(d=1, scramble=False)
    halton_points = halton_engine.random(n=num_points)
    x_rand = a + (b - a) * halton_points.flatten()
    y_rand = f_lambdified(x_rand)
    avg_value = np.mean(y_rand)
    approx_integral = (b - a) * avg_value
    print(f"\tKzelt rtk: {approx_integral:.6f}, n = {num_points}")

def plot_trapezoidal_subplot(ax, f_lambdified, a, b, n):
    x_vals = np.linspace(a, b, 1000)
    y_vals = f_lambdified(x_vals)

    x_trap = np.linspace(a, b, n + 1)
    y_trap = f_lambdified(x_trap)

    h = (b - a) / n
    approx_integral = (h / 2) * (y_trap[0] + 2 *
    ↪  np.sum(y_trap[1:-1]) + y_trap[-1])

    ax.plot(x_vals, y_vals, label="f(x)", color='blue')

    for i in range(n):
        xs = [x_trap[i], x_trap[i], x_trap[i+1], x_trap[i+1]]
        ys = [0, y_trap[i], y_trap[i+1], 0]
        ax.fill(xs, ys, color='orange', edgecolor='black', alpha=0.5)

    ax.set_title(f"Kzelt: {float(approx_integral):.6f}, n = {n}",
    ↪  fontsize=9)
    ax.set_xlabel("x")
    ax.set_ylabel("f(x)")
    ax.grid(True, linestyle='--', alpha=0.6)

def plot_simpson(ax, f_lambdified, a, b, n):
    if n % 2 != 0:
        raise ValueError("A Simpson-mdszerhez n-nek prosnak kell
        ↪  lennie.")
```

```
    x_vals = np.linspace(a, b, 1000)
    y_vals = f_lambdified(x_vals)

    x_simp = np.linspace(a, b, n + 1)
    y_simp = f_lambdified(x_simp)
    h = (b - a) / n
    approx_integral = (h / 3) * (
        y_simp[0] +
        4 * np.sum(y_simp[1:n:2]) +
        2 * np.sum(y_simp[2:n-1:2]) +
        y_simp[n]
    )

    ax.plot(x_vals, y_vals, label="f(x)", color='blue')

    for i in range(0, n, 2):
        xi = x_simp[i:i+3]
        yi = y_simp[i:i+3]
        coeffs = np.polyfit(xi, yi, 2)
        parabola = np.poly1d(coeffs)
        x_par = np.linspace(xi[0], xi[-1], 100)
        y_par = parabola(x_par)
        ax.fill_between(x_par, y_par, color='green', alpha=0.3)

    ax.set_title(f"Kzelt rtk: {approx_integral:.6f}, n = {n}",
    ↪  fontsize = 9)
    ax.set_xlabel("x")
    ax.set_ylabel("f(x)")
    ax.grid(True, linestyle='--', alpha=0.6)

def plot_monte_carlo_hit_and_miss(ax, f_lambdified, a, b,
↪  num_points):
    x_vals = np.linspace(a, b, 1000)
    y_vals = f_lambdified(x_vals)
    y_min = 0
    y_max = 1

    x_rand = np.random.uniform(a, b, num_points)
    y_rand = np.random.uniform(y_min, y_max, num_points)

    f_rand = f_lambdified(x_rand)

    under_mask = y_rand < f_rand
    over_mask = ~under_mask
    hits = np.sum(under_mask)
    approx_integral = (b - a) * y_max * hits / num_points

    ax.plot(x_vals, y_vals, color='blue', label="f(x)")
    ax.scatter(x_rand[under_mask], y_rand[under_mask],
    ↪  color='green', s=5, alpha=0.6, label='Alatta')
```

```
    ax.scatter(x_rand[over_mask], y_rand[over_mask], color='red',
    ↪  s=5, alpha=0.6, label='Fltte')

    ax.set_title(f"Kzelt rtk: {float(approx_integral):.6f}, \n
    ↪  n={num_points}, tallat: {hits}", fontsize=9)
    ax.set_xlabel("x")
    ax.set_ylabel("f(x)")
    ax.grid(True, linestyle='--', alpha=0.6)

def plot_monte_carlo_hit_and_miss_halton(ax, f_lambdified, a, b,
↪  num_points):
    x_vals = np.linspace(a, b, 1000)
    y_vals = f_lambdified(x_vals)
    y_min = 0
    y_max = 1

    halton_engine = Halton(d=2, scramble=False)
    halton_points = halton_engine.random(n=num_points)

    x_rand = a + (b - a) * halton_points[:, 0]
    y_rand = y_min + (y_max - y_min) * halton_points[:, 1]

    f_rand = f_lambdified(x_rand)
    under_mask = y_rand < f_rand
    over_mask = ~under_mask
    hits = np.sum(under_mask)

    approx_integral = (b - a) * y_max * hits / num_points

    ax.plot(x_vals, y_vals, color='blue', label="f(x)")
    ax.scatter(x_rand[under_mask], y_rand[under_mask],
    ↪  color='green', s=5, alpha=0.6, label='Alatta')
    ax.scatter(x_rand[over_mask], y_rand[over_mask], color='red',
    ↪  s=5, alpha=0.6, label='Fltte')

    ax.set_title(
        f"Kzelt rtk Halton pontokkal: {float(approx_integral):.6f},
        ↪  \n n={num_points}, tallat: {hits}",
        fontsize=9
    )
    ax.set_xlabel("x")
    ax.set_ylabel("f(x)")
    ax.grid(True, linestyle='--', alpha=0.6)

def plot_monte_carlo_sampling(ax, f_lambdified, a, b, num_points):
    x_vals = np.linspace(a, b, 1000)
    y_vals = f_lambdified(x_vals)

    x_rand = np.random.uniform(a, b, num_points)
    y_rand = f_lambdified(x_rand)
    avg_value = np.mean(y_rand)
```

```
    approx_integral = (b - a) * avg_value

    ax.plot(x_vals, y_vals, label="f(x)", color='blue')
    ax.hlines(avg_value, a, b, color='purple', linestyle='--',
    ↪  label='tlag f(x)')

    ax.scatter(x_rand, y_rand, color='orange', s=10, alpha=0.4,
    ↪  label='mintk')
    ax.fill_between(x_vals, 0, y_vals, alpha=0.2, color='blue')

    ax.set_title(f"Kzelt rtk: {approx_integral:.6f},
    ↪  n={num_points}", fontsize=9)
    ax.set_xlabel("x")
    ax.set_ylabel("f(x)")
    ax.grid(True, linestyle='--', alpha=0.6)

def plot_monte_carlo_sampling_halton(ax, f_lambdified, a, b,
↪  num_points):
    x_vals = np.linspace(a, b, 1000)
    y_vals = f_lambdified(x_vals)

    halton_engine = Halton(d=1, scramble=False)
    halton_points = halton_engine.random(n=num_points)
    x_rand = a + (b - a) * halton_points.flatten()
    y_rand = f_lambdified(x_rand)
    avg_value = np.mean(y_rand)
    approx_integral = (b - a) * avg_value

    ax.plot(x_vals, y_vals, label="f(x)", color='blue')
    ax.hlines(avg_value, a, b, color='purple', linestyle='--',
    ↪  label='tlag f(x)')

    ax.scatter(x_rand, y_rand, color='orange', s=10, alpha=0.4,
    ↪  label='mintk')
    ax.fill_between(x_vals, 0, y_vals, alpha=0.2, color='blue')

    ax.set_title(
        f"Kzelt rtk Halton pontokkal: {approx_integral:.6f},
        ↪  n={num_points}",
        fontsize=9
    )
    ax.set_xlabel("x")
    ax.set_ylabel("f(x)")
    ax.grid(True, linestyle='--', alpha=0.6)

x = sp.Symbol('x')
f = sp.exp(-x**2)
f_lambdified = sp.lambdify(x, f, modules=['numpy'])
exact_integral = float(sp.integrate(f, (x, 0, 1)))
x_vals = np.linspace(0, 1, 100)
y_vals = f_lambdified(x_vals)
```

```
plt.figure(figsize=(2, 2))
plt.plot(x_vals, y_vals, label=fr"${sp.latex(f)}$", color='b')

plt.axhline(0, color='black', linewidth=1)
plt.axvline(0, color='black', linewidth=1)
plt.grid(True, linestyle='--', alpha=0.6)

plt.xlabel("x")
plt.ylabel("f(x)")
plt.legend()
plt.title("A fggvny brzolsa")
plt.show()

n_values = [2, 10, 20, 50, 100]
print("Trapz mdszer:")
for n in n_values:
    approx_trapezodal(f_lambdified, 0, 1, n)
n_values = [2, 10, 20]
print("Simpson mdszer:")
for n in n_values:
    approx_simpson(f_lambdified, 0, 1, n)

n_mc = [1000, 5000, 10000, 20000, 50000, 100000, 200000, 300000,
↪  500000,]
print("Monte Carlo - Hit and miss:")
for n in n_mc:
    monte_carlo_hit_and_miss(f_lambdified, 0, 1, n)

print("Monte Carlo - Hit and miss - Halton:")
for n in n_mc:
    monte_carlo_hit_and_miss_halton(f_lambdified, 0, 1, n)

print("Monte Carlo - Sampling:")
for n in n_mc:
    monte_carlo_sampling(f_lambdified, 0, 1, n)

print("Monte Carlo - Sampling method - Halton:")
for n in n_mc:
    monte_carlo_sampling_halton(f_lambdified, 0, 1, n)

fig, axes = plt.subplots(2, 3, figsize=(7, 7))
fig.suptitle(f"Numerikus s Monte-Carlo mdszerek sszehasonltsa.
↪  Pontos rtk: {float(exact_integral):.6f}", fontsize=12)
fig.tight_layout(rect=[0, 0, 1, 0.95])

axes_flat = axes.flatten()
plot_trapezoidal_subplot(axes_flat[0], f_lambdified, 0, 1, 10)
plot_simpson(axes_flat[3], f_lambdified, 0, 1, 10)
plot_monte_carlo_hit_and_miss(axes_flat[1], f_lambdified, 0, 1,
↪  10000)
```

```
plot_monte_carlo_sampling(axes_flat[4], f_lambdified, 0, 1, 1000)
plot_monte_carlo_hit_and_miss_halton(axes_flat[2], f_lambdified, 0,
↪  1, 10000)
plot_monte_carlo_sampling_halton(axes_flat[5], f_lambdified, 0, 1,
↪  1000)
plt.tight_layout()
plt.show()
```

osszehasonlitas_egyvaltozos_integralok2.py

A `osszehasonlitas_egyvaltozos_integralok2.py` a $\int_{-5}^{5} \frac{1}{1+x^2} dx$ határozott integrál értékére ad becslést a numerikus és Monte-Carlo módszerek alkalmazásával, emellett ábrákkal szemlélteti a módszerek működését.

```
import numpy as np
import matplotlib.pyplot as plt
import sympy as sp
from scipy.stats.qmc import Halton
from numpy.random import default_rng
rng = default_rng()

def approx_trapezodal(f_lambdified, a, b, n):
    x_trap = np.linspace(a, b, n + 1)
    y_trap = f_lambdified(x_trap)

    h = (b - a) / n
    approx_integral = (h / 2) * (y_trap[0] + 2 *
    ↪  np.sum(y_trap[1:-1]) + y_trap[-1])
    print(f"\tKzelt rtk: {approx_integral:.6f}, n = {n}")

def approx_simpson(f_lambdified, a, b, n):
    if n % 2 != 0:
        raise ValueError("A Simpson-mdszerhez n-nek prosnak kell
        ↪  lennie.")

    x_simp = np.linspace(a, b, n + 1)
    y_simp = f_lambdified(x_simp)
    h = (b - a) / n
    approx_integral = (h / 3) * (
        y_simp[0] +
        4 * np.sum(y_simp[1:n:2]) +
        2 * np.sum(y_simp[2:n-1:2]) +
        y_simp[n]
    )
    print(f"\tKzelt rtk: {approx_integral:.6f}, n = {n}")

def monte_carlo_hit_and_miss(f_lambdified, a, b, num_points):
    y_min = 0
    y_max = 1
```

```
    x_rand = rng.uniform(a, b, num_points)
    y_rand = rng.uniform(y_min, y_max, num_points)
    f_rand = f_lambdified(x_rand)
    under_mask = y_rand <= f_rand
    hits = np.sum(under_mask)
    approx_integral = (b - a) * y_max * hits / num_points
    print(f"\tKzelt rtk: {approx_integral:.6f}, n = {num_points}, nt
    ↪  = {hits}")


def monte_carlo_hit_and_miss_halton(f_lambdified, a,  b, num_points):
    y_min = 0
    y_max = 1

    halton_engine = Halton(d=2, scramble=False)
    halton_points = halton_engine.random(n=num_points)

    x_rand = a + (b - a) * halton_points[:, 0]
    y_rand = y_min + (y_max - y_min) * halton_points[:, 1]
    f_rand = f_lambdified(x_rand)
    under_mask = y_rand <= f_rand
    hits = np.sum(under_mask)
    approx_integral = (b - a) * y_max * hits / num_points
    print(f"\tKzelt rtk: {approx_integral:.6f}, n = {num_points}, nt
    ↪  = {hits}")


def monte_carlo_sampling(f_lambdified, a, b, num_points):
    x_rand = np.random.uniform(a, b, num_points)
    y_rand = f_lambdified(x_rand)
    avg_value = np.mean(y_rand)
    approx_integral = (b - a) * avg_value
    print(f"\tKzelt rtk: {approx_integral:.6f}, n = {num_points}")


def monte_carlo_sampling_halton(f_lambdified, a, b, num_points):
    halton_engine = Halton(d=1, scramble=False)
    halton_points = halton_engine.random(n=num_points)
    x_rand = a + (b - a) * halton_points.flatten()

    # f(x_rand) rtkek s tlaguk
    y_rand = f_lambdified(x_rand)
    avg_value = np.mean(y_rand)
    approx_integral = (b - a) * avg_value
    print(f"\tKzelt rtk: {approx_integral:.6f}, n = {num_points}")

def plot_trapezoidal_subplot(ax, f_lambdified, a, b, n):
    x_vals = np.linspace(a, b, 1000)
    y_vals = f_lambdified(x_vals)

    x_trap = np.linspace(a, b, n + 1)
```

```
    y_trap = f_lambdified(x_trap)

    h = (b - a) / n
    approx_integral = (h / 2) * (y_trap[0] + 2 *
    ↪  np.sum(y_trap[1:-1]) + y_trap[-1])

    ax.plot(x_vals, y_vals, label="f(x)", color='blue')

    for i in range(n):
        xs = [x_trap[i], x_trap[i], x_trap[i+1], x_trap[i+1]]
        ys = [0, y_trap[i], y_trap[i+1], 0]
        ax.fill(xs, ys, color='orange', edgecolor='black', alpha=0.5)

    ax.set_title(f"Kzelt: {float(approx_integral):.6f}, n = {n}",
    ↪  fontsize=9)
    ax.set_xlabel("x")
    ax.set_ylabel("f(x)")
    ax.grid(True, linestyle='--', alpha=0.6)

def plot_simpson(ax, f_lambdified, a, b, n):
    if n % 2 != 0:
        raise ValueError("A Simpson-mdszerhez n-nek prosnak kell
        ↪  lennie.")

    x_vals = np.linspace(a, b, 1000)
    y_vals = f_lambdified(x_vals)

    x_simp = np.linspace(a, b, n + 1)
    y_simp = f_lambdified(x_simp)
    h = (b - a) / n
    approx_integral = (h / 3) * (
        y_simp[0] +
        4 * np.sum(y_simp[1:n:2]) +
        2 * np.sum(y_simp[2:n-1:2]) +
        y_simp[n]
    )

    ax.plot(x_vals, y_vals, label="f(x)", color='blue')

    for i in range(0, n, 2):
        xi = x_simp[i:i+3]
        yi = y_simp[i:i+3]
        coeffs = np.polyfit(xi, yi, 2)
        parabola = np.poly1d(coeffs)
        x_par = np.linspace(xi[0], xi[-1], 100)
        y_par = parabola(x_par)
        ax.fill_between(x_par, y_par, color='green', alpha=0.3)

    ax.set_title(f"Kzelt rtk: {approx_integral:.6f}, n = {n}",
    ↪  fontsize = 9)
    ax.set_xlabel("x")
```

```
    ax.set_ylabel("f(x)")
    ax.grid(True, linestyle='--', alpha=0.6)

def plot_monte_carlo_hit_and_miss(ax, f_lambdified, a, b,
↪  num_points):
    x_vals = np.linspace(a, b, 10000)
    y_vals = f_lambdified(x_vals)
    y_min = 0
    y_max = 1

    x_rand = np.random.uniform(a, b, num_points)
    y_rand = np.random.uniform(y_min, y_max, num_points)
    f_rand = f_lambdified(x_rand)
    under_mask = y_rand <= f_rand
    over_mask = ~under_mask
    hits = np.sum(under_mask)
    approx_integral = (b - a) * y_max * hits / num_points

    ax.plot(x_vals, y_vals, color='blue', label="f(x)")
    ax.scatter(x_rand[under_mask], y_rand[under_mask],
    ↪  color='green', s=5, alpha=0.6, label='Alatta')
    ax.scatter(x_rand[over_mask], y_rand[over_mask], color='red',
    ↪  s=5, alpha=0.6, label='Fltte')

    ax.set_title(f"Kzelt rtk: {float(approx_integral):.6f}, \n
    ↪  n={num_points}, tallat: {hits}", fontsize=9)
    ax.set_xlabel("x")
    ax.set_ylabel("f(x)")
    ax.grid(True, linestyle='--', alpha=0.6)

def plot_monte_carlo_hit_and_miss_halton(ax, f_lambdified, a, b,
↪  num_points):
    x_vals = np.linspace(a, b, 10000)
    y_vals = f_lambdified(x_vals)
    y_min = 0
    y_max = 1

    halton_engine = Halton(d=2, scramble=False)
    halton_points = halton_engine.random(n=num_points)

    x_rand = a + (b - a) * halton_points[:, 0]
    y_rand = y_min + (y_max - y_min) * halton_points[:, 1]

    f_rand = f_lambdified(x_rand)

    under_mask = y_rand <= f_rand
    over_mask = ~under_mask
    hits = np.sum(under_mask)

    approx_integral = (b - a) * y_max * hits / num_points
```

```
    ax.plot(x_vals, y_vals, color='blue', label="f(x)")
    ax.scatter(x_rand[under_mask], y_rand[under_mask],
    ↪  color='green', s=5, alpha=0.6, label='Alatta')
    ax.scatter(x_rand[over_mask], y_rand[over_mask], color='red',
    ↪  s=5, alpha=0.6, label='Fltte')

    ax.set_title(
        f"Kzelt rtk Halton pontokkal: {float(approx_integral):.6f},
        ↪  \n n={num_points}, tallat: {hits}",
        fontsize=9
    )
    ax.set_xlabel("x")
    ax.set_ylabel("f(x)")
    ax.grid(True, linestyle='--', alpha=0.6)

def plot_monte_carlo_sampling(ax, f_lambdified, a, b, num_points):
    x_vals = np.linspace(a, b, 1000)
    y_vals = f_lambdified(x_vals)

    x_rand = np.random.uniform(a, b, num_points)
    y_rand = f_lambdified(x_rand)
    avg_value = np.mean(y_rand)
    approx_integral = (b - a) * avg_value

    ax.plot(x_vals, y_vals, label="f(x)", color='blue')
    ax.hlines(avg_value, a, b, color='purple', linestyle='--',
    ↪  label='tlag f(x)')

    ax.scatter(x_rand, y_rand, color='orange', s=10, alpha=0.4,
    ↪  label='mintk')
    ax.fill_between(x_vals, 0, y_vals, alpha=0.2, color='blue')

    ax.set_title(f"Kzelt rtk: {approx_integral:.6f},
    ↪  n={num_points}", fontsize=9)
    ax.set_xlabel("x")
    ax.set_ylabel("f(x)")
    ax.grid(True, linestyle='--', alpha=0.6)

def plot_monte_carlo_sampling_halton(ax, f_lambdified, a, b,
↪  num_points):
    x_vals = np.linspace(a, b, 1000)
    y_vals = f_lambdified(x_vals)

    halton_engine = Halton(d=1, scramble=False)
    halton_points = halton_engine.random(n=num_points)
    x_rand = a + (b - a) * halton_points.flatten()

    y_rand = f_lambdified(x_rand)
    avg_value = np.mean(y_rand)
    approx_integral = (b - a) * avg_value
```

```
    ax.plot(x_vals, y_vals, label="f(x)", color='blue')
    ax.hlines(avg_value, a, b, color='purple', linestyle='--',
    ↪  label='tlag f(x)')

    ax.scatter(x_rand, y_rand, color='orange', s=10, alpha=0.4,
    ↪  label='mintk')
    ax.fill_between(x_vals, 0, y_vals, alpha=0.2, color='blue')

    ax.set_title(
        f"Kzelt rtk Halton pontokkal: {approx_integral:.6f},
        ↪  n={num_points}",
        fontsize=9
    )
    ax.set_xlabel("x")
    ax.set_ylabel("f(x)")
    ax.grid(True, linestyle='--', alpha=0.6)

x = sp.Symbol('x')
f = 1 / (1 + x**2)
a = -5
b = 5
f_lambdified = sp.lambdify(x, f, modules=['numpy'])
exact_integral = float(sp.integrate(f, (x, a, b)))

x_vals = np.linspace(a, b, 10000)
y_vals = f_lambdified(x_vals)

plt.figure(figsize=(2, 2))
plt.plot(x_vals, y_vals, label=fr"${sp.latex(f)}$", color='b')
plt.axhline(0, color='black', linewidth=1)
plt.axvline(0, color='black', linewidth=1)
plt.grid(True, linestyle='--', alpha=0.6)
plt.xlabel("x")
plt.ylabel("f(x)")
plt.legend()
plt.title("A fggvny brzolsa")

plt.show()

n_values = [2, 10, 20, 30, 40, 50, 100]
print("Trapz mdszer:")
for n in n_values:
    approx_trapezodal(f_lambdified, a, b, n)

print("Simpson mdszer:")
for n in n_values:
    approx_simpson(f_lambdified, a, b, n)

n_mc = [500, 1000, 5000, 10000, 20000, 50000, 100000, 200000,
↪  300000, 500000, 1000000, 2000000, 3000000]
print("Monte Carlo - Hit and miss:")
```

```
for n in n_mc:
    monte_carlo_hit_and_miss(f_lambdified, a, b, n)

print("Monte Carlo - Hit and miss - Halton:")
for n in n_mc:
    monte_carlo_hit_and_miss_halton(f_lambdified, a, b, n)

print("Monte Carlo - Sampling:")
for n in n_mc:
    monte_carlo_sampling(f_lambdified, a, b, n)

print("Monte Carlo - Sampling method - Halton:")
for n in n_mc:
    monte_carlo_sampling_halton(f_lambdified, a, b, n)

fig, axes = plt.subplots(2, 3, figsize=(7, 7))
fig.suptitle(f"Numerikus s Monte-Carlo mdszerek sszehasonltsa.
↪  Pontos rtk: {float(exact_integral):.6f}", fontsize=12)
fig.tight_layout(rect=[0, 0, 1, 0.95])

axes_flat = axes.flatten()
plot_trapezoidal_subplot(axes_flat[0], f_lambdified, a, b, 10)
plot_simpson(axes_flat[3], f_lambdified, a, b, 10)
plot_monte_carlo_hit_and_miss(axes_flat[1], f_lambdified, a, b, 1000)
plot_monte_carlo_sampling(axes_flat[4], f_lambdified, a, b, 1000)
plot_monte_carlo_hit_and_miss_halton(axes_flat[2], f_lambdified, a,
↪  b, 1000)
plot_monte_carlo_sampling_halton(axes_flat[5], f_lambdified, a, b,
↪  1000)
plt.tight_layout()
plt.show()
```

monte_carlo_animated.py

A `monte_carlo_animated.py` animáció segítségével mutatja be a Monte-Carlo elutasítási módszer működését. Az animációt meg lehet állítani, és folytatni. A közelítendő határozott integrál: $\int_0^2 (x^2 + 1) dx$.

```
import numpy as np
import matplotlib.pyplot as plt
import matplotlib.animation as animation
from matplotlib.widgets import Button

def f(x):
    return x**2 + 1

a, b = 0, 2
max_y = f(b)
n_points = 0
```

```
x_inside, y_inside = [], []
x_outside, y_outside = [], []

fig, ax = plt.subplots()
plt.subplots_adjust(bottom=0.15)
ax.set_xlim(a-0.1, b+0.1)
ax.set_ylim(0-0.1, max_y + 0.1)
ax.set_aspect('equal')
ax.set_xlabel("x")
ax.set_ylabel("y")

x_vals = np.linspace(a, b, 500)
y_vals = f(x_vals)
ax.plot(x_vals, y_vals, color='blue', label='f(x) = x')

scat_inside = ax.scatter([], [], color='green', s=10, label='Fggvny
↪  alatt')
scat_outside = ax.scatter([], [], color='red', s=10, label='Fggvny
↪  felett')

text_estimate = fig.text(0.5, 0.96, '', ha='center', fontsize=10)
text_hits = fig.text(0.5, 0.92, '', ha='center', fontsize=10)

is_running = [False]

def init():
    scat_inside.set_offsets(np.empty((0, 2)))
    scat_outside.set_offsets(np.empty((0, 2)))
    text_estimate.set_text('')
    text_hits.set_text('')
    return scat_inside, scat_outside, text_estimate, text_hits

def update(frame):
    if not is_running[0]:
        return scat_inside, scat_outside, text_estimate, text_hits

    global n_points, x_inside, y_inside, x_outside, y_outside

    x_rand = np.random.uniform(a, b)
    y_rand = np.random.uniform(0, max_y)
    n_points += 1

    if y_rand <= f(x_rand):
        x_inside.append(x_rand)
        y_inside.append(y_rand)
    else:
        x_outside.append(x_rand)
        y_outside.append(y_rand)

    scat_inside.set_offsets(np.column_stack((x_inside, y_inside)))
    scat_outside.set_offsets(np.column_stack((x_outside, y_outside)))
```

```
    ratio = len(x_inside) / n_points
    area = (b - a) * max_y
    estimate = ratio * area

    text_estimate.set_text(f"Kzeltett integrl: {estimate:.4f}")
    text_hits.set_text(f"Tallatok szma: {len(x_inside)}; Pontok
    ↪  szma: {n_points}")
    return scat_inside, scat_outside, text_estimate, text_hits

ax_pause = plt.axes([0.7, 0.05, 0.1, 0.075])
btn_pause = Button(ax_pause, 'Pause')

ax_continue = plt.axes([0.81, 0.05, 0.1, 0.075])
btn_continue = Button(ax_continue, 'Continue')

def pause(event):
    is_running[0] = False

def cont(event):
    is_running[0] = True

btn_pause.on_clicked(pause)
btn_continue.on_clicked(cont)

ani = animation.FuncAnimation(
    fig, update, init_func=init, interval=500, blit=False,
    frames=None, save_count=500, cache_frame_data=False
)

plt.tight_layout
plt.show()
```

kettos_trapez_sulyok.py

A `kettos_trapez_sulyok.py` a pontok súlyait mutatja be a trapéz módszer alkalmazása esetén ábrákkal szemléltetve.

```
import numpy as np
import matplotlib.pyplot as plt

def trapez_sulyok(n, m):
    weights = np.ones((n + 1, m + 1))
    for i in range(n+1):
        for j in range(m+1):
            if (i == 0 or i == n) and (j == 0 or j == m):
                weights[i, j] = 1 # sarok
            elif (i == 0 or i == n) or (j == 0 or j == m):
                weights[i, j] = 2
            else:
```

```
                weights[i, j] = 4
    return weights

def plot_kettos_trapez_sulyok(ax, x_intervals, y_intervals, n, m):
    xa, xb = x_intervals
    ya, yb = y_intervals
    x_vals = np.linspace(xa, xb, n + 1)
    y_vals = np.linspace(ya, yb, m + 1)
    X, Y = np.meshgrid(x_vals, y_vals, indexing='ij')
    W = trapez_sulyok(n, m)

    colors = {1: 'black', 2: 'purple', 4: 'blue'}
    labels = {1: 'Sarok (1)', 2: 'Szln (2)', 4: 'Bels (4)' }
    plotted_labels = set()

    for i in range(n + 1):
        for j in range(m + 1):
            weight = W[i, j]
            color = colors[weight]
            label = labels[weight] if weight not in plotted_labels
             ↪  else None
            ax.scatter(X[i, j], Y[i, j], color=color, s=50,
             ↪  label=label)
            ax.text(X[i, j]+0.025, Y[i, j], str(int(weight)),
             ↪  color='black', fontsize=10)
            plotted_labels.add(weight)

    ax.set_title(f"n = {n}, m = {m}")
    ax.set_xlabel("x")
    ax.set_ylabel("y")

a, b = 0, 1
c, d = 0, 1
n, m = 2, 2

fig = plt.figure(figsize=(10, 5))
ax1 = fig.add_subplot(1, 2, 1)
ax2 = fig.add_subplot(1, 2, 2)

plot_kettos_trapez_sulyok(ax1, (a, b), (c, d), n, m)
n, m = 3, 2
plot_kettos_trapez_sulyok(ax2, (a, b), (c, d), n, m)

plt.tight_layout()
plt.show()
```

kettos_simpson_bemutatasa2.py

A `kettos_simpson_bemutatasa2.py` a pontok súlyait mutatja be a Simpson-módszer alkalmazása esetén ábrákkal szemléltetve.

```python
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

def simpson_weights(nx, ny):
    weights = np.ones((nx + 1, ny + 1))
    for i in range(nx + 1):
        for j in range(ny + 1):
            if (i == 0 or i == nx) and (j == 0 or j == ny):
                weights[i, j] = 1  # sarok
            elif (i == 0 or i == nx) or (j == 0 or j == ny):
                if i % 2 ==0 and j % 2 == 0:
                    weights[i, j] = 2  # szln (de nem sarok) - egy
                    ↪ illesztes sarka
                else:
                    weights[i, j] = 4  # szln (de nem sarok) - egy
                    ↪ illeszts kt sarka kzt
            elif i % 2 == 1 and j % 2 == 1:
                weights[i, j] = 16  # bels, mindkett pratlan
            elif i % 2 == 0 and j % 2 == 0:
                weights[i, j] = 4   # bels, mindkett pros
            else:
                weights[i, j] = 8   # vegyes (egyik pros, msik
                ↪ pratlan)
    return weights

def plot_kettos_simpson_sulyok(ax, x_interval, y_interval, n, m):
    a, b = x_interval
    c, d = y_interval
    x_vals = np.linspace(a, b, n + 1)
    y_vals = np.linspace(c, d, m + 1)
    X, Y = np.meshgrid(x_vals, y_vals, indexing='ij')
    W = simpson_weights(n, m)
    colors = {1: 'black', 2: 'purple', 4: 'blue', 8: 'orange', 16:
    ↪ 'red'}
    labels = {1: 'Sarok (1)', 2: 'Szln (2)', 4: 'Szln (4)', 8:
    ↪ 'Vegyes (8)', 16: 'Bels (16)'}
    plotted_labels = set()

    for i in range(n + 1):
        for j in range(m + 1):
            weight = W[i, j]
            color = colors[weight]
            label = labels[weight] if weight not in plotted_labels
            ↪ else None
            ax.scatter(X[i, j], Y[i, j], color=color, s=50,
            ↪ label=label)
            ax.text(X[i, j]+0.025, Y[i, j], str(int(weight)),
            ↪ color='black', fontsize=10)
            plotted_labels.add(weight)
    ax.set_title(f"n = {n}, m = {m}")
```

```
    ax.set_xlabel("x")
    ax.set_ylabel("y")

a, b = 0, 1
c, d = 0, 1
n, m = 4, 2

fig = plt.figure(figsize=(10, 5))
ax1 = fig.add_subplot(1, 2, 1)
ax2 = fig.add_subplot(1, 2, 2)
plot_kettos_simpson_sulyok(ax1, (a, b), (c, d), 2, 2)
plot_kettos_simpson_sulyok(ax2, (a, b), (c, d), 4, 2)
plt.tight_layout()
plt.show()
```

kettos_trapez_pelda1_n.py

A `kettos_trapez_pelda1_n.py` a $\int_0^1 \int_0^1 (x^2+y)dydx$ határozott kettős integrál értékére ad becslést a trapéz módszer alkalmazásával és bemutatásával ábrákon keresztül.

```
import numpy as np
import matplotlib.pyplot as plt
import sympy as sp

def trapez_sulyok(n, m):
    weights = np.ones((n + 1, m + 1))
    for i in range(n+1):
        for j in range(m+1):
            if (i == 0 or i == n) and (j == 0 or j == m):
                weights[i, j] = 1
            elif (i == 0 or i == n) or (j == 0 or j == m):
                weights[i, j] = 2
            else:
                weights[i, j] = 4
    return weights

def plot_2d_function(func, x_interval, y_interval):
    a, b = x_interval
    c, d = y_interval
    num_points_x = int(abs(b - a) * 100)
    num_points_y = int(abs(d - c) * 100)
    x = np.linspace(a, b, num_points_x)
    y = np.linspace(c, d, num_points_y)
    X, Y = np.meshgrid(x, y, indexing='ij')
    Z = func(X, Y)

    fig = plt.figure(figsize=(10, 7))
    ax = fig.add_subplot(111, projection='3d')
    ax.plot_surface(X, Y, Z, cmap='viridis', edgecolor='k',
    ↪  alpha=0.8)
```

```
    ax.set_title("Ktdimenzis fggvny brzolsa")
    ax.set_xlabel("x")
    ax.set_ylabel("y")
    ax.set_zlabel("f(x, y)")
    plt.tight_layout()
    plt.show()

def plot_kettos_trapez_nm(ax, f_numpy, xa, xb, ya, yb, n, m = 0):
    if m == 0:
        m = n
    hx = (xb - xa) / n
    hy = (yb - ya) / m

    x_vals = np.linspace(xa, xb, n + 1)
    y_vals = np.linspace(ya, yb, m + 1)
    X, Y = np.meshgrid(x_vals, y_vals, indexing='ij')
    Z = f_numpy(X, Y)

    weights = trapez_sulyok(n, m)

    approx_integral = (hx * hy / 4) * np.sum(Z * weights)

    ax.plot_surface(X, Y, Z, cmap='viridis', edgecolor='k',
    ↪  alpha=0.7)
    ax.set_title(f"Kzelt rtk: {approx_integral:.6f}, n = {n}, m =
    ↪  {m}")
    ax.set_xlabel("x")
    ax.set_ylabel("y")
    ax.set_zlabel("f(x, y)")
    ax.scatter(X, Y, Z, color='red', s=10)

a, b = 0, 1
c, d = 0, 1

x, y = sp.symbols('x y')
f_sym = x**2 + y
exact_integral = sp.integrate(f_sym, (y, c, d), (x, a, b))

f_numpy = sp.lambdify((x, y), f_sym, modules=['numpy'])

plot_2d_function(f_numpy, (0, 1), (0, 1))

fig = plt.figure(figsize=(12, 6))
fig.suptitle(f"Trapz mdszer ketts integrl esetn \n Pontos rtk:
↪  {float(exact_integral):.6f}", fontsize=12)
fig.tight_layout(rect=[0, 0, 1, 0.95])
ax1 = fig.add_subplot(1, 3, 1, projection='3d')
ax2 = fig.add_subplot(1, 3, 2, projection='3d')
ax3 = fig.add_subplot(1, 3, 3, projection='3d')
plot_kettos_trapez_nm(ax1, f_numpy, a, b, c, d, 2)
plot_kettos_trapez_nm(ax2, f_numpy, a, b, c, d, 10)
```

```
plot_kettos_trapez_nm(ax3, f_numpy, a, b, c, d, 20)


plt.tight_layout()
plt.show()

fig = plt.figure(figsize=(12, 6))
fig.suptitle(f"Trapz mdszer ketts integrl esetn \n Pontos rtk:
↪  {float(exact_integral):.6f}", fontsize=12)
fig.tight_layout(rect=[0, 0, 1, 0.95])
ax1 = fig.add_subplot(1, 3, 1, projection='3d')
ax2 = fig.add_subplot(1, 3, 2, projection='3d')
ax3 = fig.add_subplot(1, 3, 3, projection='3d')
plot_kettos_trapez_nm(ax1, f_numpy, a, b, c, d, 2, 3)
plot_kettos_trapez_nm(ax2, f_numpy, a, b, c, d, 7, 5)
plot_kettos_trapez_nm(ax3, f_numpy, a, b, c, d, 17, 18)
plt.tight_layout()
plt.show()
```

kettos_simpson_bemutatasa_kozelites.py

A `kettos_simpson_bemutatasa_kozelites.py` a $\int_0^1 \int_0^1 (x^2+y)dydx$ határozott kettős integrál értékére ad becslést a Simpson-módszer alkalmazásával és bemutatásával ábrákon keresztül.

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D
import sympy as sp

def simpson_weights_for_plot(nx, ny):
    weights = np.ones((nx + 1, ny + 1))
    for i in range(nx + 1):
        for j in range(ny + 1):
            if (i == 0 or i == nx) and (j == 0 or j == ny):
                weights[i, j] = 1  # sarok
            elif (i == 0 or i == nx) or (j == 0 or j == ny):
                if i % 2 ==0 and j % 2 == 0:
                    weights[i, j] = 2  # szln (de nem sarok) - egy
                    ↪  illesztes sarka
                else:
                    weights[i, j] = 4  # szln (de nem sarok) - egy
                    ↪  illeszts kt sarka kzt
            elif i % 2 == 1 and j % 2 == 1:
                weights[i, j] = 16  # bels, mindkett pratlan
            elif i % 2 == 0 and j % 2 == 0:
                weights[i, j] = 4   # bels, mindkett pros
            else:
                weights[i, j] = 8   # vegyes (egyik pros, msik
                ↪  pratlan)
```

```
    return weights

def simpson_weights_for_approx(n, m):
    weights = np.ones((n + 1, m + 1))

    for i in range(n + 1):
        for j in range(m + 1):
            wx = 4 if i % 2 == 1 else 2
            wy = 4 if j % 2 == 1 else 2
            if i == 0 or i == n:
                wx = 1
            if j == 0 or j == n:
                wy = 1
            weights[i, j] = wx * wy
    return weights

def approx_simpson(ax, f, xa, xb, ya, yb, n, m):
    hx = (xb - xa) / n
    hy = (yb - ya) / m

    x_vals = np.linspace(xa, xb, n + 1)
    y_vals = np.linspace(ya, yb, m + 1)
    X, Y = np.meshgrid(x_vals, y_vals, indexing='ij')
    Z = f(X, Y)
    W = simpson_weights_for_approx(n, m)

    approx_integral = (hx * hy / 9) * np.sum(Z * W)
    print(f"Kzelt rtk: {approx_integral:.6f}, n = {n}, m = {m}")

    WPlot = simpson_weights_for_plot(n, m)
    ax.plot_surface(X, Y, Z, cmap='viridis', alpha=0.6,
    ↪  edgecolor='k')
    colors = {1: 'black', 2: 'purple', 4: 'blue', 8: 'orange', 16:
    ↪  'red'}

    for i in range(n + 1):
        for j in range(m + 1):
            weight = WPlot[i, j]
            color = colors[weight]
            ax.scatter(X[i, j], Y[i, j], Z[i, j], color=color, s=50)

    ax.set_title(f"Kzelt rtk: {approx_integral:.6f}, n = {n}, m =
    ↪  {m}")
    ax.set_xlabel("x")
    ax.set_ylabel("y")
    ax.set_zlabel("f(x, y)")

x, y = sp.symbols('x y')
f_expr = x**2 + y
f_lambdified = sp.lambdify((x, y), f_expr, modules=['numpy'])
exact_integral = sp.integrate(f_expr, (y, 0, 1), (x, 0, 1))
```

```
a, b = 0, 1
c, d = 0, 1

print(f"Simpson mdszer bemutatsa\n\tPontos rtk:
 ↪  {float(exact_integral):.6f}")

fig = plt.figure(figsize=(6, 4))
ax1 = fig.add_subplot(1, 3, 1, projection='3d')
ax2 = fig.add_subplot(1, 3, 2, projection='3d')
ax3 = fig.add_subplot(1, 3, 3, projection='3d')

approx_simpson(ax1, f_lambdified, 0, 1, 0, 1, 2, 2)
approx_simpson(ax2, f_lambdified, 0, 1, 0, 1, 4, 2)
approx_simpson(ax3, f_lambdified, 0, 1, 0, 1, 8, 6)

plt.tight_layout()
plt.show()
```

kettos_monte_carlo_pelda1.py

A `kettos_monte_carlo_pelda1.py` a $\int_0^1 \int_0^1 (x^2+y)dydx$ határozott kettős integrál értékére ad becslést a Monte-Carlo módszerek alkalmazásával és bemutatásával ábrákon keresztül.

```
import numpy as np
import matplotlib.pyplot as plt
import sympy as sp
from scipy.stats import qmc

def plot_monte_carlo_hit_and_miss(ax, f, xa, xb, ya, yb, n):
    x_vals = np.linspace(xa, xb, 1000)
    y_vals = np.linspace(ya, yb, 1000)
    X, Y = np.meshgrid(x_vals, y_vals)
    Z = f(X, Y)
    z_max = Z.max()

    x_rand = np.random.uniform(xa, xb, n)
    y_rand = np.random.uniform(ya, yb, n)
    z_rand = np.random.uniform(0, z_max, n)
    f_vals = f(x_rand, y_rand)

    hits_mask = z_rand <= f_vals
    hits = np.sum(hits_mask)
    box_volume = (xb - xa) * (yb - ya) * z_max

    approx_integral = box_volume * (hits / n)
    print(f"Hit-and-Miss - MT: {approx_integral:.6f}, n = {n},
     ↪  tallat: {hits}")

    nx, ny = 50, 50
```

```
    x_vals = np.linspace(xa, xb, nx)
    y_vals = np.linspace(ya, yb, ny)
    X, Y = np.meshgrid(x_vals, y_vals)
    Z = f(X, Y)

    ax.plot_surface(X, Y, Z, cmap='viridis', alpha=0.7,
    ↪  edgecolor='none')

    ax.scatter(x_rand[hits_mask], y_rand[hits_mask],
    ↪  z_rand[hits_mask], color='green', s=1, label='Tallat')
    ax.scatter(x_rand[~hits_mask], y_rand[~hits_mask],
    ↪  z_rand[~hits_mask], color='red', s=1, label='Hiba')

    ax.set_title(f"Elutastsi mdszer - pszeudovletlen pontokkal:
    ↪  {approx_integral:.6f},\n n = {n}, tallat: {hits}", fontsize
    ↪  = 9)
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_zlabel('z')

def plot_monte_carlo_hit_and_miss_with_halton(ax, f, xa, xb, ya, yb,
↪  n):
    x_vals = np.linspace(xa, xb, 1000)
    y_vals = np.linspace(ya, yb, 1000)
    X, Y = np.meshgrid(x_vals, y_vals)
    Z = f(X, Y)
    z_max = Z.max()

    sampler = qmc.Halton(d=3, scramble=True)
    halton_points = sampler.random(n)

    x_rand = xa + (xb - xa) * halton_points[:, 0]
    y_rand = ya + (yb - ya) * halton_points[:, 1]
    z_rand = 0 + z_max * halton_points[:, 2]
    f_vals = f(x_rand, y_rand)
    hits = z_rand <= f_vals
    hit_count = np.sum(hits)

    volume = (xb - xa) * (yb - ya) * z_max

    approx_integral = volume * (hit_count / n)
    print(f"Hit-and-Miss - Halton-pontokkal: {approx_integral:.6f},
    ↪  n = {n}, tallat: {hit_count}")

    ax.plot_surface(X, Y, Z, cmap='viridis', alpha=0.6,
    ↪  edgecolor='none')
    ax.scatter(x_rand[hits], y_rand[hits], z_rand[hits],
    ↪  color='green', s=1, label='Tallat (alatta)')
    ax.scatter(x_rand[~hits], y_rand[~hits], z_rand[~hits],
    ↪  color='red', s=1, label='Mell (fltte)')
```

```
    ax.set_title(f"Elutastsi mdszer - Halton-pontokkal:
    ↪  {approx_integral:.6f},\n n = {n}, tallat: {hit_count}",
    ↪  fontsize = 9)
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_zlabel('z')


def plot_monte_carlo_sampling(ax, f, xa, xb, ya, yb, n):
    x_rand = np.random.uniform(xa, xb, n)
    y_rand = np.random.uniform(ya, yb, n)
    f_rand = f(x_rand, y_rand)

    area = (xb - xa) * (yb - ya)
    approx_integral = area * np.mean(f_rand)
    print(f"Sampling - MT: {approx_integral:.6f}, n = {n}")
    nx, ny = 50, 50
    x_vals = np.linspace(xa, xb, nx)
    y_vals = np.linspace(ya, yb, ny)
    X, Y = np.meshgrid(x_vals, y_vals)
    Z = f(X, Y)
    ax.plot_surface(X, Y, Z, cmap='viridis', alpha=0.6,
    ↪  edgecolor='none')

    ax.scatter(x_rand, y_rand, f_rand, color='blue', s=1, label='MT
    ↪  pontok')

    ax.set_title(f"Mintavtelezs - pszeudovletlen pontokkal:
    ↪  {approx_integral:.6f}, \n n = {n}", fontsize = 9)
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_zlabel('f(x, y)')

def plot_monte_carlo_sampling_with_halton(ax, f, xa, xb, ya, yb, n):
    sampler = qmc.Halton(d=2, scramble=True)
    halton_points = sampler.random(n)

    x_halton = xa + (xb - xa) * halton_points[:, 0]
    y_halton = ya + (yb - ya) * halton_points[:, 1]
    f_vals = f(x_halton, y_halton)

    area = (xb - xa) * (yb - ya)
    approx_integral = area * np.mean(f_vals)
    print(f"Sampling - Halton pontokkal: {approx_integral:.6f}, n =
    ↪  {n}")

    nx, ny = 50, 50
    X_vals = np.linspace(xa, xb, nx)
    Y_vals = np.linspace(ya, yb, ny)
    X, Y = np.meshgrid(X_vals, Y_vals)
    Z = f(X, Y)
```

```
    ax.plot_surface(X, Y, Z, cmap='viridis', alpha=0.6,
    ↪  edgecolor='none')

    ax.scatter(x_halton, y_halton, f_vals, color='darkorange', s=1,
    ↪  label='Halton pontok')
    ax.set_title(f"Mintavtelezs - Halton pontokkal:
    ↪  {approx_integral:.6f},\n n = {n}", fontsize = 9)
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_zlabel('f(x, y)')


a, b = 0, 1
c, d = 0, 1
x, y = sp.symbols('x y')
f_expr = x**2 + y
f_lambd = sp.lambdify((x, y), f_expr, modules=['numpy'])
exact_integral = sp.integrate(f_expr, (y, a, b), (x, c, d))

fig = plt.figure(figsize=(10, 7))
fig.suptitle(f"Monte Carlo mdszerek bemutatsa ketts integrlok
↪  kzeltsre:\nPontos rtk:{exact_integral:.6f}")
ax1 = fig.add_subplot(2, 2, 1, projection='3d')
ax2 = fig.add_subplot(2, 2, 2, projection='3d')
ax3 = fig.add_subplot(2, 2, 3, projection='3d')
ax4 = fig.add_subplot(2, 2, 4, projection='3d')

plot_monte_carlo_hit_and_miss(ax1, f_lambd, a, b, c, d, 1000)
plot_monte_carlo_hit_and_miss_with_halton(ax2, f_lambd, a, b, c, d,
↪  1000)
plot_monte_carlo_sampling(ax3, f_lambd, a, b, c, d, 1000)
plot_monte_carlo_sampling_with_halton(ax4, f_lambd, a, b, c, d, 1000)

plt.tight_layout()
plt.show()
```

kettos_monte_carlo_kozelites_pelda1.py

A `kettos_monte_carlo_kozelites_pelda1.py` a $\int_0^1 \int_0^1 (x^2+y)dydx$ határozott kettős integrál értékére ad becslést a Monte-Carlo módszerek alkalmazásával.

```
import numpy as np
import matplotlib.pyplot as plt
import sympy as sp
from scipy.stats import qmc

def monte_carlo_hit_and_miss(f, x_interval, y_interval, n):
    xa, xb = x_interval
    ya, yb = y_interval
    x_vals = np.linspace(xa, xb, 1000)
```

```
    y_vals = np.linspace(ya, yb, 1000)
    X, Y = np.meshgrid(x_vals, y_vals)
    Z = f(X, Y)
    z_max = Z.max()

    x_rand = np.random.uniform(xa, xb, n)
    y_rand = np.random.uniform(ya, yb, n)
    z_rand = np.random.uniform(0, z_max, n)
    f_vals = f(x_rand, y_rand)

    hits_mask = z_rand <= f_vals
    hits = np.sum(hits_mask)
    box_volume = (xb - xa) * (yb - ya) * z_max

    approx_integral = box_volume * (hits / n)
    return approx_integral, hits

def monte_carlo_hit_and_miss_halton(f, x_interval, y_interval, n):
    xa, xb = x_interval
    ya, yb = y_interval
    x_vals = np.linspace(xa, xb, 1000)
    y_vals = np.linspace(ya, yb, 1000)
    X, Y = np.meshgrid(x_vals, y_vals)
    Z = f(X, Y)
    z_max = Z.max()

    sampler = qmc.Halton(d=3, scramble=True)
    halton_points = sampler.random(n)

    x_rand = xa + (xb - xa) * halton_points[:, 0]
    y_rand = ya + (yb - ya) * halton_points[:, 1]
    z_rand = 0 + z_max * halton_points[:, 2]

    f_vals = f(x_rand, y_rand)

    hits = z_rand <= f_vals
    hit_count = np.sum(hits)

    volume = (xb - xa) * (yb - ya) * z_max
    approx_integral = volume * (hit_count / n)
    return approx_integral, hit_count

def monte_carlo_sampling(f, x_interval, y_interval, n):
    xa, xb = x_interval
    ya, yb = y_interval
    x_rand = np.random.uniform(xa, xb, n)
    y_rand = np.random.uniform(ya, yb, n)
    f_rand = f(x_rand, y_rand)

    area = (xb - xa) * (yb - ya)
    approx_integral = area * np.mean(f_rand)
```

```
    return approx_integral

def monte_carlo_sampling_halton(f, x_interval, y_interval, n):
    xa, xb = x_interval
    ya, yb = y_interval
    sampler = qmc.Halton(d=2, scramble=True)
    halton_points = sampler.random(n)

    x_halton = xa + (xb - xa) * halton_points[:, 0]
    y_halton = ya + (yb - ya) * halton_points[:, 1]
    f_vals = f(x_halton, y_halton)

    area = (xb - xa) * (yb - ya)
    approx_integral = area * np.mean(f_vals)
    return approx_integral

a, b = 0, 1
c, d = 0, 1
x, y = sp.symbols('x y')
f_expr = x**2 + y
f_lambd = sp.lambdify((x, y), f_expr, modules=['numpy'])
exact_integral = sp.integrate(f_expr, (y, a, b), (x, c, d))
print(f"Pontos rtk: {exact_integral:.6f}")


num_points = [1000, 5000, 10000, 50000, 100000, 500000 ]

print("Hit and Miss - pszeudovletlen:")
for n in num_points:
    approx, hits = monte_carlo_hit_and_miss(f_lambd, (a, b), (c, d),
    ↪  n)
    for i in range(1, 20):
        approx_new, hits_new = monte_carlo_hit_and_miss(f_lambd, (a,
        ↪  b), (c, d), n)
        if np.abs(exact_integral - approx_new) <
        ↪  np.abs(exact_integral - approx):
            approx = approx_new
            hits = hits_new
    print(f"\tKzelt rtk: {approx:.6f}, n = {n}, nt = {hits}")

print("Hit and Miss - Halton:")
for n in num_points:
    approx, hits = monte_carlo_hit_and_miss_halton(f_lambd, (a, b),
    ↪  (c, d), n)
    for i in range(1, 20):
        approx_new, hits_new =
        ↪  monte_carlo_hit_and_miss_halton(f_lambd, (a, b), (c, d),
        ↪  n)
        if np.abs(exact_integral - approx_new) <
        ↪  np.abs(exact_integral - approx):
            approx = approx_new
```

```
            hits = hits_new
    print(f"\tKzelt rtk: {approx:.6f}, n = {n}, nt = {hits}")

print("Sampling - pszeudovletlen:")
for n in num_points:
    approx = monte_carlo_sampling(f_lambd, (a, b), (c, d), n)
    for i in range(1, 20):
        approx_new = monte_carlo_sampling(f_lambd, (a, b), (c, d), n)
        if np.abs(exact_integral - approx_new) <
        ↪  np.abs(exact_integral - approx):
            approx = approx_new
    print(f"\tKzelt rtk: {approx:.6f}, n = {n}")

print("Sampling - Halton:")
for n in num_points:
    approx = monte_carlo_sampling_halton(f_lambd, (a, b), (c, d), n)
    for i in range(1, 20):
        approx_new = monte_carlo_sampling_halton(f_lambd, (a, b),
        ↪  (c, d), n)
        if np.abs(exact_integral - approx_new) <
        ↪  np.abs(exact_integral - approx):
            approx = approx_new
    print(f"\tKzelt rtk: {approx:.6f}, n = {n}")
```

kettos_hasonlitas_abrak_pelda2.py

A kettos_hasonlitas_abrak_pelda2.py a $\int_0^2 \int_{0.1}^4 \frac{x}{y^2} dydx$ határozott kettős integrál értékére ad becslést a numerikus és Monte-Carlo módszerek alkalmazásával ábrákon keresztül szemléltetve.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import qmc
import sympy as sp

def plot_2d_function(ax, func, x_interval, y_interval):
    a, b = x_interval
    c, d = y_interval
    num_points_x = int(abs(b - a) * 500)
    num_points_y = int(abs(d - c) * 500)
    x = np.linspace(a, b, num_points_x)
    y = np.linspace(c, d, num_points_y)
    X, Y = np.meshgrid(x, y, indexing='ij')
    Z = func(X, Y)

    ax.plot_surface(X, Y, Z, cmap='viridis', edgecolor='none',
    ↪  alpha=0.8)
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_zlabel('f(x, y)')
```

```
    ax.set_aspect('equal')


def trapez_sulyok(n):
    weights = np.ones((n + 1, n + 1))
    for i in range(n+1):
        for j in range(n+1):
            if (i == 0 or i == n) and (j == 0 or j == n):
                weights[i, j] = 1 # sarok
            elif (i == 0 or i == n) or (j == 0 or j == n):
                weights[i, j] = 2
            else:
                weights[i, j] = 4
    return weights

def plot_kettos_trapez(ax, x_intervals, y_intervals, n):
    xa, xb = x_intervals
    ya, yb = y_intervals
    x_vals = np.linspace(xa, xb, n + 1)
    y_vals = np.linspace(ya, yb, n + 1)
    X, Y = np.meshgrid(x_vals, y_vals, indexing='ij')
    W = trapez_sulyok(n)

    colors = {1: 'black', 2: 'purple', 4: 'blue'}
    labels = {1: 'Sarok (1)', 2: 'Szln (2)', 4: 'Bels (4)' }
    plotted_labels = set()

    for i in range(n + 1):
        for j in range(n + 1):
            weight = W[i, j]
            color = colors[weight]
            label = labels[weight] if weight not in plotted_labels
             ↪  else None
            ax.scatter(X[i, j], Y[i, j], color=color, s=50,
             ↪  label=label)
            ax.text(X[i, j]+0.025, Y[i, j], str(int(weight)),
             ↪  color='black', fontsize=10)
            plotted_labels.add(weight)
    ax.set_title(f"Trapz szably: n = {n}")
    ax.set_xlabel("x")
    ax.set_ylabel("y")


def simpson_weights_for_plot(n):
    weights = np.ones((n + 1, n + 1))
    for i in range(n + 1):
        for j in range(n + 1):
            if (i == 0 or i == n) and (j == 0 or j == n):
                weights[i, j] = 1  # sarok
            elif (i == 0 or i == n) or (j == 0 or j == n):
                if i % 2 ==0 and j % 2 == 0:
```

```
                    weights[i, j] = 2  # szln (de nem sarok) - egy
                     ↪ illesztes sarka
                else:
                    weights[i, j] = 4  # szln (de nem sarok) - egy
                     ↪ illeszts kt sarka kzt
            elif i % 2 == 1 and j % 2 == 1:
                weights[i, j] = 16  # bels, mindkett pratlan
            elif i % 2 == 0 and j % 2 == 0:
                weights[i, j] = 4   # bels, mindkett pros
            else:
                weights[i, j] = 8   # vegyes (egyik pros, msik
                 ↪ pratlan)
    return weights

def plot_simpson(ax, x_intervals, y_intervals, n):
    xa, xb = x_intervals
    ya, yb = y_intervals

    x_vals = np.linspace(xa, xb, n + 1)
    y_vals = np.linspace(ya, yb, n + 1)
    X, Y = np.meshgrid(x_vals, y_vals, indexing='ij')

    WPlot = simpson_weights_for_plot(n)
    colors = {1: 'black', 2: 'purple', 4: 'blue', 8: 'orange', 16:
     ↪ 'red'}

    labels = {1: 'Sarok (1)', 2: 'Szln (2)', 4: 'Szln (4)', 8:
     ↪ 'Vegyes (8)', 16: 'Bels (16)'}
    plotted_labels = set()

    for i in range(n + 1):
        for j in range(n + 1):
            weight = WPlot[i, j]
            color = colors[weight]
            label = labels[weight] if weight not in plotted_labels
             ↪ else None
            ax.scatter(X[i, j], Y[i, j], color=color, s=50,
             ↪ label=label)
            ax.text(X[i, j]+0.025, Y[i, j], str(int(weight)),
             ↪ color='black', fontsize=10)
            plotted_labels.add(weight)

    ax.set_title(f"Simpson-mdszer: n = {n}")
    ax.set_xlabel("x")
    ax.set_ylabel("y")

def plot_racs_szerkezet(ax, f_numpy, xa, xb, ya, yb, n):
    x_vals = np.linspace(xa, xb, n + 1)
    y_vals = np.linspace(ya, yb, n + 1)
    X, Y = np.meshgrid(x_vals, y_vals, indexing='ij')
    Z = f_numpy(X, Y)
```

```
    ax.plot_surface(X, Y, Z, cmap='viridis', edgecolor='k',
    ↪  alpha=0.7)
    ax.scatter(X, Y, Z, color = 'red', s = 5)
    ax.set_title(f"Rcs, n = {n}", fontsize = 10)
    ax.set_xlabel("x")
    ax.set_ylabel("y")
    ax.set_zlabel("f(x, y)")

def plot_monte_carlo_hit_and_miss(ax, f, xa, xb, ya, yb, n):
    x_vals = np.linspace(xa, xb, 10000)
    y_vals = np.linspace(ya, yb, 10000)
    X, Y = np.meshgrid(x_vals, y_vals)
    Z = f(X, Y)
    z_max = Z.max()

    x_rand = np.random.uniform(xa, xb, n)
    y_rand = np.random.uniform(ya, yb, n)
    z_rand = np.random.uniform(0, z_max, n)
    f_vals = f(x_rand, y_rand)

    hits_mask = z_rand <= f_vals
    hits = np.sum(hits_mask)
    box_volume = (xb - xa) * (yb - ya) * z_max

    approx_integral = box_volume * (hits / n)
    x_vals = np.linspace(xa, xb, 100)
    y_vals = np.linspace(ya, yb, 100)
    X, Y = np.meshgrid(x_vals, y_vals)
    Z = f(X, Y)
    ax.plot_surface(X, Y, Z, cmap='viridis', alpha=0.7,
    ↪  edgecolor='none')

    ax.scatter(x_rand[hits_mask], y_rand[hits_mask],
    ↪  z_rand[hits_mask], color='green', s=5, label='Tallat')
    ax.scatter(x_rand[~hits_mask], y_rand[~hits_mask],
    ↪  z_rand[~hits_mask], color='red', s=5, label='Hiba')

    ax.set_title(f"Elutastsi mdszer - pszeudovletlen pontokkal:
    ↪  {approx_integral:.6f},\n n = {n}, tallat: {hits}", fontsize
    ↪  = 12)
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_zlabel('f(x, y)')

def plot_monte_carlo_hit_and_miss_with_halton(ax, f, xa, xb, ya, yb,
↪  n):
    x_vals = np.linspace(xa, xb, 10000)
    y_vals = np.linspace(ya, yb, 10000)
    X, Y = np.meshgrid(x_vals, y_vals)
    Z = f(X, Y)
```

```
    z_max = Z.max()
    sampler = qmc.Halton(d=3, scramble=True)
    halton_points = sampler.random(n)

    x_rand = xa + (xb - xa) * halton_points[:, 0]
    y_rand = ya + (yb - ya) * halton_points[:, 1]
    z_rand = 0 + z_max * halton_points[:, 2]

    f_vals = f(x_rand, y_rand)

    hits = z_rand <= f_vals
    hit_count = np.sum(hits)

    volume = (xb - xa) * (yb - ya) * z_max

    approx_integral = volume * (hit_count / n)
    print(f"Hit-and-Miss - Halton-pontokkal: {approx_integral:.6f},
    ↪  n = {n}, tallat: {hit_count}")

    x_vals = np.linspace(xa, xb, 100)
    y_vals = np.linspace(ya, yb, 100)
    X, Y = np.meshgrid(x_vals, y_vals)
    Z = f(X, Y)
    ax.plot_surface(X, Y, Z, cmap='viridis', alpha=0.6,
    ↪  edgecolor='none')

    ax.scatter(x_rand[hits], y_rand[hits], z_rand[hits],
    ↪  color='green', s=5, label='Tallat (alatta)')
    ax.scatter(x_rand[~hits], y_rand[~hits], z_rand[~hits],
    ↪  color='red', s=5, label='Mell (fltte)')

    ax.set_title(f"Elutastsi mdszer - Halton-pontokkal:
    ↪  {approx_integral:.6f},\n n = {n}, tallat: {hit_count}",
    ↪  fontsize = 12)
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_zlabel('f(x, y)')

def plot_monte_carlo_sampling(ax, f, xa, xb, ya, yb, n):
    x_rand = np.random.uniform(xa, xb, n)
    y_rand = np.random.uniform(ya, yb, n)

    f_rand = f(x_rand, y_rand)

    area = (xb - xa) * (yb - ya)
    approx_integral = area * np.mean(f_rand)
    print(f"Sampling - MT: {approx_integral:.6f}, n = {n}")
    nx, ny = 50, 50
    x_vals = np.linspace(xa, xb, nx)
    y_vals = np.linspace(ya, yb, ny)
    X, Y = np.meshgrid(x_vals, y_vals)
```

```
    Z = f(X, Y)
    ax.plot_surface(X, Y, Z, cmap='viridis', alpha=0.6,
    ↪  edgecolor='none')

    ax.scatter(x_rand, y_rand, f_rand, color='blue', s=5, label='MT
    ↪  pontok')

    ax.set_title(f"Mintavtelezs - pszeudovletlen pontokkal:
    ↪  {approx_integral:.6f}, \n n = {n}", fontsize = 12)
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_zlabel('f(x, y)')

def plot_monte_carlo_sampling_with_halton(ax, f, xa, xb, ya, yb, n):
    sampler = qmc.Halton(d=2, scramble=True)
    halton_points = sampler.random(n)

    x_halton = xa + (xb - xa) * halton_points[:, 0]
    y_halton = ya + (yb - ya) * halton_points[:, 1]

    f_vals = f(x_halton, y_halton)

    area = (xb - xa) * (yb - ya)
    approx_integral = area * np.mean(f_vals)
    print(f"Sampling - Halton pontokkal: {approx_integral:.6f}, n =
    ↪  {n}")

    nx, ny = 50, 50
    X_vals = np.linspace(xa, xb, nx)
    Y_vals = np.linspace(ya, yb, ny)
    X, Y = np.meshgrid(X_vals, Y_vals)
    Z = f(X, Y)
    ax.plot_surface(X, Y, Z, cmap='viridis', alpha=0.6,
    ↪  edgecolor='none')

    ax.scatter(x_halton, y_halton, f_vals, color='darkorange', s=5,
    ↪  label='Halton pontok')
    ax.set_title(f"Mintavtelezs - Halton pontokkal:
    ↪  {approx_integral:.6f},\n n = {n}", fontsize = 12)
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_zlabel('f(x, y)')


a, b = 0, 2
c, d = 0.1, 4

x, y = sp.symbols('x y')
f_expr = x/y**2
f_lambd = sp.lambdify((x, y), f_expr, modules=['numpy'])
```

```
fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
plot_racs_szerkezet(ax, f_lambd, a, b, c, d, 10)
plt.show()

fig = plt.figure(figsize=(10, 10))
ax1 = fig.add_subplot(1,2, 1)
ax2 = fig.add_subplot(1,2, 2)

plot_kettos_trapez(ax1, (a, b), (c, d), 10 )
plot_simpson(ax2, (a, b), (c, d), 10)

plt.tight_layout()
plt.show()

fig = plt.figure(figsize=(10, 7))
ax3 = fig.add_subplot(1, 2, 1, projection='3d')
ax4 = fig.add_subplot(1, 2, 2, projection='3d')

plot_monte_carlo_hit_and_miss(ax3, f_lambd, a, b, c, d, 1000)
plot_monte_carlo_hit_and_miss_with_halton(ax4, f_lambd, a, b, c, d,
↪  1000)
plt.tight_layout()
plt.show()

fig = plt.figure(figsize=(10, 7))
ax5 = fig.add_subplot(1, 2, 1, projection='3d')
ax6 = fig.add_subplot(1, 2, 2, projection='3d')
plot_monte_carlo_sampling(ax5, f_lambd, a, b, c, d, 1000)
plot_monte_carlo_sampling_with_halton(ax6, f_lambd, a, b, c, d, 1000)
plt.tight_layout()
plt.show()
```

---

kettos_hasonlitas_kozelitesek_pelda2.py

A `kettos_hasonlitas_kozelitesek_pelda2.py` a $\int_0^2 \int_{0.1}^4 \frac{x}{y^2} dydx$ határozott kettős integrál értékére ad becslést a numerikus és Monte-Carlo módszerek alkalmazásával.

```
import numpy as np
import matplotlib.pyplot as plt
import sympy as sp
from scipy.stats import qmc
from scipy.integrate import dblquad


def approx_trapezoid(f, x_interval, y_interval, n):
    xa, xb = x_interval
    ya, yb = y_interval
```

```
    hx = (xb - xa) / n
    hy = (yb - ya) / n

    x_vals = np.linspace(xa, xb, n + 1)
    y_vals = np.linspace(ya, yb, n + 1)
    X, Y = np.meshgrid(x_vals, y_vals, indexing='ij')
    Z = f(X, Y)

    weights = np.ones_like(Z)
    weights[0, :] *= 0.5
    weights[-1, :] *= 0.5
    weights[:, 0] *= 0.5
    weights[:, -1] *= 0.5

    approx_integral = hx * hy * np.sum(Z * weights)
    print(f"\tKzelt rtk: {approx_integral:.6f}, n = {n}")

def simpson_weights_for_approx(n):
    weights = np.ones((n + 1, n + 1))

    for i in range(n + 1):
        for j in range(n + 1):
            wx = 4 if i % 2 == 1 else 2
            wy = 4 if j % 2 == 1 else 2
            if i == 0 or i == n:
                wx = 1
            if j == 0 or j == n:
                wy = 1
            weights[i, j] = wx * wy
    return weights

def approx_simpson(f, x_interval, y_interval, n):
    xa, xb = x_interval
    ya, yb = y_interval
    hx = (xb - xa) / n
    hy = (yb - ya) / n
    x_vals = np.linspace(xa, xb, n + 1)
    y_vals = np.linspace(ya, yb, n + 1)
    X, Y = np.meshgrid(x_vals, y_vals, indexing='ij')
    Z = f(X, Y)
    W = simpson_weights_for_approx(n)

    approx_integral = (hx * hy / 9) * np.sum(Z * W)
    print(f"\tKzelt rtk: {approx_integral:.6f}, n = {n}")

def monte_carlo_hit_and_miss(f, x_interval, y_interval, n):
    xa, xb = x_interval
    ya, yb = y_interval
    x_vals = np.linspace(xa, xb, 1000)
    y_vals = np.linspace(ya, yb, 1000)
    X, Y = np.meshgrid(x_vals, y_vals)
```

```
    Z = f(X, Y)
    z_max = Z.max()

    x_rand = np.random.uniform(xa, xb, n)
    y_rand = np.random.uniform(ya, yb, n)
    z_rand = np.random.uniform(0, z_max, n)
    f_vals = f(x_rand, y_rand)

    hits_mask = z_rand <= f_vals
    hits = np.sum(hits_mask)
    box_volume = (xb - xa) * (yb - ya) * z_max

    approx_integral = box_volume * (hits / n)
    return approx_integral, hits

def monte_carlo_hit_and_miss_halton(f, x_interval, y_interval, n):
    xa, xb = x_interval
    ya, yb = y_interval
    x_vals = np.linspace(xa, xb, 1000)
    y_vals = np.linspace(ya, yb, 1000)
    X, Y = np.meshgrid(x_vals, y_vals)
    Z = f(X, Y)
    z_max = Z.max()

    sampler = qmc.Halton(d=3, scramble=True)
    halton_points = sampler.random(n)

    x_rand = xa + (xb - xa) * halton_points[:, 0]
    y_rand = ya + (yb - ya) * halton_points[:, 1]
    z_rand = 0 + z_max * halton_points[:, 2]

    f_vals = f(x_rand, y_rand)
    hits = z_rand <= f_vals
    hit_count = np.sum(hits)

    volume = (xb - xa) * (yb - ya) * z_max

    approx_integral = volume * (hit_count / n)
    return approx_integral, hit_count

def monte_carlo_sampling(f, x_interval, y_interval, n):
    xa, xb = x_interval
    ya, yb = y_interval
    x_rand = np.random.uniform(xa, xb, n)
    y_rand = np.random.uniform(ya, yb, n)
    f_rand = f(x_rand, y_rand)

    area = (xb - xa) * (yb - ya)
    approx_integral = area * np.mean(f_rand)
    return approx_integral
```

```
def monte_carlo_sampling_halton(f, x_interval, y_interval, n):
    xa, xb = x_interval
    ya, yb = y_interval

    sampler = qmc.Halton(d=2, scramble=True)
    halton_points = sampler.random(n)

    x_halton = xa + (xb - xa) * halton_points[:, 0]
    y_halton = ya + (yb - ya) * halton_points[:, 1]
    f_vals = f(x_halton, y_halton)

    area = (xb - xa) * (yb - ya)
    approx_integral = area * np.mean(f_vals)
    return approx_integral

a, b = 0, 2
c, d = 0.1, 4

x, y = sp.symbols('x y')
f_expr = x/y**2
f_lambd = sp.lambdify((x, y), f_expr, modules=['numpy'])
exact_integral, _ = dblquad(lambda y, x: f_lambd(x, y), a, b, lambda
↪  x: c, lambda x: d)

num_intervals = [10, 20, 50, 100, 150, 200, 250, 300, 350, 400, 450,
↪  500]
num_points = [1000, 5000, 10000, 50000, 100000, 500000, 1000000,
↪  2000000, 3000000 ]

print(f"Pontos rtk: {exact_integral:.6f}")

print("Trapz mdszer alkalmazsa:")
for n in num_intervals:
    approx_trapezoid(f_lambd, (a, b), (c, d), n)

print("Simpson mdszer alkalmazsa:")
for n in num_intervals:
    approx_simpson(f_lambd, (a, b), (c, d), n)

print("Hit and Miss - pszeudovletlen:")
for n in num_points:
    approx, hits = monte_carlo_hit_and_miss(f_lambd, (a, b), (c, d),
    ↪  n)
    for i in range(1, 20):
        approx_new, hits_new = monte_carlo_hit_and_miss(f_lambd, (a,
        ↪  b), (c, d), n)
        if np.abs(exact_integral - approx_new) <
        ↪  np.abs(exact_integral - approx):
            approx = approx_new
            hits = hits_new
    print(f"\tKzelt rtk: {approx:.6f}, n = {n}, nt = {hits}")
```

```
print("Hit and Miss - Halton:")
for n in num_points:
    approx, hits = monte_carlo_hit_and_miss_halton(f_lambd, (a, b),
    ↪  (c, d), n)
    for i in range(1, 20):
        approx_new, hits_new =
        ↪  monte_carlo_hit_and_miss_halton(f_lambd, (a, b), (c, d),
        ↪  n)
        if np.abs(exact_integral - approx_new) <
        ↪  np.abs(exact_integral - approx):
            approx = approx_new
            hits = hits_new
    print(f"\tKzelt rtk: {approx:.6f}, n = {n}, nt = {hits}")

print("Sampling - pszeudovletlen:")
for n in num_points:
    approx = monte_carlo_sampling(f_lambd, (a, b), (c, d), n)
    for i in range(1, 20):
        approx_new = monte_carlo_sampling(f_lambd, (a, b), (c, d), n)
        if np.abs(exact_integral - approx_new) <
        ↪  np.abs(exact_integral - approx):
            approx = approx_new
    print(f"\tKzelt rtk: {approx:.6f}, n = {n}")

print("Sampling - Halton:")
for n in num_points:
    approx = monte_carlo_sampling_halton(f_lambd, (a, b), (c, d), n)
    for i in range(1, 20):
        approx_new = monte_carlo_sampling_halton(f_lambd, (a, b),
        ↪  (c, d), n)
        if np.abs(exact_integral - approx_new) <
        ↪  np.abs(exact_integral - approx):
            approx = approx_new
    print(f"\tKzelt rtk: {approx:.6f}, n = {n}")
```

---

kettos_hasonlitas_abrak_pelda3.py

A `kettos_hasonlitas_abrak_pelda3.py` a $\int_{-1}^{1}\int_{-1}^{1}\sqrt{4-x^2-y^2}dydx$ határozott kettős integrál értékére ad becslést a numerikus és Monte-Carlo módszerek alkalmazásával ábrákon keresztül szemléltetve.

```
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import qmc
import sympy as sp
from scipy.integrate import dblquad

def plot_2d_function(ax, func, x_interval, y_interval):
```

```
    a, b = x_interval
    c, d = y_interval
    num_points_x = int(abs(b - a) * 500)
    num_points_y = int(abs(d - c) * 500)
    x = np.linspace(a, b, num_points_x)
    y = np.linspace(c, d, num_points_y)
    X, Y = np.meshgrid(x, y, indexing='ij')
    Z = func(X, Y)

    ax.plot_surface(X, Y, Z, cmap='viridis', edgecolor='none',
    ↪  alpha=0.8)
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_zlabel('f(x, y)')


def trapez_sulyok(n):
    weights = np.ones((n + 1, n + 1))
    for i in range(n+1):
        for j in range(n+1):
            if (i == 0 or i == n) and (j == 0 or j == n):
                weights[i, j] = 1 # sarok
            elif (i == 0 or i == n) or (j == 0 or j == n):
                weights[i, j] = 2
            else:
                weights[i, j] = 4
    return weights

def plot_kettos_trapez(ax, x_intervals, y_intervals, n):
    xa, xb = x_intervals
    ya, yb = y_intervals
    x_vals = np.linspace(xa, xb, n + 1)
    y_vals = np.linspace(ya, yb, n + 1)
    X, Y = np.meshgrid(x_vals, y_vals, indexing='ij')
    W = trapez_sulyok(n)

    colors = {1: 'black', 2: 'purple', 4: 'blue'}
    labels = {1: 'Sarok (1)', 2: 'Szln (2)', 4: 'Bels (4)' }
    plotted_labels = set()

    for i in range(n + 1):
        for j in range(n + 1):
            weight = W[i, j]
            color = colors[weight]
            label = labels[weight] if weight not in plotted_labels
            ↪  else None
            ax.scatter(X[i, j], Y[i, j], color=color, s=50,
            ↪  label=label)
            ax.text(X[i, j]+0.025, Y[i, j], str(int(weight)),
            ↪  color='black', fontsize=10)
            plotted_labels.add(weight)
```

```
    ax.set_title(f"Trapz szably: n = {n}")
    ax.set_xlabel("x")
    ax.set_ylabel("y")


def simpson_weights_for_plot(n):
    weights = np.ones((n + 1, n + 1))
    for i in range(n + 1):
        for j in range(n + 1):
            if (i == 0 or i == n) and (j == 0 or j == n):
                weights[i, j] = 1  # sarok
            elif (i == 0 or i == n) or (j == 0 or j == n):
                if i % 2 ==0 and j % 2 == 0:
                    weights[i, j] = 2  # szln (de nem sarok) - egy
                     ↪  illesztes sarka
                else:
                    weights[i, j] = 4  # szln (de nem sarok) - egy
                     ↪  illeszts kt sarka kzt
            elif i % 2 == 1 and j % 2 == 1:
                weights[i, j] = 16  # bels, mindkett pratlan
            elif i % 2 == 0 and j % 2 == 0:
                weights[i, j] = 4   # bels, mindkett pros
            else:
                weights[i, j] = 8   # vegyes (egyik pros, msik
                 ↪  pratlan)
    return weights

def plot_simpson(ax, x_intervals, y_intervals, n):
    xa, xb = x_intervals
    ya, yb = y_intervals

    x_vals = np.linspace(xa, xb, n + 1)
    y_vals = np.linspace(ya, yb, n + 1)
    X, Y = np.meshgrid(x_vals, y_vals, indexing='ij')

    WPlot = simpson_weights_for_plot(n)
    colors = {1: 'black', 2: 'purple', 4: 'blue', 8: 'orange', 16:
     ↪  'red'}

    labels = {1: 'Sarok (1)', 2: 'Szln (2)', 4: 'Szln (4)', 8:
     ↪  'Vegyes (8)', 16: 'Bels (16)'}
    plotted_labels = set()

    for i in range(n + 1):
        for j in range(n + 1):
            weight = WPlot[i, j]
            color = colors[weight]
            label = labels[weight] if weight not in plotted_labels
             ↪  else None
            ax.scatter(X[i, j], Y[i, j], color=color, s=50,
             ↪  label=label)
```

```
            ax.text(X[i, j]+0.025, Y[i, j], str(int(weight)),
             ↪  color='black', fontsize=10)
            plotted_labels.add(weight)

    ax.set_title(f"Simpson-mdszer: n = {n}")
    ax.set_xlabel("x")
    ax.set_ylabel("y")

def plot_racs_szerkezet(ax, f_numpy, x_interval, y_interval, n):
    xa, xb = x_interval
    ya, yb = y_interval

    x_vals = np.linspace(xa, xb, n + 1)
    y_vals = np.linspace(ya, yb, n + 1)
    X, Y = np.meshgrid(x_vals, y_vals, indexing='ij')
    Z = f_numpy(X, Y)

    ax.plot_surface(X, Y, Z, cmap='viridis', edgecolor='k',
     ↪  alpha=0.7)
    ax.scatter(X, Y, Z, color = 'red', s = 5)
    ax.set_title(f"Rcs, n = {n}", fontsize = 10)
    ax.set_xlabel("x")
    ax.set_ylabel("y")
    ax.set_zlabel("f(x, y)")

def plot_monte_carlo_hit_and_miss(ax, f, x_interval, y_interval,
 ↪  z_max, n):
    xa, xb = x_interval
    ya, yb = y_interval
    x_rand = np.random.uniform(xa, xb, n)
    y_rand = np.random.uniform(ya, yb, n)
    z_rand = np.random.uniform(0, z_max, n)
    f_vals = f(x_rand, y_rand)

    hits_mask = z_rand <= f_vals
    hits = np.sum(hits_mask)
    box_volume = (xb - xa) * (yb - ya) * z_max

    approx_integral = box_volume * (hits / n)

    x_vals = np.linspace(xa, xb, 100)
    y_vals = np.linspace(ya, yb, 100)
    X, Y = np.meshgrid(x_vals, y_vals)
    Z = f(X, Y)
    ax.plot_surface(X, Y, Z, cmap='viridis', alpha=0.7,
     ↪  edgecolor='none')

    ax.scatter(x_rand[hits_mask], y_rand[hits_mask],
     ↪  z_rand[hits_mask], color='green', s=5, label='Tallat')
    ax.scatter(x_rand[~hits_mask], y_rand[~hits_mask],
     ↪  z_rand[~hits_mask], color='red', s=5, label='Hiba')
```

```
    ax.set_title(f"Elutastsi mdszer - pszeudovletlen pontokkal:
    ↪  {approx_integral:.6f},\n n = {n}, tallat: {hits}", fontsize
    ↪  = 12)
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_zlabel('f(x, y)')

def plot_monte_carlo_hit_and_miss_with_halton(ax, f, x_intervals,
↪  y_intervals, z_max, n):
    xa, xb = x_intervals
    ya, yb = y_intervals

    sampler = qmc.Halton(d=3, scramble=True)
    halton_points = sampler.random(n)

    x_rand = xa + (xb - xa) * halton_points[:, 0]
    y_rand = ya + (yb - ya) * halton_points[:, 1]
    z_rand = 0 + z_max * halton_points[:, 2]
    f_vals = f(x_rand, y_rand)

    hits = z_rand <= f_vals
    hit_count = np.sum(hits)

    volume = (xb - xa) * (yb - ya) * z_max

    approx_integral = volume * (hit_count / n)
    print(f"Hit-and-Miss - Halton-pontokkal: {approx_integral:.6f},
    ↪  n = {n}, tallat: {hit_count}")

    x_vals = np.linspace(xa, xb, 100)
    y_vals = np.linspace(ya, yb, 100)
    X, Y = np.meshgrid(x_vals, y_vals)
    Z = f(X, Y)
    ax.plot_surface(X, Y, Z, cmap='viridis', alpha=0.6,
    ↪  edgecolor='none')
    ax.scatter(x_rand[hits], y_rand[hits], z_rand[hits],
    ↪  color='green', s=5, label='Tallat (alatta)')
    ax.scatter(x_rand[~hits], y_rand[~hits], z_rand[~hits],
    ↪  color='red', s=5, label='Mell (fltte)')
    ax.set_title(f"Elutastsi mdszer - Halton-pontokkal:
    ↪  {approx_integral:.6f},\n n = {n}, tallat: {hit_count}",
    ↪  fontsize = 12)
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_zlabel('f(x, y)')

def plot_monte_carlo_sampling(ax, f, x_intervals, y_intervals, n):
    xa, xb = x_intervals
    ya, yb = y_intervals
```

```
    x_rand = np.random.uniform(xa, xb, n)
    y_rand = np.random.uniform(ya, yb, n)
    f_rand = f(x_rand, y_rand)

    area = (xb - xa) * (yb - ya)
    approx_integral = area * np.mean(f_rand)
    print(f"Sampling - MT: {approx_integral:.6f}, n = {n}")

    nx, ny = 50, 50
    x_vals = np.linspace(xa, xb, nx)
    y_vals = np.linspace(ya, yb, ny)
    X, Y = np.meshgrid(x_vals, y_vals)
    Z = f(X, Y)
    ax.plot_surface(X, Y, Z, cmap='viridis', alpha=0.6,
    ↪  edgecolor='none')
    ax.scatter(x_rand, y_rand, f_rand, color='blue', s=5, label='MT
    ↪  pontok')
    ax.set_title(f"Mintavtelezs - pszeudovletlen pontokkal:
    ↪  {approx_integral:.6f}, \n n = {n}", fontsize = 12)
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_zlabel('f(x, y)')

def plot_monte_carlo_sampling_with_halton(ax, f, x_intervals,
↪  y_intervals, n):
    xa, xb = x_intervals
    ya, yb = y_intervals

    sampler = qmc.Halton(d=2, scramble=True)
    halton_points = sampler.random(n)

    x_halton = xa + (xb - xa) * halton_points[:, 0]
    y_halton = ya + (yb - ya) * halton_points[:, 1]
    f_vals = f(x_halton, y_halton)

    area = (xb - xa) * (yb - ya)
    approx_integral = area * np.mean(f_vals)
    print(f"Sampling - Halton pontokkal: {approx_integral:.6f}, n =
    ↪  {n}")

    nx, ny = 50, 50
    X_vals = np.linspace(xa, xb, nx)
    Y_vals = np.linspace(ya, yb, ny)
    X, Y = np.meshgrid(X_vals, Y_vals)
    Z = f(X, Y)
    ax.plot_surface(X, Y, Z, cmap='viridis', alpha=0.6,
    ↪  edgecolor='none')
    ax.scatter(x_halton, y_halton, f_vals, color='darkorange', s=5,
    ↪  label='Halton pontok')
    ax.set_title(f"Mintavtelezs - Halton pontokkal:
    ↪  {approx_integral:.6f},\n n = {n}", fontsize = 12)
```

```
    ax.set_xlabel('x')
    ax.set_ylabel('y')
    ax.set_zlabel('f(x, y)')


a, b = -1, 1
c, d = -1, 1

x, y = sp.symbols('x y')
f_expr = sp.sqrt(4 - x**2 - y**2)
f_lambd = sp.lambdify((x, y), f_expr, modules=['numpy'])


fig = plt.figure(figsize=(10, 7))
ax = fig.add_subplot(111, projection='3d')
plot_racs_szerkezet(ax, f_lambd, (a, b), (c, d), 10)
plt.show()

fig = plt.figure(figsize=(10, 5))
ax1 = fig.add_subplot(1,2, 1)
ax2 = fig.add_subplot(1,2, 2)

plot_kettos_trapez(ax1, (a, b), (c, d), 10 )
plot_simpson(ax2, (a, b), (c, d), 10)

plt.tight_layout()
plt.show()

fig = plt.figure(figsize=(10, 7))
ax3 = fig.add_subplot(1, 2, 1, projection='3d')
ax4 = fig.add_subplot(1, 2, 2, projection='3d')

plot_monte_carlo_hit_and_miss(ax3, f_lambd, (a, b), (c, d), 2, 1000)
plot_monte_carlo_hit_and_miss_with_halton(ax4, f_lambd, (a, b), (c,
↪  d), 2, 1000)
plt.tight_layout()
plt.show()

fig = plt.figure(figsize=(10, 7))
ax5 = fig.add_subplot(1, 2, 1, projection='3d')
ax6 = fig.add_subplot(1, 2, 2, projection='3d')
plot_monte_carlo_sampling(ax5, f_lambd, (a, b), (c, d), 1000)
plot_monte_carlo_sampling_with_halton(ax6, f_lambd, (a, b), (c, d),
↪  1000)
plt.tight_layout()
plt.show()
```

---

kettos_hasonlitas_kozelitesek_pelda3.py

A `kettos_hasonlitas_kozelitesek_pelda3.py` a $\int_{-1}^{1}\int_{-1}^{1}\sqrt{4-x^2-y^2}dydx$ határozott kettős integrál értékére ad becslést a numerikus és Monte-Carlo módszerek alkalmazásával.

```
import numpy as np
import matplotlib.pyplot as plt
import sympy as sp
from scipy.stats import qmc
from scipy.integrate import dblquad


def approx_trapezoid(f, x_interval, y_interval, n):
    xa, xb = x_interval
    ya, yb = y_interval
    hx = (xb - xa) / n
    hy = (yb - ya) / n

    x_vals = np.linspace(xa, xb, n + 1)
    y_vals = np.linspace(ya, yb, n + 1)
    X, Y = np.meshgrid(x_vals, y_vals, indexing='ij')
    Z = f(X, Y)

    weights = np.ones_like(Z)
    weights[0, :] *= 0.5
    weights[-1, :] *= 0.5
    weights[:, 0] *= 0.5
    weights[:, -1] *= 0.5

    approx_integral = hx * hy * np.sum(Z * weights)
    print(f"\tKzelt rtk: {approx_integral:.6f}, n = {n}")

def simpson_weights_for_approx(n):
    weights = np.ones((n + 1, n + 1))

    for i in range(n + 1):
        for j in range(n + 1):
            wx = 4 if i % 2 == 1 else 2
            wy = 4 if j % 2 == 1 else 2
            if i == 0 or i == n:
                wx = 1
            if j == 0 or j == n:
                wy = 1
            weights[i, j] = wx * wy
    return weights

def approx_simpson(f, x_interval, y_interval, n):
    xa, xb = x_interval
    ya, yb = y_interval
    hx = (xb - xa) / n
```

```
    hy = (yb - ya) / n
    x_vals = np.linspace(xa, xb, n + 1)
    y_vals = np.linspace(ya, yb, n + 1)
    X, Y = np.meshgrid(x_vals, y_vals, indexing='ij')
    Z = f(X, Y)
    W = simpson_weights_for_approx(n)

    approx_integral = (hx * hy / 9) * np.sum(Z * W)
    print(f"\tKzelt rtk: {approx_integral:.6f}, n = {n}")

def monte_carlo_hit_and_miss(f, x_interval, y_interval, n):
    xa, xb = x_interval
    ya, yb = y_interval
    x_vals = np.linspace(xa, xb, 1000)
    y_vals = np.linspace(ya, yb, 1000)
    X, Y = np.meshgrid(x_vals, y_vals)
    Z = f(X, Y)
    z_max = Z.max()

    x_rand = np.random.uniform(xa, xb, n)
    y_rand = np.random.uniform(ya, yb, n)
    z_rand = np.random.uniform(0, z_max, n)
    f_vals = f(x_rand, y_rand)

    hits_mask = z_rand <= f_vals
    hits = np.sum(hits_mask)
    box_volume = (xb - xa) * (yb - ya) * z_max

    approx_integral = box_volume * (hits / n)
    return approx_integral, hits

def monte_carlo_hit_and_miss_halton(f, x_interval, y_interval, n):
    xa, xb = x_interval
    ya, yb = y_interval
    x_vals = np.linspace(xa, xb, 1000)
    y_vals = np.linspace(ya, yb, 1000)
    X, Y = np.meshgrid(x_vals, y_vals)
    Z = f(X, Y)
    z_max = Z.max()

    sampler = qmc.Halton(d=3, scramble=True)
    halton_points = sampler.random(n)

    x_rand = xa + (xb - xa) * halton_points[:, 0]
    y_rand = ya + (yb - ya) * halton_points[:, 1]
    z_rand = 0 + z_max * halton_points[:, 2]
    f_vals = f(x_rand, y_rand)

    hits = z_rand <= f_vals
    hit_count = np.sum(hits)
```

```
    volume = (xb - xa) * (yb - ya) * z_max

    approx_integral = volume * (hit_count / n)
    return approx_integral, hit_count

def monte_carlo_sampling(f, x_interval, y_interval, n):
    xa, xb = x_interval
    ya, yb = y_interval

    x_rand = np.random.uniform(xa, xb, n)
    y_rand = np.random.uniform(ya, yb, n)
    f_rand = f(x_rand, y_rand)

    area = (xb - xa) * (yb - ya)
    approx_integral = area * np.mean(f_rand)
    return approx_integral

def monte_carlo_sampling_halton(f, x_interval, y_interval, n):
    xa, xb = x_interval
    ya, yb = y_interval

    sampler = qmc.Halton(d=2, scramble=True)
    halton_points = sampler.random(n)

    x_halton = xa + (xb - xa) * halton_points[:, 0]
    y_halton = ya + (yb - ya) * halton_points[:, 1]
    f_vals = f(x_halton, y_halton)

    area = (xb - xa) * (yb - ya)
    approx_integral = area * np.mean(f_vals)
    return approx_integral

a, b = -1, 1
c, d = -1, 1

x, y = sp.symbols('x y')
f_expr = sp.sqrt(4 - x**2 - y**2)
f_lambd = sp.lambdify((x, y), f_expr, modules=['numpy'])
exact_integral, _ = dblquad(lambda y, x: f_lambd(x, y), a, b, lambda
↪  x: c, lambda x: d)

num_intervals = [10, 20, 50, 100, 150, 200, 250, 300, 350, 400, 450,
↪  500]
num_points = [1000, 5000, 10000, 50000, 100000, 500000, 1000000,
↪  2000000, 3000000 ]

print(f"Pontos rtk: {exact_integral:.6f}")

print("Trapz mdszer alkalmazsa:")
for n in num_intervals:
    approx_trapezoid(f_lambd, (a, b), (c, d), n)
```

```
print("Simpson mdszer alkalmazsa:")
for n in num_intervals:
    approx_simpson(f_lambd, (a, b), (c, d), n)

print("Hit and Miss - pszeudovletlen:")
for n in num_points:
    approx, hits = monte_carlo_hit_and_miss(f_lambd, (a, b), (c, d),
    ↪  n)
    for i in range(1, 20):
        approx_new, hits_new = monte_carlo_hit_and_miss(f_lambd, (a,
        ↪  b), (c, d), n)
        if np.abs(exact_integral - approx_new) <
        ↪  np.abs(exact_integral - approx):
            approx = approx_new
            hits = hits_new
    print(f"\tKzelt rtk: {approx:.6f}, n = {n}, nt = {hits}")

print("Hit and Miss - Halton:")
for n in num_points:
    approx, hits = monte_carlo_hit_and_miss_halton(f_lambd, (a, b),
    ↪  (c, d), n)
    for i in range(1, 20):
        approx_new, hits_new =
        ↪  monte_carlo_hit_and_miss_halton(f_lambd, (a, b), (c, d),
        ↪  n)
        if np.abs(exact_integral - approx_new) <
        ↪  np.abs(exact_integral - approx):
            approx = approx_new
            hits = hits_new
    print(f"\tKzelt rtk: {approx:.6f}, n = {n}, nt = {hits}")

print("Sampling - pszeudovletlen:")
for n in num_points:
    approx = monte_carlo_sampling(f_lambd, (a, b), (c, d), n)
    for i in range(1, 20):
        approx_new = monte_carlo_sampling(f_lambd, (a, b), (c, d), n)
        if np.abs(exact_integral - approx_new) <
        ↪  np.abs(exact_integral - approx):
            approx = approx_new
    print(f"\tKzelt rtk: {approx:.6f}, n = {n}")

print("Sampling - Halton:")
for n in num_points:
    approx = monte_carlo_sampling_halton(f_lambd, (a, b), (c, d), n)
    for i in range(1, 20):
        approx_new = monte_carlo_sampling_halton(f_lambd, (a, b),
        ↪  (c, d), n)
        if np.abs(exact_integral - approx_new) <
        ↪  np.abs(exact_integral - approx):
            approx = approx_new
```

```
    print(f"\tKzelt rtk: {approx:.6f}, n = {n}")
```

Резюме

У своїй роботі я займався оцінюванням визначених інтегралів з однією та двома змінними. Для цього я дослідила різні чисельні та методи Монте-Карло як з теоретичної, так і з практичної точки зору. Під час написання роботи я ознайомилась з викладанням інтегралів у середній школі, а також, спираючись на систему викладання математики в українських середніх школах, намагалась зважити, чи можна навчати учнів якомусь із методів або хоча б ознайомити їх з ним.

У першому розділі я ознайомилась з основною ідеєю методу Монте-Карло та його історичним підґрунтям. У другому розділі я розглянула чисельні методи. Я вивчила необхідність їх використання, зокрема детальніше зосередилась на двох з них: методі трапецій та методі Сімпсона. Я вивчила і застосувала їх як для визначених інтегралів з однією, так і з двома змінними.

У третьому розділі я докладно розглянула застосування методів Монте-Карло для оцінювання визначених інтегралів з однією та двома змінними. Я також дослідила, як можна застосовувати ці методи для оцінки площі неправильних фігур. На теоретичному рівні я розглянула аналіз похибок та деякі складніші методи Монте-Карло, як-от важливісне вибіркове дослідження (importance sampling) та контроль варіації (control variates).

У четвертому розділі я виклада основи та методи генерації випадкових чисел, оскільки випадкове вибіркове дослідження є ключовим елементом функціонування методів Монте-Карло. Тут я також порівняла псевдовипадкові точки з точками Галтона (Halton).

У п’ятому розділі я розглянула методи з практичної точки зору. Спочатку я детально описала, як працюють методи для оцінки визначених інтегралів з однією змінною, а потім порівняла їх на прикладі двох задач. Аналогічно, для подвійних інтегралів я спочатку на прикладі продемонструвала детальне застосування, а потім показала ефективність методів ще на двох прикладах. Для демонстрації методів я написала Python-коди, які будують наочні графіки та обчислюють наближені значення для заданих прикладів. Я дійшла висновку, що для одновимірних інтегралів простіше отримати гарну чисельну оцінку, ніж використовувати метод Монте-Карло. У випадку подвійних інтегралів метод Монте-Карло виявився значно ефективнішим, хоча його повний потенціал у деяких ситуаціях ще не розкривається. Те, який метод доцільно використовувати в конкретному випадку, визначається властивостями функції.

У останньому розділі роботи я розглянула, як чисельні методи та методи Монте-Карло можуть бути представлені у середній школі. Я дослідила, які методичні можливості існують для того, щоб зробити ці методи зрозумілими та захопливими для учнів.

Nyilatkozat

Alulírott, Király Éva, 014. Középiskolai oktatás (Matematika) képzési program hallgatója, kijelentem, hogy a dolgozatomat a II. Rákóczi Ferenc Kárpátaljai Magyar Főiskolán, a Matematika és Informatika Tanszéken készítettem, 014. Középiskolai oktatás (Matematika) BSc diploma megszerzése végett.

Kijelentem, hogy a dolgozatot más szakon korábban nem védtem meg, saját munkám eredménye, és csak a hivatkozott forrásokat (szakirodalom, eszközök stb.) használtam fel.

Tudomásul veszem, hogy dolgozatomat a II. Rákóczi Ferenc Kárpátaljai Magyar Főiskola könyvtárában a kölcsönözhető könyvek között helyezik el.

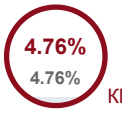
Звіт подібності

метадані

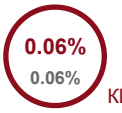
Назва організації
Hungarian College of Higher Education Ferenc Rakoczi II Transcarpathian
Заголовок
evfolyammunk_Király_Éva
Науковий керівник / Експерт
Автор**Габріелла Пап**
підрозділ
Закарпатський угорський інститут імені Ференца Ракоці II

Обсяг знайдених подібностей

Коефіцієнт подібності визначає, який відсоток тексту по відношенню до загального обсягу тексту було знайдено в різних джерелах. Зверніть увагу, що високі значення коефіцієнта не автоматично означають плагіат. Звіт має аналізувати компетентна / уповноважена особа.



КП 1



КЦ

25	14770	98151
Довжина фрази для коефіцієнта подібності 2	Кількість слів	Кількість символів

Тривога

У цьому розділі ви знайдете інформацію щодо текстових спотворень. Ці спотворення в тексті можуть говорити про МОЖЛИВІ маніпуляції в тексті. Спотворення в тексті можуть мати навмисний характер, але частіше характер технічних помилок при конвертації документа та його збереженні, тому ми рекомендуємо вам підходити до аналізу цього модуля відповідально. У разі виникнення запитань, просимо звертатися до нашої служби підтримки.

Заміна букв		7
Інтервали	A→	0
Мікропробіли	·	0
Білі знаки		0
Парафрази (SmartMarks)	a	40

Подібності за списком джерел

Нижче наведений список джерел. В цьому списку є джерела із різних баз даних. Колір тексту означає в якому джерелі він був знайдений. Ці джерела і значення Коефіцієнту Подібності не відображають прямого плагіату. Необхідно відкрити кожне джерело і проаналізувати зміст і правильність оформлення джерела.

10 найдовших фраз — Колір тексту

ПОРЯДКОВИЙ НОМЕР	НАЗВА ТА АДРЕСА ДЖЕРЕЛА URL (НАЗВА БАЗИ)	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	https://fac.ksu.edu.sa/sites/default/files/numerical_analysis_9th.pdf	38 0.26 %
2	https://fac.ksu.edu.sa/sites/default/files/numerical_analysis_9th.pdf	38 0.26 %
3	https://fac.ksu.edu.sa/sites/default/files/numerical_analysis_9th.pdf	20 0.14 %
4	https://fac.ksu.edu.sa/sites/default/files/numerical_analysis_9th.pdf	20 0.14 %
5	https://studfile.net/preview/5471038/page:4/	18 0.12 %

6	https://fac.ksu.edu.sa/sites/default/files/numerical_analysis_9th.pdf	18 0.12 %
7	https://fac.ksu.edu.sa/sites/default/files/numerical_analysis_9th.pdf	17 0.12 %
8	https://fac.ksu.edu.sa/sites/default/files/numerical_analysis_9th.pdf	16 0.11 %
9	https://studfile.net/preview/5471038/page:4/	14 0.09 %
10	http://rgmia.org/papers/monographs/Master.pdf	13 0.09 %

з домашньої бази даних (0.00 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)

з програми обміну базами даних (0.00 %)

ПОРЯДКОВИЙ НОМЕР	ЗАГОЛОВОК	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)

з Інтернету (4.76 %)

ПОРЯДКОВИЙ НОМЕР	ДЖЕРЕЛО URL	КІЛЬКІСТЬ ІДЕНТИЧНИХ СЛІВ (ФРАГМЕНТІВ)
1	**https://fac.ksu.edu.sa/sites/default/files/numerical_analysis_9th.pdf**	**405 (38) 2.74 %**
2	http://rgmia.org/papers/monographs/Master.pdf	63 (8) 0.43 %
3	https://studfile.net/preview/5471038/page:4/	45 (4) 0.30 %
4	https://epdf.tips/sources-in-the-development-of-mathematics-series-and-products-from-the-fifteenth19276.html	41 (6) 0.28 %
5	http://issc.uj.ac.za/downloads/problems/SCandPP.pdf	20 (2) 0.14 %
6	https://www.cs.purdue.edu/homes/enh/courses/cs158a/cs158ap1/c9.pdf	17 (2) 0.12 %
7	https://en.wikipedia.org/wiki/Line%E2%80%93line_intersection	16 (2) 0.11 %
8	http://people.math.umass.edu/~kevrekid/132_f10/132class3.pdf	15 (2) 0.10 %
9	http://www.fizyka.umk.pl/~fifi/Materiay_MMF_files/taylor_dywer.pdf	13 (2) 0.09 %
10	https://znayshov.com/FR/34017/metod_Chern_2024_2025-180-186.pdf	12 (1) 0.08 %
11	https://ar5iv.labs.arxiv.org/html/0905.1886	12 (1) 0.08 %
12	https://www.slideshare.net/duytay94/vnmathcom-13kithuatgiaiphuongtrinhham	11 (2) 0.07 %
13	http://value-at-risk.net/references/	11 (1) 0.07 %
14	https://stud.kz/referat/show/95028	6 (1) 0.04 %
15	https://studyx.ai/homework/106785946-4-diketahui-f-x-1-x-dan-g-x-x-3-tentukan	6 (1) 0.04 %
16	https://sala.moeys.gov.kh/kh/library/00002601	5 (1) 0.03 %
17	https://people.gnome.org/~sragavan/osc.log	5 (1) 0.03 %

Список прийнятих фрагментів (немає прийнятих фрагментів)

ПОРЯДКОВИЙ НОМЕР	ЗМІСТ	КІЛЬКІСТЬ ОДНАКОВИХ СЛІВ (ФРАГМЕНТІВ)