Euler-Maruyama metoda za rješavanje stohastičkih diferencijalnih jednadžbi uz implementaciju u Pythonu

Markulak, Dunja

Sveučilište Josipa Jurja Strossmayera u Osijeku
Fakultet primijenjene matematike i informatike

Sveučilišni diplomski studij matematike
smjer: Matematika i računarstvo

Dunja Markulak

Euler-Maruyama metoda za rješavanje stohastičkih diferencijalnih jednadžbi uz implementaciju u Pythonu

Diplomski rad

Mentor: Nenad Šuvak

Komentor: Mateja Đumić

Osijek, 2023.

Sadržaj

1 Uvod

Stohastične diferencijalne jednadžbe (SDJ) igraju ključnu ulogu u modeliranju složenih sustava obilježenih nesigurnošću, slučajnostima i dinamikom kontinuiranog vremena. Ove jednadžbe nalaze primjenu u raznim područjima, kao što su financije, fizika, biologija i inženjerstvo, gdje je razumijevanje i kvantificiranje utjecaja stohastičkih procesa ključno za donošenje informiranih odluka i predviđanja. Jedan od temeljnih izazova u radu sa SDJ je pronalaženje učinkovitih i preciznih numeričkih metoda za njihovo rješavanje.

Među dostupnim numeričkim tehnikama, Eulerova metoda, poznatija i kao Euler-Maruyama metoda, stajala je kao praktičan i dostupan alat za aproksimaciju rješenja SDJ. Eulerova metoda, izvorno razvijena za rješavanje običnih diferencijalnih jednadžbi (ODJ), prilagođena je i proširena kako bi se nosila sa stohastičkim karakterom prisutnim u SDJ. Ova metoda pruža jednostavan i intuitivan pristup za simulaciju trajektorija stohastičkih procesa, čineći je vrijednim sredstvom za istraživače i praktičare.

Ključna ideja iza Eulerove metode za SDJ je diskretizacija vremena na male intervale i ažuriranje stanja sustava koristeći jednostavan sustav aproksimacije u svakom koraku vremena. Iako Eulerova metoda možda djeluje jednostavno u usporedbi s naprednijim numeričkim tehnikama, njezina učinkovitost leži u toj jednostavnosti i svestranosti. Ona služi kao temeljni alat za razumijevanje složenijih algoritama i pruža važne uvide u ponašanje stohastičkih sustava.

Struktura ovog rada organizirana je kako slijedi: U poglavlju 1 predstavljena je Itôva SDJ te matematički izrazi za razumijevanje iste. Definirana je difuzija kao rješenje SDJ, dani su primjeri rješavanja SDJ te je dana motivacija za primjenu numeričkih metoda. Poglavlje 2 sadrži izvod Euler-Maruyama metode, uključujući temeljne principe i algoritam. Također su dani praktični primjeri i detalji implementacije uz dijelove programskog koda u Pythonu. U 3. poglavlju napravljena je pseudo-likelihood procjena parametara metode. Zatim odjeljak 4 daje prikaz i informacije o izvedbi grafičkog sučelja u Pythonu za simulaciju trajektorija SDJ dobivenih Euler-Maruyama metodom.

2 Itôva stohastička diferencijalna jednadžba

U ovom poglavlju navest ćemo najvažnije pojmove i teorijske rezultate vezane za Itôve stohastičke diferencijalne jednadžbe (SDJ).

Prvo ćemo definirati Brownovo gibanje, pogonski proces u SDJ. Definicija Brownovog gibanja glasi:

Definicija 2.1. Slučajni proces $(B_t, t \geq 0)$ na vjerojatnosnom prostoru (Ω, F, P) je standardno Brownovo gibanje ako vrijedi:

1. $\forall \omega \in \Omega$ trajektorija $t \mapsto B_t(\omega)$ je gotovo sigurno neprekidna funkcija s $[0, \infty)$ u $\mathbb{R}$,
2. $B_0 = 0$,
3. za sve $0 = t_0 < t_1 < ... < t_m$ prirasti,

$$B_{t_1} = B_{t_1} - B_{t_0}, B_{t_2} - B_{t_1}, ...B_{t_m} - B_{t_{m-1}}$$

   su nezavisne slučajne varijable

4. za sve $0 \leq s < t$ je prirast $(B_t - B_s)$ normalno distribuiran s očekivanjem nula i varijancom $(t - s)$.

Nadalje, neka je $C = (C_t, 0 \leq t \leq T)$ $\mathbb{F}^B$-adaptiran slučajni proces na vjerojatnosnom prostoru (Ω, F, P), s prirodnom filtracijom $\mathbb{F}^B = (F_t^B, t \geq 0)$, gdje je $F_t^B = \sigma(B_s, 0 \leq s \leq t)$, za koji vrijedi

$$\int_0^T E(C_t^2)\, dt < \infty, \tag{1}$$

što podrazumjeva i kvadratnu integrabilnost procesa C.

Osnovna ideja proširenja Itôve integracije jest aproksimacija općeg integranda nizom jednostavnih slučajnih procesa koji u određenom smislu konvergira prema slučajnom procesu čiju trajektoriju želimo integrirati u odnosu na trajektoriju Brownovog gibanja $(B_t, t \geq 0)$. Odnosno, za slučajni proces $C = (C_t, 0 \leq t \leq T)$ koji zadovoljava uvijet (1) postoji niz jednostavnih slučajnih procesa $(C^{(n)}, n \in \mathbb{N})$, $C^{(n)} = (C_t^{(n)}, 0 \leq t \leq T)$, takvih da je

$$\int_0^T E\left((C_t - C_t^{(n)})^2\right) dt \to 0, \quad n \to \infty.$$

Dakle, niz slučajnih procesa $(C^{(n)}, n \in \mathbb{N})$ u srednje kvadratnom smislu konvergira prema procesu $C = (C_t, 0 \leq t \leq T)$. S obzirom da je niz Itôvih

integrala jednostavnih procesa Cauchyjev niz u prostoru $L^2(\Omega, F, P)$, on ima limes, a taj limes nazivamo Itôv stohastički integral i definiramo ga na sljedeći način:

$$I_t = \lim_{n\to\infty} \int_0^t C_s{}^{(n)}\, dB_s = \int_0^t C_s\, dB_s \quad 0 \le t \le T.$$

Definicija i svojstva Itôvog stohastičkog integrala mogu se naći u [9].

2.1 Difuzija - rješenje SDJ

Markovljev proces u neprekidnom vremenu s neprebrojivim skupom stanja i g.s. neprekidnim trajektorijama nazivamo difuzija. Kako bismo definirali difuziju prvo ćemo se upoznati s konceptom Markovljevog svojstva.

Definicija 2.2. Markovljev proces u neprekidnom vremenu je slučajni proces $(X_t, t \ge 0)$ definiran na vjerojatnosnom prostoru $(\Omega, \mathcal{F}, P)$ za koji vrijedi sljedeće svojstvo:

$$P(X_{t_{n+1}} \in F | X_{t_1} = x_1, X_{t_2} = x_2, ..., X_{t_n} = x_n) = P(X_{t_{n+1}} \in F | X_{t_n} = x_n) \tag{2}$$

$\forall F \in \mathcal{F}$ za proizvoljne vremenske trenutke $0 < t_1 < t_2 < ... < t_n < t_{n+1}$ i za proizvoljna stanja $x_1, x_2, ...x_n \in \mathbb{R}$.
Svojstvo (2) nazivamo Markovljevo svojstvo, koje kaže da pozicija (stanje) procesa u budućem trenutku t ovisi samo o poziciji procesa u sadašnjosti, tj. da su, uvjetno na poznatu sadašnjost, budućnost i prošlost procesa nezavisne.

Definicija 2.3. Prijelazna vjerojatnost Markovljevog procesa definirana je na sljedeći način:

$$P(F, t; y, s) = P(X_t \in F | X_s = y), \quad 0 \le s < t.$$

Definicija 2.4. Difuzijski proces ili difuzija jest Markovljev proces $X = (X_t, t \ge 0)$ na vjerojatnosnom prostoru (Ω, F, P) u neprekidnom vremenu s neprebrojivim skupom stanja S za koji vrijedi:

$$\lim_{h\to\ 0} \frac{1}{h} P(|X_{t+h} - x| > \varepsilon | X_t = x] = 0, \forall \varepsilon > 0, \forall x \in S \tag{3}$$

i za kojeg vrijedi da su:

$$\lim_{n\to\infty} \frac{1}{h} E[\Delta X_t(h) | X_t = x] = \mu(x, t), \tag{4}$$

$$\lim_{n\to\infty} \frac{1}{h} E[(\Delta X_t(h))^2 | X_t = x] = \sigma^2(x,t) \tag{5}$$

neprekidne funkcije obiju nezavisnih varijabli, gdje je $\Delta X_t(h) = X_{t+h} - X_t$ i svi potrebni momenti postoje.

Parametre $\mu(x,t)$ i $\sigma(x,t)$ zajedničkim imenom nazivamo infinitezimalni parametri difuzije, a zasebno μ nazivamo drift parametrom, a σ parametrom difuzije. Difuzija je regularan proces ako je uvjetno na start difuzije iz stanja $x \in S$, vrijeme prvog dolaska difuzije u stanje $s \in S$

$$T_s = \inf\{t \geq 0 : X_s = s\},$$

konačno s pozitivnom vjerojatnošću, tj.

$$P(T_s < \infty | X_0 = x) > 0.$$

Difuzija je rješenje SDJ, a opći oblik Itôve SDJ je:

$$dX_t = \mu(t, X_t)dt + \sigma(t, X_t)dB_t, \quad 0 \leq t \leq T, \tag{6}$$

gdje je $B = (B_t, 0 \leq t \leq T)$ Brownovo gibanje koje nazivamo pogonski proces.
Svojstvo (3) interpretiramo na sljedeći način: veliki prirasti su malo vjerojatni na dovoljno malim intervalima. Ono što razlikuje ovakvu jednadžbu od obične diferencijalne jednadžbe (ODJ) jest stohastički element koji dolazi od početnog uvjeta X_0, koji je dan u terminima slučajne varijable nezavisne od pogonskog procesa, te šuma generiranog Brownovim gibanjem B. SDJ (6) opisuje dinamiku difuzije X koju *naivno* možemo opisati na sljedeći način: na intervalu duljine dt prirast difuzije X ($dX_t = X_{t+dt} - X_t$) uzrokovan je prirastom vremena dt s koeficijentom $\sigma(t, X_t)$ i prirastom Brownovog gibanja $dB_t = B_{t+dt} - B_t$ s koeficijentom $\sigma(t, X_t)$.
S obzirom na to da trajektorije Brownovog gibanja nisu diferencijabilne, prethodno navedenu jednadžbu (6) trebamo razmatrati kao stohastičku integralnu jednadžbu:

$$X_t = X_0 + \int_0^t \mu(s, X_s)\, ds + \int_0^t \sigma(s, X_s)\, dB_s, \quad 0 \leq t \leq T, \tag{7}$$

gdje je prvi integral Riemannov, a drugi Itôv stohastički integral.
Nadalje, u nastavku ćemo navesti dva tipa rješenja, jako i slabo rješenje SDJ te uvjete za egzistenciju i jedinstvenost jakog rješenja.

Definicija 2.5. Jako rješenje SDJ (7) je slučajni proces $X = (X_t, 0 \leq t \leq T)$ koji zadovoljava zahtjeve:

1. integrali u (7) su redom dobro definirani Riemannov i Itôv stohastički integral,

2. X je adaptiran na prirodnu filtraciju Brownovog gibanja, tj. u trenutku t ovisi samo o slučajnim varijablama $B_s, 0 \leq s \leq t$,

3. X je transformacija pogonskog procesa, odnosno Brownovog gibanja B i funkcijskih koeficijenata $\mu(t,x)$ i $\sigma(t,x)$ koje su neprekidne funkcije.

Definicija 2.6. Slabo rješenje SDJ daje karakterizaciju difuzije u kontekstu distribucijskih svojstava i dovoljno je za određivanje funkcije očekivanja, funkcije autokovarijanci i autokorelacijske funkcijske procesa, pa trajektorije Brownovog gibanja nisu ključne.

S obzirom na jače uvjete u jakom rješenju SDJ da se pretpostaviti da neke SDJ nemaju jako rješenje već samo slabo rješenje.

Što se tiče uvjeta o egzistenciji i jedinstvenosti rješenja, dovoljni uvjeti su:

1. X_0 je nezavisna od Brownovog gibanja B i kvadratno je integrabilna ($E[X_0^2] < \infty$),

2. funkcije $\mu(t,x)$ i $\sigma(t,x)$ su neprekidne te za sve $t \in [0,T]$ i sve $x \in \mathbb{R}$

   - zadovoljavaju Lipschitzov uvjet po prostornoj varijabli, tj. $\exists k_1 > 0$ td.

   $$|\mu(t,x) - \mu(t,y)| + |\sigma(t,x) - \sigma(t,y)| \leq k_1|x-y| \tag{8}$$

   - zadovoljavaju uvjet linearnog rasta, tj. $\exists k_2 > 0$ td.

   $$\mu^2(t,x) + \sigma^2(t,x) \leq k_2^2(1+x^2). \tag{9}$$

Navedeni dovoljni uvjeti mogu se ublažiti, tj. Lipschitzov uvjet možemo relaksirati na lokalni Lipschitzov uvjet

$$|\mu(t,x) - \mu(t,y)| + |\sigma(t,x) - \sigma(t,y)| \leq k_N|x-y|,$$
$$\forall |x|, |y| \leq N \ , \ k_N > 0, \ N > 0.$$

Za (9) je važno znati da svaka neprekidno derivabilna funkcija zadovoljava lokalni Lipschitzov uvjet. Dodatno možemo ublažiti dovoljne uvjete ako bismo zanemarili uvjet kvadratne integrabilnosti slučajne varijable X_0 jer on sam po sebi ne daje egzistenciju ni jedinstvenost rješenja, ali bez njega je moguća eksplozija trajektorija rješenja X, to znači da proces X u konačnom

vremenu beskonačno mnogo puta promijeni stanje. U nastavku rada ćemo pretpostaviti da radimo sa SDJ koje imaju jako rješenje.

Kroz ovaj rad kao primjere koristit ćemo dva procesa: *geometrijsko Brownovo gibanje* i *Ornstein-Uhlenbeckov proces.*

Geometrijsko Brownovo gibanje (GBG) jedan je od često korištenih modela za opisivanje kretanja cijena dionica. SDJ za GBG je:

$$dX_t = \mu X_t dt + \sigma X_t dB_t, X_0 = x_0 > 0. \tag{10}$$

Egzaktno rješenje SDJ za GBG s trajektorijom prikazanom na Slici 1 dano je izrazom:

$$X_t = X_0 e^{(\mu - \sigma^2/2)t + \sigma B_t}, 0 \leq t \leq T. \tag{11}$$

Drugi spomenuti proces, Ornstein-Uhlenbeckov proces (OU proces) je rješenje Langevineove stohastičke diferencijalne jednadžbe:

$$dX_t = -\theta(X_t - \mu)dt + \sqrt{2\theta\sigma^2}dB_t \quad \theta > 0, \quad \mu \in \mathbb{R}, \quad \sigma > 0. \tag{12}$$

Egzaktno rješenje SDJ (12) je oblika:

$$X_t = \mu + e^{-\theta t}(X_0 - \mu) + \sigma\sqrt{2\theta}e^{-\theta t}\int_0^t e^{\theta s}\, dBs, \tag{13}$$

te vrijedi $X_t \sim \mathcal{N}(e^{ct}x, \sigma^2/2c(e^{2ct} - 1))$, uvjetno na $X_0 = x$. Generalnija parametrizacija SDJ (12) koji ćemo također koristiti u radu jest:

$$dX_t = (\theta_1 - \theta_2 X_t)dt + \theta_3 dB_t. \tag{14}$$

Za ovaj proces dan SDJ (14) vrijedi da je stacionaran ako je $X_0 \sim \mathcal{N}(\frac{-\theta_2}{\theta_1}, \theta_3)$.

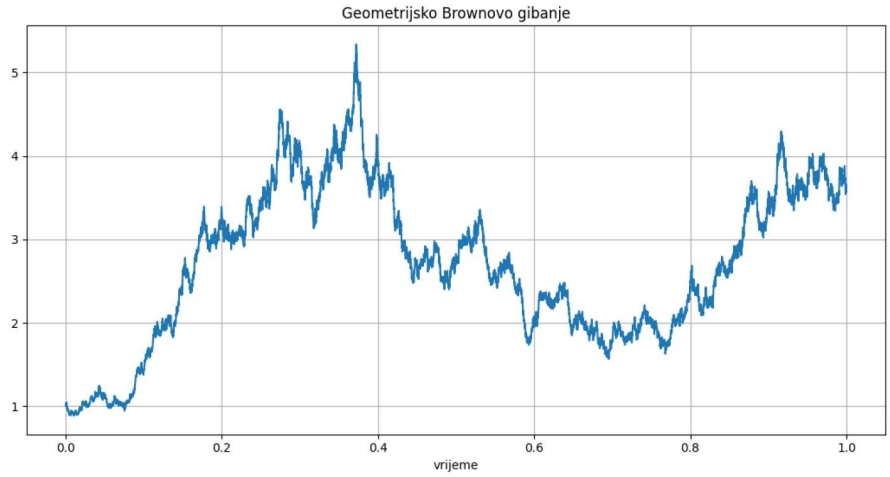
Primjer 1. *Provjerimo egzistenciju i jedinstvenost jakog rješenja SDJ za geometrijsko Brownovo gibanje* (10).

Dovoljan uvjet koji smo naveli je Lipschitzovost po prostornoj varijabli, za $s, t \geq 0$ imamo:

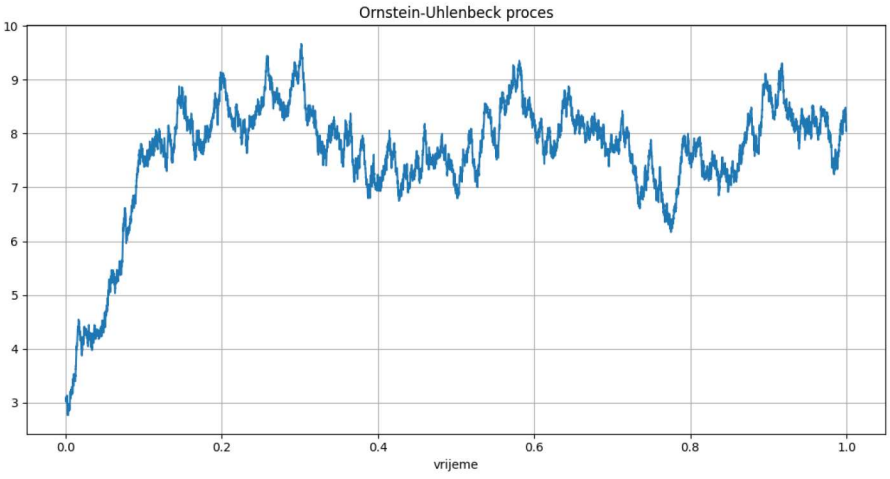
$$|\mu X_s - \mu X_t| + |\sigma X_s - \sigma X_t| \leq \max\{\mu, \sigma\}|X_s - X_t|,$$

gdje je očito $k_1 = \max\{\mu, \sigma\}$.
Kako su drift i difuzija linearne funkcije prostorne varijable, uz prethodno

pokazanu zadovoljenost Lipschitzovog uvjeta, slijedi da SDJ za GBG ima jedinstveno jako rješenje.



Slika 1: Trajektorija geometrijskog Brownovog gibanja s početkom u $X_0 = 1$ i parametrima $\mu = 2$, $\sigma = 1$



Slika 2: Trajektorija Ornstein-Uhlenbeckovog procesa s početkom u $X_0 = 10$ i parametrima $\mu = 10$, $\theta = 5$, $\sigma = -5$

2.2 Rješavanje stohastičkih diferencijalnih jednadžbi

U ovom potpoglavlju navest ćemo nekoliko rezultata koji se koriste za rješavanje SDJ. Za rješavanje prvog primjera potrebno nam je prvo definirati Itôv proces i odgovarajuću Itôvu formulu.

Definicija 2.7. Za slučajni proces $(X_t, t \geq 0)$ kažemo da je Itôv proces ako vrijedi

$$X_t = X_0 + \int_0^t A_s^{(1)}\, ds + \int_0^t A_s^{(2)}\, dB_s,$$

gdje su $(A_t^{(1)}, t \geq 0)$ i $(A_t^{(2)}, t \geq 0)$ adaptirani u odnosu na prirodnu filtraciju Brownovog gibanja $(B_t, t \geq 0)$ te su navedeni integrali dobro definirani.

Lema 2.1. *(Itôva formula II za Itôv proces) Neka je $(X_t, t \geq 0)$ Itôv proces i neka je $f\colon [0,\infty\rangle \times S \to \mathbb{R}$, gdje je S skup vrijednosti slučajnog procesa $(X_t,\ t \geq 0)$, funkcija dviju varijabli s neprekidnim derivacijama drugog reda. Tada vrijedi:*

$$f(t, X_t) - f(s, X_s) = \int_s^t \left(f_1(y, X_y) + A_y^{(1)} f_2(y, X_y) + \frac{1}{2}(A_y^{(2)})^2 f_{22}(y, X_y)\, dy \right) + \\ + \int_s^t A_y^{(2)} f_2(y, X_y)\, dB_y, \quad s < t. \quad (15)$$

SDJ možemo riješiti na nekoliko načina koje ćemo vidjeti u sljedećim primjerima.

Primjer 2. *Riješimo SDJ za GBG oblika*

$$dX_t = \mu X_t dt + \sigma X_t dB_t, \quad \mu \in \mathbb{R}, \quad \sigma > 0, \quad E[X_0^2] < \infty.$$

Za početak pretvorimo ovu diferencijalnu jednadžbu u integralnu, sada imamo:

$$X_t = X_0 + \mu \int_0^t X_s\, ds + \sigma \int_0^t X_s\, dB_s.$$

Primjenom Itôve leme za ovu SDJ i funkciju $X_t = f(t, B_t)$ slijedi:

$$X_t - X_0 = \int_0^t (f_1(s, B_s) + \frac{1}{2} f_{22}(s, B_s))\, ds + \int_0^t f_2(s, dB_s)\, dB_s.$$

Kako bismo došli do rješenja cilj nam je odrediti funkciju f. Usporedbom integralne jednadžbe koju rješavamo i jednadžbe izvedene iz Itôve leme za Itôve procese, slijedi:

$$\mu f(t,x) = f_1(t,x) + \tfrac{1}{2} f_{22}(t,x) \quad \text{i} \quad \sigma f(t,x) = f_2(t,x).$$

Ovim smo dobili sustav parcijalnih diferencijalnih jednadžbi koje možemo riješiti.
Znamo $\sigma f_2(t,x) = f_{22}(t,x)$ tj $\sigma^2 f(t,x) = f_{22}(t,x)$ pa početni sustav diferencijalnih jednadžbi postaje:

$$\left(\mu - \frac{1}{2}\sigma^2\right) f(t,x) = f_1(t,x), \quad \sigma f(t,x) = f_2(t,x).$$

Za supstituciju $f(t,x) = h(t)g(x)$ dani sustav postaje:

$$\left(\mu - \frac{1}{2}\sigma^2\right) h(t) = h'(t), \quad \sigma g(x) = g'(x),$$

koji možemo riješiti metodom separacije. Rješenja dobivena metodom separacije su:

$$h(t) = h(0)e^{(\mu - \frac{\sigma^2}{2})t}, \quad g(x) = g(0)e^{\sigma x}.$$

Primijetimo da je $X_0 = f(0, B_0) = f(0,0)$, pa je konačno rješenje je

$$X_t = f(t, B_t) = X_0 e^{(\mu - \frac{\sigma^2}{2})t + \sigma B_t}.$$

Također znamo da je dobiveno rješenje jedinstveno i jako rješenje jer pripadni infinitezimalni parametri, $\mu(t,x) = \mu x$, $\sigma(t,x) = \sigma x$, zadovoljavaju u Lipschitzov uvjet te uvjet linearnog rasta, a početno stanje difuzije je kvadratno integrabilno.

Za razliku od prethodne, postoje SDJ koje nije moguće riješiti eksplicitnim metodama kao npr. SDJ sa skokovima (*Merton jump diffusion model*).

Također, rješavanje stohastičkih integrala analitički može biti vrlo kompleksno te rješavanje SDJ računalno na eksplicitan način može biti dug proces. Zbog svih prijašnjih razloga u praksi se najčešće koriste numeričke metode za rješavanje SDJ.
Korištenje numeričkih metoda omogućava nam vizualizaciju trajektorija difuzije, odnosno uvid u buduće kretanje procesa u praksi. Zatim omogućava nam procjenu distribucijskih karakteristika difuzije kao što su očekivanje, autokovarijanca, itd. Numeričke metode, poput Euler-Maruyama metode na koju ćemo se i bazirati u ovom radu, aproksimiraju rješenje SDJ što optimizira cijeli proces, a također daje i rješenje SDJ koje nemaju eksplicitno rješenje.

3 Euler-Maruyama metoda za rješavanje SDJ

Euler-Maruyama metoda je jedna od najpoznatijih metoda za numeričko rješavanje stohastičkih diferencijalnih jednadžbi, nastala prilagodbom Eulerove metode za rješavanje običnih diferencijalnih jednadžbi. Osnovna ideja ove metode za SDJ je aproksimacija rješenja SDJ kroz jednostavne iteracije te promatranje kontinuiranog vremena kao konačne familije disjunktnih intervala u čijim se rubovima računa numeričko rješenje SDJ. Počinjemo s početnim uvjetima i definicijom vremenskog koraka te informacije iz prethodne iteracije koristimo za izračunavanje nove vrijednosti aproksimacije stohastičkog procesa koji rješava promatranu SDJ.
Dakle, neka je $\pi_n = \pi_n([0,T])$ subdivizija intervala $[0,T]$, takva da $0 < t_0 < t_1 < ... < t_n = T$. Za subdiviziju najčešće uzimamo ekvidistantnu subdiviziju danog segmenta [0,T]:

$$\pi_n = \left\{ \frac{iT}{n} : i = 0, ..., n \right\}.$$

Sada uvedimo pojam *numeričko rješenje*.
Numeričko rješenje SDJ je slučajni proces koji aproksimira difuziju. Odnosno, slučajni proces $X^{(n)} = (X_t^{(n)}, 0 \leq t \leq T)$ je numeričko rješenje SDJ (6) koje se temelji na subdiviziji π_n segmenta $[0,T]$:

$$\pi_n = \{t_0, \ldots, t_n\}, 0 = t_0 < t_1 < \ldots < t_n = T$$
$$||\pi_n|| = \max_{i \in \{0,\ldots,n\}} \{t_i - t_{i-1}\} = \max_i \Delta_i.$$

Numeričko rješenje $X^{(n)}$ računa se samo u točkama subdivizije $\pi_n \subset [0,T]$, a pomoću vrijednosti $(X_{t_i}^{(n)}, t_i \in \pi)$ numeričkog rješenja u točkama subdivizije aproksimiramo difuziju na intervalima $\langle t_{i-1}, t_i \rangle$ linearnom interpolacijom točaka $(t_{i-1}, X_{t_{i-1}}^{(n)})$ i $(t_i, X_{t_i}^{(n)})$.
Nadalje, neka je $(X_t, 0 \leq t \leq T)$ rješenje SDJ oblika

$$dX_t = \mu(t, X_t)dt + \sigma(t, X_t)dB_t,$$

uz početnu vrijednost $X_{t_0} = X_0$.

Sada u trenutku t_i promatramo sljedeću integralnu jednadžbu:

$$X_{t_i} = X_{t_{i-1}} + \int_{t_{i-1}}^{t_i} \mu(X_s)\, ds + \int_{t_{i-1}}^{t_i} \sigma(X_s)\, dB_s. \tag{16}$$

Eulerova aproksimacija temelji se na aproksimaciji integrala u (16), gdje Riemannov integral aproksimiramo na sljedeći način:

$$\int_{t_{i-1}}^{t_i} \mu(X_s)\, ds \approx \mu(X_{t_{i-1}})(t_i - t_{i-1}), \tag{17}$$

a Itôv integral na sljedeći način:

$$\int_{t_{i-1}}^{t_i} \sigma(X_s)\, dB_s \approx \sigma(X_{t_{i-1}})(B_{t_i} - B_{t_{i-1}}). \tag{18}$$

Euler-Maruyama aproksimacija konstruirana je na sljedeći način:

$$\begin{aligned} X_0^{(n)} &= X_0 \\ X_{t_1}^{(n)} &= X_0^{(n)} + \mu(X_0^{(n)})\Delta_1 + \sigma(X_0^{(n)})\Delta_1 B \\ X_{t_2}^{(n)} &= X_{t_1}^{(n)} + \mu(X_{t_1}^{(n)})\Delta_2 + \sigma(X_{t_1}^{(n)})\Delta_2 B \\ &\vdots \\ X_{t_{n-1}}^{(n)} &= X_{t_{n-2}}^{(n)} + \mu(X_{t_{n-2}}^{(n)})\Delta_{n-1} + \sigma(X_{t_{n-2}}^{(n)})\Delta_{n-1} B \\ X_T^{(n)} &= X_{t_{n-1}}^{(n)} + \mu(X_{t_{n-1}}^{(n)})\Delta_n + \sigma(X_{t_{n-1}}^{(n)})\Delta_n B, \end{aligned} \tag{19}$$

odnosno

$$X_{t_{i+1}} = X_{t_i} + \mu(t_i, X_{t_i})\Delta_i + \sigma(t_i, X_{t_i})\Delta_i B, \tag{20}$$

gdje je $\Delta_i = t_i - t_{i-1}$, $\Delta_i B = B_{t_i} - B_{t_i - 1}$, $i = 0, 1, .., N-1$ te X_0 početna vrijednost.
Kako bismo ustanovili kvalitetu aproksimacije koju nam daje Eulerova shema gledamo podudarnost trajektorija procesa, tj. želimo da za fiksni $\omega \in \Omega$ trajektorija $t \mapsto X_t^{(n)}(\omega)$ dobro aproksimira trajektoriju $t \mapsto X_t(\omega)$ za danu trajektoriju Brawnovog gibanja. Očekivanje apsolutne pogreške numeričkog i egzaktnog rješenja u točki T nam daje mjeru tražene kvalitete tj. promatramo

$$E[|X_T(\omega) - X_T^{(n)}(\omega)|], \tag{21}$$

što nam daje informaciju o podudarnosti trajektorija u trenutku T. Druga mjera kvalitete, koja ne uzima u obzir samo krajnji trenutak T, dana je sljedećim izrazom:

$$\sup_{0 \le t \le T} E[|X_t(\omega) - X_t^{(n)}(\omega)|]. \tag{22}$$

3.1 Algoritam Euler-Maruyama aproksimacijske sheme

U ovom potpoglavlju ćemo zapisati i pojasniti pseudokod za Euler-Maruyama metodu te taj pseudokod implementirati u Pythonu (blok 1 programskog koda). Najvažniji dio algoritma nam je iterativni postupak u kojem direktno primjenjujemo (20).

U nastavku rada ćemo za sve prikaze koristiti implementaciju Euler-Maruyama aproksimacije u Pythonu koji smo prilagodili za rješavanje SDJ za geometrijsko Brownovo gibanje i Ornstein-Uhlenbeckov proces.

Algorithm 1 Euler-Maruyama algoritam

Input: X_0 **- početni uvjet,** N **- broj točaka, T - desni rub promatranog intervala**

Output: X

$dt \leftarrow duljina$ subdivizije τ segmenta $[0, T]$
$timesList \leftarrow$ ekvidistantan niz od 0 do T s N točaka
$dB \leftarrow$ Brownovo gibanje
$i \leftarrow 0$
$XAprox \leftarrow []$
$X \leftarrow X0$
while $i < N$ **do**
    $X \leftarrow X + \mu(t, Y) * dt + \sigma(t, Y) * dB$
    $XAprox[i] \leftarrow X$
    $i \leftarrow i + 1$
end while

3.1.1 Python implementacija

Python je dinamički programski jezik koji je postao ključan alat u mnogim inženjerskim i znanstvenim granama. Osim svoje jednostavnosti i čitljivosti koda, velika prednost Pythona su i bogate biblioteke od kojih su neke *matplotlib* i *numpy*, korištene u ovom radu.
Prvotno definiramo klasu SDJ s parametrima potrebnima za definiranje nama potrebnih primjera. Odlučili smo se za klasu kako bi se kod mogao lakše prilagoditi drugim oblicima SDJ, ako bi bilo potrebe. Dodali smo i tri metode. Prva metoda *Model* koja nam vraća $\mu(t, X_t), \sigma(t, X_t)$ funkcije iz (6) za trenutnu SDJ. Druga metoda daje egzaktno rješenje SDJ, ako ono postoji. Treća metoda daje aproksimacijsko rješenje SDJ koristeći iterativni proces Euler-Maruyama metode. Python kod vidimo u sljedeća dva bloka.

```
import numpy as np
import matplotlib.pyplot as plt
import statsmodels.api as sm

np.random.seed(42)

class SDJ:
    def __init__(self, X0, mu, sigma,theta,sdj):
        self.X0 = X0
        self.mu = mu
        self.sigma = sigma
        self.theta = theta
```

```
13          self.name = sdj
```

Listing 1: Dio Python koda u kojem se postavljaju parametri klase te se učitavaju korišteni paketi i postavlja se seed

```
1     def Model(self,y):
2         # elementi za SDJ Geometrijskog Brownovog gibanja
3         if self.name=="GBG":
4             return (self.mu * y, self.sigma * y)
5         # elementi za SDJ Ornstein Uhlenbeck procesa
6         elif self.name == "OU":
7             return(self.theta * (self.mu - y), self.sigma)
8 
9     # metoda za egzaktno rjesavanje SDJ Geometrijskog
    Brownovog gibanja
10    def Exact(self,t,dB):
11        if self.name == "GBG":
12            B = np.cumsum(dB)
13            return (self.X0 * np.exp((self.mu - 0.5 * self.
    sigma**2) * t + B * self.sigma))
14        else:
15            pass
16 
17    # Iterativni postupak Euler Maruyama metode
18    def EM(self,N,dt,dB):
19        XEm,X = [],self.X0
20        for i in range (N):
21            a, b = self.Model(X)
22            X += a*dt + b*dB[i]
23            XEm.append(X)
24        return XEm
```

Listing 2: Dio Python koda u kojem su definirane metode klase SDJ

Nadalje, potrebno je definirati sljedeće funkcije:

1. BG - simulira Brownovo gibanje uz pomoć biblioteke *numpy*,
2. plotExEm - služi za prikaz simulirane trajektorije.

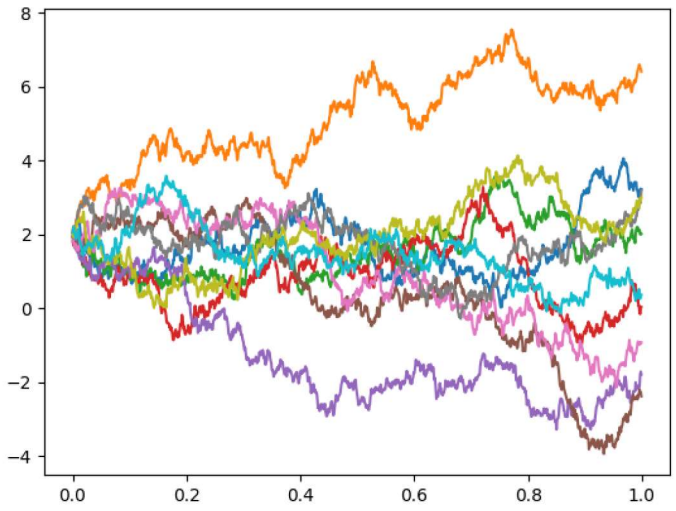
Definirane funkcije vidimo u sljedećem bloku.

```
1 def BG(dt,N):
2     return (np.sqrt(dt) * np.random.randn(N))
3 
4 def plotExEm(N,dt,S,ex,paths):
5     fig, ax = plt.subplots();ax.grid()
6 
7     for i in range (paths):
8         dB = BG(dt,N)
```

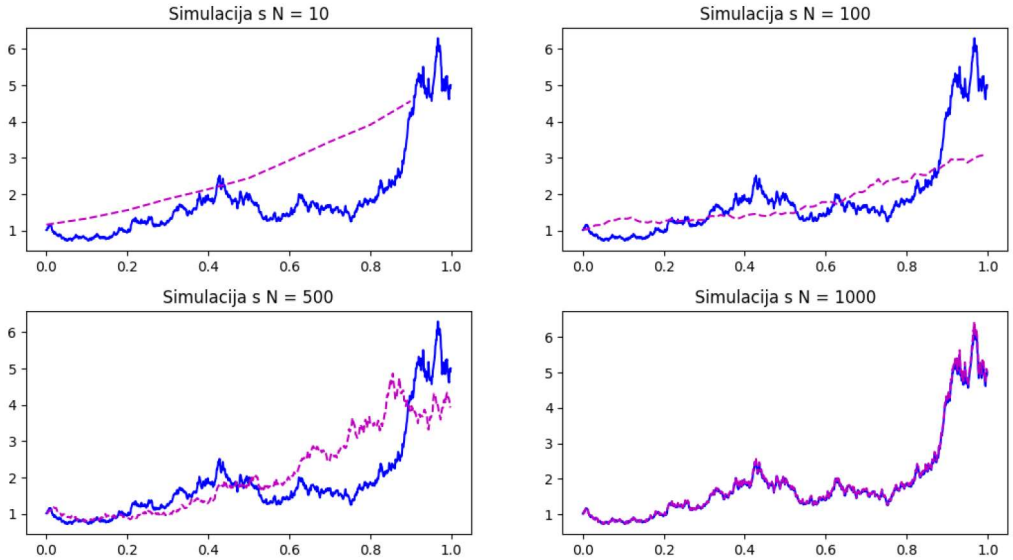
```
        XEm = S.EM(N,dt,dB)
        t = np.arange(0,1,dt)
        if ex:
            Exact = S.Exact(t,dB)
            plt.plot(t,Exact, label = "Exact ($X_t$)", color
    = 'b')
        plt.plot(t,XEm,label="EM ($X_t$)")
        if S.name == "GBG":
            plt.title("Geometrijsko Brownovo gibanje")

        else:
            plt.title("Ornstein-Uhlenbeck proces")
        plt.xlabel("vrijeme")
     plt.show()
```

Listing 3: Dio Python koda u kojem su definirane funkcije BG i plotExEm

Na Slici 3 možemo vidjeti simulaciju deset trajektorija OU procesa generiranih Euler-Maruyama metodom implementiranoj u prethodno prikazanom kodu.
Na Slici 4 prikazali smo trajektoriju SDJ za GBG (označena plavom bojom) te aproksimacije Euler-Maruyama shemom (označena ljubičastom bojom) uz povećavanje broja točaka N u subdiviziji segmenta $[0, T]$. Primjećujemo da kako povećavamo N, odnosno smanjujemo vremenski korak (jer je definiran kao $\frac{T}{N}$), tako se trajektorija aproksimacije približava trajektoriji egzaktnog rješenja (Slika 4). Približavanje trajektorija smanjenjem vremenskih pomaka pomnije ćemo objasniti u sljedećem potpoglavlju.



Slika 3: Deset trajektorija OU procesa na intervalu $[0,\ 1]$ s parametrima $\mu = 0.5, \theta = 0.5, \sigma = 3$ dobivenih Euler-Maruyama aproksimacijskom shemom temeljenoj na $N = 1000$ točaka s početnim uvjetom $X_0 = 2$.



Slika 4: Aproksimacija i egzaktno rješenje SDJ za GBG s početnim uvjetom $X_0 = 1$ i parametrima $\mu = 1.5, \sigma = 1$

3.2 Konvergencija Euler-Maruyama metode

Metode aproksimacije klasificiramo po različitim svojstvima. Najčešća svojstva koja se koriste u literaturi su jaka i slaba konvergencija.

Definicija 3.1. Za numeričko rješenje $(X_t^{(n)}, 0 \leq t \leq T)$ koje predstavlja aproksimaciju slučajnog procesa $(X_t, 0 \leq t \leq T)$ na temelju subdivizije τ s dijametrom δ kažemo da konvergira u jakom smislu reda γ ako za proizvoljan fiksni T vrijedi:

$$E[|X_T^{(n)} - X_T|] \leq C\delta^{\gamma}, \quad \forall \delta < \delta_0, \tag{23}$$

za $\delta_0 > 0$, gdje je C konstanta nezavisna od δ.

Definicija 3.2. Za numeričko rješenje $(X_t^{(n)}, 0 \leq t \leq T)$ koje predstavlja aproksimaciju slučajnog procesa $(X_t, 0 \leq t \leq T)$ na temelju subdivizije τ s dijametrom δ kažemo da konvergira u slabom smislu reda γ ako za fiksni T i za svaku neprekidnu funkciju g koja je derivabilna reda $(\delta + 1)$, odnosno $g \in C_P^{2(\delta+1)}$, postoji pozitivna konstanta C nezavisna od δ te $\delta_0 > 0$ tako da:

$$|E[g(X_T)] - E[g(X_T^{(n)})]| \leq C\delta^{\beta}. \tag{24}$$

Pojasnimo malo razliku između ove dvije vrste konvergencije te se osvrnimo na njihovu korisnost u praksi.
Kriterij za slabu konvergenciju temelji se na razlici očekivanih vrijednosti stohastičkih procesa u danom trenutku T. S druge strane, kriterij jake konvergencije temelji se na očekivanoj vrijednosti apsolutne pogreške u danom trenutku T te nam osigurava da aproksimacija bude blizu egzaktnog rješenja u zadanom limitu. Dakle, jaku konvergenciju ćemo koristiti kada nas zanima koliko je trajektorija naše aproksimacije blizu egzaktnog rješenja u trenutku T, a slabu konvergenciju ako nas zanima očekivana vrijednost neke funkcije procesa.

Iskažimo teorem koji nam daje rezultat o jakoj konvergenciji Eulerove metode.

Teorem 3.1. *Neka je*

$$E[|X_0|^2] < \infty \tag{25}$$

$$E[|X_0 - Y_0|^2]^{\frac{1}{2}} \leq C_1\delta^{\frac{1}{2}}, \tag{26}$$

$$|\mu(t,x) - \mu(t,y)| + |\sigma(t,x) - \sigma(t,y)| \leq C_2|x-y|, \tag{27}$$

$$|\mu(t,x)| + |\sigma(t,x)| \leq C_3(1+|x|), \tag{28}$$

$$|\mu(s,x) - \mu(t,x)| + |\sigma(s,x) - \sigma(t,x)| \leq C_4(1+|x|)|s-t|^{\frac{1}{2}}, \tag{29}$$

za sve $s,t \in [0,T]$ *i* $x,y \in \mathbb{R}$ *gdje su* $K_1,..,K_4$ *konstante koje ne ovise o* δ. *Tada za Eulerovu aproksimaciju* Y^δ *vrijedi:*

$$E[|X_T - Y^\delta(T)|] \leq C_5\delta^{\frac{1}{2}}, \tag{30}$$

gdje je C_5 *konstanta nezavisna o* δ.

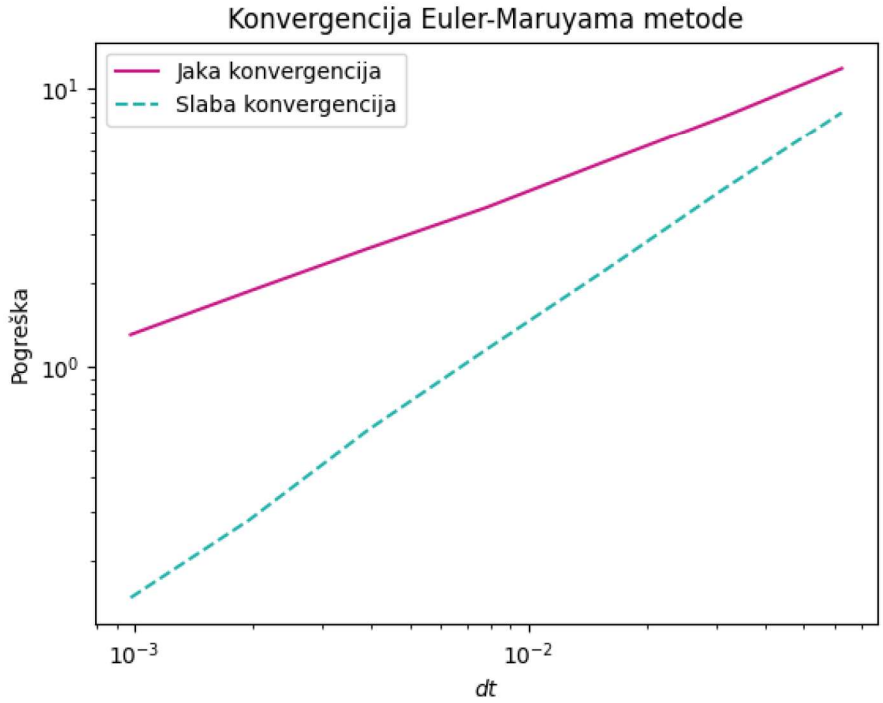
Iz Teorema 3.1 vidimo da Euler-Maruyama metoda konvergira jako, reda $\frac{1}{2}$. Dokaz Teorema 3.1 može se pronaći u [8].

Pogledajmo sada implementaciju te prikaz kriterija za jaku i slabu konvergenciju u ovisnosti o *dt*.

```
    strongError, weakError = [], []
    dt_grid = [2 ** (R-10) for R in range(7)]
    numberOfSample = 1000
    # ponavljajmo po dt
    for dt in dt_grid:
        t = np.arange(dt, 1 + dt, dt)
        N = len(t)
        # inicijalizacija vektora
        eulerError = np.zeros(N)
        YSum, XEmSum  = np.zeros(N), np.zeros(N)

        # generiramo 10000 uzoraka
        for i in range(m):
            dB = BG(dt,N)
            B = np.cumsum(dB)

            Y = S.Exact(t,dB)

            Xem = S.EM(N,dt,dB)

            # Racunanje apsolutne pogreske
            # i dodavanje onima iz prijasnjih uzoraka
            eulerError  += abs(Y - Xem)

            # Dodamo Y,X onima iz prijasnjih uzoraka
            YSum += Y
            XEmSum += Xem

        # racunamo maksimum prosjeka -> jaka konvergencija
        strongError.append(max(eulerError / numberOfSample))
        # racunamo maksimum pogreske prosjeka -> slaba
konvergencija
```

```
32          weakError.append(max(abs(YSum - XEmSum)/
    numberOfSample))
33
34
35 plt.loglog(dt_grid, strongError, label="Jaka konvergencija",
    color='mediumvioletred')
36 plt.loglog(dt_grid, weakError, label="Slaba konvergencija",
    color='lightseagreen',ls='--')
37 plt.title('Konvergencija Euler-Maruyama metode')
38 plt.xlabel('$dt$'); plt.ylabel('Pogreska'); plt.legend(loc=2)
39 plt.show()
```

Listing 4: Python kod za provjeru kriterija jake i slabe konvergencije Euler-Maruyama metode



Slika 5: Vrijednosti kriterija za jaku i slabu konvergenciju Euler-Maruyama aproksimacijske sheme u ovisnosti o dijametru subdivizije

Na Slici 5 vidimo da se odstupanje iz kriterija za jaku i slabu konvergenciju smanjuje s povećanjem točaka subdivizije segmenta $[0, T]$, tj. sa smanjenjem duljine dt podintervala generiranih tom subdivizijom, iz čega možemo zaključiti da metoda konvergira i u jakom i u slabom smislu. Također, primjećujemo da je pogreška za slabu konvergenciju manja od one za

jaku konvergenciju. Aproksimacijske sheme koje imaju red konvergencija u jakom smislu često istovremeno imaju viši red konvergencije u slabom smislu.
Zašto nam je red konvergencije uopće važan?
Kada koristimo numeričke metode, osim što znamo da se pogreška smanjuje sa što manjim vremenskim korakom najčešće nas zanima i koliko brzo aproksimacija konvergira smanjenjem vremenskog koraka. Razlog tomu je što smanjenje vremenskog koraka uzrokuje dulje vrijeme izvođenja koda. Također, zbog *ograničenja* računala, odnosno računalne pogreške, imamo granicu do koje možemo smanjivati vremenski korak.
Tu nam do izražaja dolazi *red konvergencije* koji nam označava kojom brzinom će pogreška padati. Dakle, ako je red konvergencije za metodu γ i naš korak je l puta manji, tada će pogreška aproksimacije biti l^γ puta manja. Na primjer za red jednak 1, ako želimo pogrešku smanjiti 10 puta, korak moramo napraviti 10 puta manjim, dok za red - $\frac{1}{2}$ moramo vremenski pomak smanjiti 10000 puta.
Kako bismo dobili red konvergencije prvo ćemo iskazati *Markovljevu nejednakost*, a više informacija o istoj moguće je pronaći u [3].

Propozicija 3.2. *Neka je $r>0$ i $E[|X|^r] < \infty$. Tada za svaki $\varepsilon > 0$ vrijedi:*

$$P(|X| \geq \varepsilon) \leq \frac{E[|X|^r]}{\varepsilon^r}. \tag{31}$$

Sada koristeći Markovljevu nejednakost za $r = 1$ i pretpostavku da numeričko rješenje $(X_t^{(n)}, 0 \leq t \leq T)$ konvergira u jakom smislu dobivamo:

$$P(|X_T^{(n)} - X_T| \geq \varepsilon) \leq E[|X_T^{(n)} - X_T|]/\varepsilon \leq C\delta^\gamma/\varepsilon. \tag{32}$$

Uzmimo da je $\varepsilon = \delta^{\gamma/2}$ te okrenimo smjer nejednakosti:

$$P(|X_T^{(n)} - X_T| < \delta^{\gamma/2}) \geq 1 - C\delta^{\gamma/2}. \tag{33}$$

Izaberemo li da je konstanta C pozitivan i mali broj, onda iz (33) vidimo da vjerojatnost da je pogreška u fiksnoj točki mala sve bliže 1 što je dijametar subdivizije manji.

Kako bismo dobili naš red konvergencije pretpostavit ćemo da je n dovoljno velik i ε dovoljno mali tako da, na temelju ocjene (32), vrijedi:

$$\log\left(E(|X_T^{(n)} - X_T|)\right) \approx \log C + \gamma \log \delta. \tag{34}$$

Sada ćemo iskoristiti dobiveno te uz pomoć Python programskog jezika i biblioteke *statsmodels.api*, točnije funkcije *LOS*, izračunati red konvergencije Euler-Maruyama metode. Korišteni kod prikazan je u sljedećem bloku, a dobiveni rezultat u Tablici 2.

```
X = sm.add_constant(np.log(dt_grid))

#Primjenjujemo funkciju OLS na prijasnju simulaciju
resultsS = sm.OLS(np.log(strongError),X).fit()
    print("\n Red Jake konvergencije EM metode      "+ str(
   resultsS.params[1]))

resultsW = sm.OLS(np.log(weakError),X).fit()
    print(" Red Slabe konvergencije EM metode     "+ str(
   resultsW.params[1]))
```

Listing 5: Python kod za računanje reda konvergencije

Red jake konvergencije EM metode	0.50187
Red slabe konvergencije EM metode	0.97493

Tablica 1: Dobivene vrijednosti za red konvergencije

Dobivenim rezultatom pokazano je da red jake konvergencije naše metode $\gamma = \frac{1}{2}$, što smo mogli vidjeti i u Teoremu 3.1. Red slabe konvergencije je $\gamma = 1$.

4 Pseudo-likelihood metoda procjene parametara temeljena na Euler-Maruyama aproksimacijskoj shemi

Procjena parametara u SDJ predstavlja ključni korak u statističkoj analizi difuzija. Ovaj proces uključuje pronalaženje procjena za parametre SDJ koje najbolje opisuju stvarne podatke.
Iako postoje mnoge metode za procjenu parametara, kada su u pitanju SDJ najčešće se koriste metode kao što je *pseudo-likelihood*. Aproksimacijske metode rade aproksimaciju trajektorije procesa tako da je za numeričko rješenje SDJ moguće eksplicitno konstruirati likelihood funkciju.
Povežemo pseudo-likelihood metodu s Euler-Maruyama aproksimacijom.
Diskretizacijom (6) Eulerovom shemom na intervalu $[t, t+\Delta t\rangle$ dobivamo:

$$X_{t+\Delta t} - X_t = \mu(\theta, X_t)\Delta t + \sigma(\theta, X_t)(B_{t+\Delta t} - B_t). \tag{35}$$

Razlika $X_{t+\Delta t} - X_t$ je normalno distribuirana s očekivanjem $\mu(\theta, X_t)\Delta t$ i varijancom $\sigma^2(\theta, X_t)\Delta t$. Dakle, prijelazna funkcija gustoće aproksimacije, tj. numeričkog rješenja, dana je sljedećim izrazom:

$$p_\theta(t, y|x) = \frac{1}{\sqrt{2\pi t\sigma^2(\theta, x)}} exp\left(-\frac{1}{2}\frac{(y - x - \mu(\theta, x)t)^2}{t\sigma^2(\theta, x)}\right). \tag{36}$$

Sada pseudo-likelihood funkciju definiramo preko prijelaznih funkcija gustoća s obzirom na točke subdivizije segmenta $[0, T]$.
Ako su nam parametri u vektoru θ različiti za drift parametar te za parametar difuzije ili ako je Δ (dijametar subdivizije) dovoljno mali, ova metoda daje dobre rezultate. Dakle, pretpostavimo da je $\sigma(\theta, x) = \sigma$ pozitivna konstanta te da su svi ostali parametri u $\mu(\theta, x)$, odnosno σ nije jedan od parametara u μ. Nadalje pretpostavimo da je subdivizija na kojoj se temelji aproksimacija ekvidistantna. Pretpostavimo i da $\forall k > 0$ svi momenti reda k procesa postoje tako da je

$$\sup_{0\le t\le T} E|X_t|^k < \infty. \tag{37}$$

Sada možemo definirati log-likelihood funkciju poznatu i kao lokalna Gaussova aproksimacija:

$$l_n(\theta) = -\frac{1}{2}\sum_{i=1}^{n} \frac{(X_i - X_{i-1} - \mu(\theta, X_{i-1})\Delta)^2}{\sigma^2\Delta} + n\log(2\pi\sigma^2\Delta). \tag{38}$$

Kada uzmemo u obzir da je σ^2 konstantan, tada je maksimizacija log-likelihood funkcije ekvivalentna maksimizaciji funkcije:

$$g(\theta) = \sum_{i=1}^{n}(X_i - X_{i-1})\mu(\theta, X_{i-1}) - \frac{\Delta}{2}\sum_{i=1}^{n}\mu^2(\theta, X_{i-1}). \tag{39}$$

Ako bismo dodali uvjet da $n\Delta_n^3 \to 0$, tada (39) vodi do konzistentnog i asimptotski normalnog procjenitelja parametara θ to je pokazano u [5]. U ovom slučaju potrebno je prvotno procijeniti parametar difuzije prije procjene θ. Nadalje, u [5] je upućeno na dokaz da je za jednodimenzionalne slučajeve konzistentan procjenitelj za σ^2:

$$\widehat{\sigma}^2 = \frac{1}{n\Delta}\sum_{i=1}^{n}(X_i - X_{i-1})^2. \tag{40}$$

Napravimo sada implementaciju u Pythonu. Biblioteke koje će nam tu biti potrebne su *numpy* i *scipy*, odnosno koristili smo *scipy.optimize* proceduru. Definirali smo drift funkciju za Euler-Maruyama aproksimaciju te pseudo-likelihood funkciju u sljedećem primjeru.

Primjer 3. *Simulirajmo OU proces koristeći SDJ* (12) *s početnim uvjetom* $X_0 = 1$ *i parametrima* $\mu = 5, \theta = 3, \sigma = 2$ *te provedimo procjenu parametara u pet scenarija koji se razlikuju s obzirom na duljinu intervala* $[0, T]$ *i dijametar subdivizije* $\frac{T}{N}$*, gdje je u svim scenarijima* $N = 2500$*. Početni interval neka bude* $[0, 0.1]$ *te je u svakom sljedećem desni rub intervala povećan za* 0.1*. Dodatno, izračunajmo srednje kvadratnu grešku procjene svakog parametra u svakom scenariju.*

U narednom bloku naveden je programski kod u kojem smo definirali drift za OU proces te simulirali trajektoriju OU proces s parametrima $\mu = 5, \theta = 3, \sigma = 2$ i zatim proveli procjenu parametara u pet scenarija, gdje se u svakom scenariju procjena vrši u odnosu na pedeset simulacija.

Nadalje, u Tablici 2 vidimo dobivene rezultate za pojedini scenarij te dobivenu srednje-kvadratnu grešku (RMSE, root mean square error) za svaki od njih. Vidimo da nam je greška najmanja u posljednjem slučaju na intervalu $[0, 0.5]$ te vidimo da su procjene parametara blizu njihovih stvarnih (teorijskih) vrijednosti.

```
import numpy as np
from scipy.optimize import minimize

# Drift funkcija za Eulerovu aproksimaciju
def driftEuler(x, t, x0, theta, drift):
```

```
    dd = drift(x0, theta)
    return (x - x0) * dd - 0.5 * t * dd ** 2

# Pseudo-likelihood funkcija za  Eulerovu aproksimaciju
def pseudoLikelihood(theta, X, deltat, drift):
    n = len(X)
    pseudoLikelihoods = []
    for i in range(1, n):
        dt = deltat[i - 1]
        pseudoLikelihood_i = -driftEuler(X[i], dt, X[i - 1],
   theta, drift)
        pseudoLikelihoods.append(pseudoLikelihood_i)
    return np.sum(pseudoLikelihoods)
```

Listing 6: Implementacija pseudo-likelihood funkcije u Pythonu

```

# Drift funkcija za OU proces
def driftOu(x, theta):
    return theta[0] * (theta[1] - x)

# Simulacija podataka za OU proces
np.random.seed(123)
theta_true = np.array([5, 3, 2])
N = 2500
start = 0
end = 0.1

X0 =  1
pseudoLikParamsMu = []
pseudoLikParamSigma = []
pseudoLikParamTheta = []
pseudoLikParamsMuErr = []
pseudoLikParamSigmaErr = []
pseudoLikParamThetaErr = []
for i in range (5):
    X = np.zeros(N)
    X[0] = X0
    dt = end/N
    dB = c.BG(dt,N)

    for j in range(50):

        deltat = np.random.uniform(start,end, N-1)  #
   Nasumicno vrijeme izmedu start i end

        sx = 0
        for i in range(1,N):
            X[i] = X[i - 1] + driftOu(X[i - 1], theta_true) *
    deltat[i - 1] + np.sqrt(deltat[i - 1]) * np.random.normal
```

```
(0, 1, 1)
33              sx += (X[i]-X[i-1])**2
34          # Procjena parametara uz Maximum Likelihood
35          initial_guess = [1.5, 1]
36          result = minimize(lambda theta: pseudoLikelihood(
    theta, X, deltat, driftOu),\
37                            initial_guess)
38          estimatedTheta = result.x
39
40          pseudoLikParamsMu.append(estimatedTheta[0])
41          pseudoLikParamTheta.append(estimatedTheta[1])
42          s=  np.sqrt ( np.mean ( np.square ( np.diff ( X ) ))
    / deltat  )
43          pseudoLikParamSigma.append(s)
44
45          pseudoLikParamsMuErr.append((estimatedTheta[0]-
    theta_true[0])**2)
46          pseudoLikParamThetaErr.append((estimatedTheta[1]-
    theta_true[1])**2)
47          pseudoLikParamSigmaErr.append((s-theta_true[2])**2)
48
49      mu = np.mean(pseudoLikParamsMu)
50      theta = np.mean(pseudoLikParamTheta)
51      sigma = np.mean(pseudoLikParamSigma)
52
53      print("Procjena parametara (mu, theta, sigma):", mu,
    theta,sigma, \
54              " za uniformno distribuirane vremenske korake
    izmedu ", start, " i ",end)
55
56
57      RMSEMU = math.sqrt((np.mean(pseudoLikParamsMuErr)))
58
59      RMSETHETA = math.sqrt((np.mean(pseudoLikParamThetaErr)))
60
61      RMSESIGMA = math.sqrt((np.mean(pseudoLikParamSigmaErr)))
62
63
64      print("Root Mean Square Error:\n")
65      print(RMSEMU," ",RMSETHETA, " ",RMSESIGMA )
66      print('\n')
67      end += 0.1
```

Listing 7: Python kod za provedbe pseudo-likelihood procjene parametara

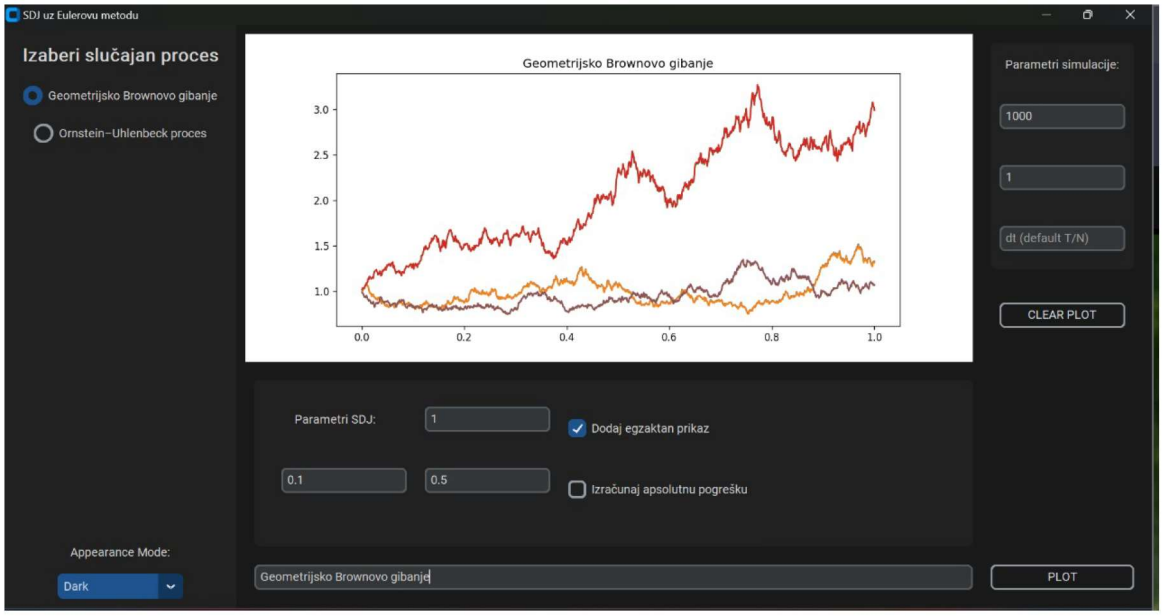
interval Δ	$\widehat{\mu}$ (RMSE)	$\widehat{\theta}$ (RMSE)	$\widehat{\sigma}$ (RMSE)
$[0, 0.1]$ (4e$-$5)	5.014 (0.241)	2.998 (0.018)	1.56 (2.905)
$[0, 0.2]$ (8e$-$5)	5.002 (0.213)	2.998(0.016)	1.65 (3.754)
$[0, 0.3]$ (1.2e$-$4)	4.997 (0.182)	2.998(0.014)	1.765 (3.515)
$[0, 0.4]$ (1.6e$-$4)	4.996 (0.165)	2.999 (0.013)	1.933 (3.459)
$[0, 0.5]$ (2e$-$4)	4.996 (0.1494)	2.998 (0.013)	2.236 (4.072)

Tablica 2: Rezultati procjene parametara OU procesa s teorijskim vrijednostima $\mu = 5, \theta = 3, \sigma = 2$

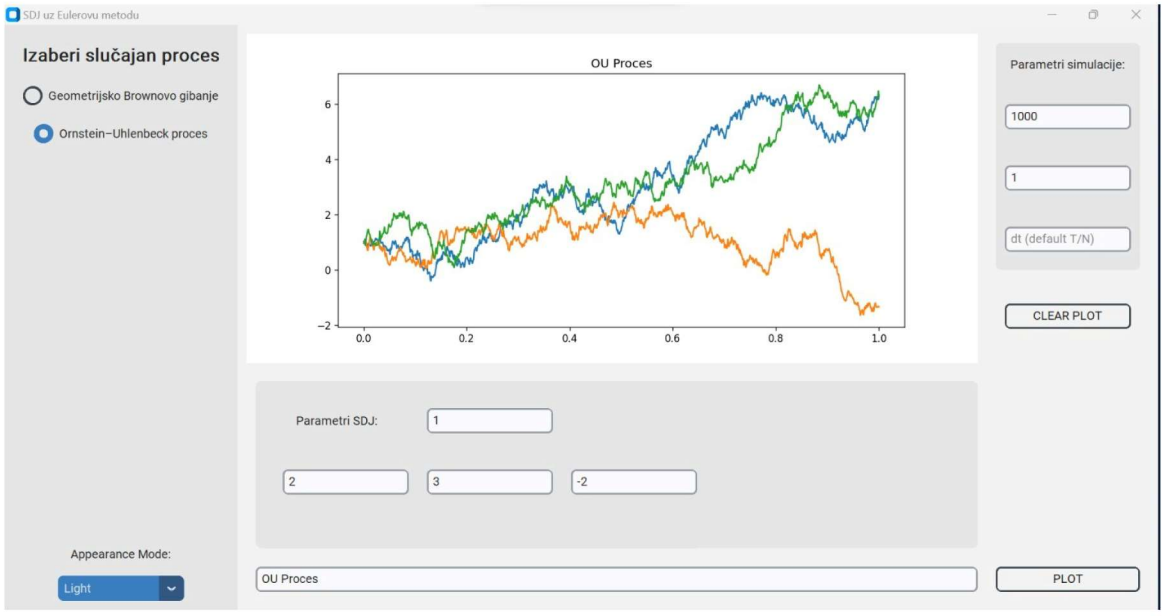
5 Grafičko sučelje

Kako bismo objedinili dijelove ovog rada u ovom poglavlju predstavit ćemo grafičko sučelje napravljeno za vizualizaciju simuliranih trajektorija GBG te OU procesa uz pomoć Eulerove aproksimacije koristeći ranije navedene metode (blok programskog koda 2). Grafičko sučelje napravljeno je u Pythonu pomoću alata *tkinter*, odnosno uz Python *UI-library* baziranom na tkinteru pod nazivom *customtkinter*.
Funkcionalnosti ovog prikaza su:

1. izbor SDJ između OU i GBG te se ovisno o odabiru dobivaju unosi za parametre,
2. omogućen je unos parametara za simulaciju - broj točaka, vremenski horizont T (pretpostavljena vrijednost dijametra je T/N),
3. za GBG moguće je dodati egzaktno rješenje te računanje pogreške,
4. dodavanje naslova prikaza,
5. prikaz trajektorija,
6. brisanje prikaza,
7. izbor tamnog ili svijetlog okruženja.



Slika 6: Prikaz simulacije triju trajektorija GBG unutar grafičkog sučelja



Slika 7: Prikaz simulacije triju trajektorija OU procesa unutar grafičkog sučelja

Literatura

[1] Mirta Benšić, Ivan Papić, *Statistika (nastavni materijali),* Fakultet primjenjene matematike i informatike u Osijeku, 2023.

[2] Ana Cenkovčan, *Ornstein-Uhlenbeckov proces (diplomski rad)*, Osijek 2013.

[3] Danijel Grahovac, *Vjerojatnost (nastavni materijali),* Fakultet primjenjene matematike i informatike u Osijeku, 2023.

[4] Desmond J. Higham, *An Algorithmic Introduction to Numerical Simulation of Stochastic Differential Equations*, Society for Industrial and Applied Mathematics 2001.

[5] Stefano M. Iacus, *Simulation and Inference for Stohastic Differential Equations*, Springer 2008.

[6] Hautahi Kingi, *Numerical SDE Simulation - Euler vs Milstein Methods* `https://hautahi.com/sde_simulation`

[7] Viktorija Kišak, *Varijacijski račun i neglatka analiza (diplomski rad)*, Zagreb 2014.

[8] Peter E. Kloeden, Eckhard Platen, *Numerical solutions of Stohastic Differential Equations*, Springer 1995.

[9] Nenad Šuvak, *Slučajni procesi II (nastavni materijali),* Fakultet primjenjene matematike i informatike u Osijeku, 2023.

[10] Nenad Šuvak, *Slučajni procesi I (nastavni materijali),* Fakultet primjenjene matematike i informatike u Osijeku, 2023.

[11] Nenad Šuvak, *Matematičke financije (nastavni materijali),* Fakultet primjenjene matematike i informatike u Osijeku, 2023.

[12] Liguo Wang, *Numerical Solutions of Stochastic Differential Equations*, University of Tennessee, Knoxville 2006.

[13] GitHub - Financial-Models-Numerical-Methods, `https://github.com/cantaro86/Financial-Models-Numerical-Methods/blob/master/6.1%20Ornstein-Uhlenbeck%20process%20and%20applications.ipynb`

[14] Python dokumentacija - https://docs.python.org/3/

Sažetak

U ovom radu definiramo stohastičke diferencijalne jednadžbe (SDJ) i dajemo kratak pregled teorije vezane uz njih. Zatim definiramo procese: geometrijsko Brownovo gibanje i Ornstein-Uhlenbeckov proces, koje kasnije koristimo kao primjere. Glavni dio rada definira Euler-Maruyama metodu za numeričko rješavanje SDJ te pruža implementaciju u programskom jeziku Python. Također predstavljamo pseudo-likelihood metodu za procjenu parametara temeljenu na Euler-Maruyama metodi. Na kraju je predstavljeno samostalno izrađeno grafičko korisničko sučelje koristeći Python za simuliranje trajektorija SDJ.

Ključne riječi

Stohastička diferencijalna jednadžba, Euler-Maruyama metoda, numeričko rješenje, geometrijsko Brownovo gibanje, Ornstein-Uhlenbeckov proces, pseudo-likelihood, procjena parametara, Python

Euler-Maruyama methode for solving stohastic differential equations with Python implementation

Summary

In this paper, we define stochastic differential equations (SDE) and the general theory connected to SDE. Afterward, we define processes: geometric Brownian Motion and Ornstein-Uhlenbeck process, which we later use in examples. The main part of the paper defines the Euler-Maruyama method for numerically solving SDEs and provides an implementation in the programming language Python. We also present the pseudo-likelihood method for parameter estimation based on the Euler-Maruyama method. Finally, a self-made graphical user interface using Python is presented for simulating SDE trajectories.

Keywords

Stohastic stochastic differential equation, Euler-Maruyama method, numerical solutions, geometric Brownian Motion, Ornstein-Uhlenbeck proces, pseudo-likelihood, parametric estimation, Python

Životopis

Rođena sam u Osijeku, 11.10.1999. godine. I dalje živim u Osijeku te sam tu pohađala i osnovnu školu te i III. gimnaziju te 2018. godine završavam svoje srednjoškolsko obrazovanje. Iste godine upisujem preddiplomski studij matematike, smjer *Matematika i računarstvo*, na Sveučilištu J.J. Strossmayera U Osijeku na Odjelu za matematiku koji završavam 2021. godine s temom završnog rada *Analiza mobilnosti građana i rasta broja zaraženih tijekom pandemije Covid-19*, izrađen pod mentorstvom prof. dr. sc. Danijela Grahovca. Na istom fakultetu upisujem i diplomski studij smjer *Matematika i računarstvo*.