CHALMERS

GÖTEBORGS UNIVERSITET

Simuleringsdriven inferens av stokastiska dynamiska system

Simulation based inference of stochastic dynamical systems

Examensarbete för kandidatexamen i matematisk statistik vid Göteborgs universitet

Kandidatarbete inom civilingenjörsutbildningen vid Chalmers

Alfred Andersson

Vilgot Jansson

Noah Trägårdh

Jacob Welander

Victor Wellsmo

Institutionen för Matematiska vetenskaper
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2023

Simuleringsdriven inferens av stokastiska dynamiska system

Examensarbete för kandidatexamen i matematisk statistik vid Göteborgs universitet
Noah Trägårdh

Kandidatarbete i matematik inom civilingenjörsprogrammet i Teknisk fysik vid Chalmers tekniska högskola
Jacob Welander Victor Wellsmo

Kandidatarbete i matematik inom civilingenjörsprogrammet i Kemiteknik vid Chalmers tekniska högskola
Alfred Andersson

Kandidatarbete i matematik inom civilingenjörsprogrammet i Teknisk matematik vid Chalmers tekniska högskola
Vilgot Jansson

Handledare: Petar Jovanovski

Institutionen för Matematiska vetenskaper
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2023

Förord

Gruppen vill rikta ett stort tack till vår handledare Petar Jovanovski som varit ett viktigt stöd genom hela skrivprocessen. Genom veckovisa möten har han svarat på gruppens frågor, lett engagerande genomgångar av teori samt bidragit med användbar litteratur.

En loggbok har förts veckovis på ett roterande schema över gruppmedlemmarnas enskilda bidrag till arbetet och nuvarande arbetsfas för rapporten. Loggboken baserades på mötesanteckningar från möten med Petar samt en tidslogg över individuell arbetstid.

Nedan är en bidragsrapport sammanställd som beskriver vilka gruppmedlemmar som författat respektive del av rapporten. Namn utan parentes avser huvudförfattare medan namn inom parentes hjälpt till att vidareutveckla delarna. Notera att alla gruppens medlemmar har gett bidrag till de flesta stycken även om det inte listas i bidragsrapporten.

Bidragsrapport

Populärvetenskaplig presentation - Jacob, Vilgot

Abstract/sammandrag - Alfred

1 Inledning - Victor, Noah

- 1.1 Syfte - Alla
- 1.2 Frågställning - Alla
- 1.3 Avgränsningar - Alla
- 1.4 Rapportens disposition - Alfred

2 Teori - Jacob, Noah

- 2.1 Stokastiska processer - Noah, Jacob
- 2.2 Stokastisk analys - Noah, Jacob
- 2.3 CKLS-processer - Noah, Vilgot
- 2.4 Simulering av lösningar till SDE:er - Vilgot, Alfred
- 2.5 Inferens av parametrar - Jacob, (Alfred)
- 2.6 Markov Chain Monte Carlo - Alfred, (Noah)
- 2.7 Approximate Bayesian Computation - Victor, (Alfred)
- 2.8 Wasserstein-avstånd - Alfred

3 Metod - Vilgot, Alfred, (Victor)

- 3.1 Implementering av MH - Alfred, (Vilgot)
- 3.2 Implementering av ABC-R - Alfred, Vilgot
- 3.3 Implementering av ABC-SMC - Victor, Vilgot
- 3.5 Olika mått på prestanda - Vilgot, Jacob

4 Resultat - Alfred, Victor, (Jacob)

- 4.1 MH-EM och ABC-metodernas noggranhet för olika tidssteg Δt - Alfred, Victor, (Jacob)
- 4.2 ABC-metodernas tidsåtgång för olika tidssteg Δt - Alfred, Victor, (Vilgot)
- 4.3 Hur toleransen ϵ påverkar ABC-R - Alfred, Victor
- 4.4 Hur antal partiklar och val av toleranskvantil påverkar ABC-SMC - Victor, (Vilgot)

5 Diskussion och slutsats - Alfred, Jacob

- 5.1 Etiska aspekter - Jacob
- 5.2 Slutsats - Alla

A Appendix 1 - begrepp och akronymer

- A Begreppslista - Alla

B Appendix 2 - teori och resultat

- B.1 Sannolikhetsteori - Noah, Jacob
- B.2 Lösning av Ornstein-Uhlenbeck processen - Noah
- B.3 Stark och svag konvergens för Euler-Maruyama-metoden - Noah
- B.4 Syntetisk data - Victor, (Alfred)
- B.5 Neurala nätverk - Victor
- B.6 Resultat - Alfred, Victor

C Appendix 3 - källkod - Victor, Vilgot, Alfred

Populärvetenskaplig presentation

Meteorologer som förutsäger vädret, Newton som beräknade tyngdaccelerationen och barn som skakar och lyssnar på sina julklappar har alla en sak gemensamt - de utför parameterinferens. Det är en metod som låter oss vara detektiver i matematikensvärld och bättre förstå världens regler och hur saker förändras över tid.

Tänk exempelvis att du slår en golfboll och vill förutsäga vart den landar. Med fysikens lagar kan du beskriva hur bollen får röra sig, men bollens bana beror också på till exempel hur hårt du slår bollen eller hur mycket massa bollen har - vi säger då att dess massa är en parameter. Om vi vet bollens massa kan vi räkna ut en potentiell bana för bollen genom att simulera den i många steg. Men verkligheten är mer komplicerad och slumpen kommer påverka hur bollen flyger i form av vind och annat. För att beskriva detta så använder vi stokastiska differentialekvationer.

Parameterinferens är att vi går åt andra hållet. Givet att vi vet att bollen har landat någonstans vill vi veta hur tung den är och för att få så mycket information som möjligt vill man också analysera hur bollen flög i luften och inte bara vart den landade. Detta kan göras genom att filma bollen. Då får vi information om bollens position vid ett stort antal tidpunkter - en för varje bild på filmrullen. Men även om vi vet hela bollbanan kan vi fortfarande inte bestämma massan exakt eftersom många fenomen är slumpmässiga. En tung boll i medvind kanske får samma bana som en lätt boll i motvind så det bästa svaret på frågan - vilken massa hade bollen? - är egentligen sannolikheter för vilka massor som är mest troliga.

Det finns olika metoder för att utföra inferens - att uppskatta bollens massa. Metropolis-Hastings metod räknar ut sannolikheterna direkt från den stokastiska differentialekvationen - regeln för bollens rörelse. När Metropolis-Hastings metod fungerar så är den jättebra på att räkna ut sannolikheterna för parametrarna så den kan användas som en mätsticka eller facit för jämföra hur bra andra inferensmetoder är. Men tyvärr är reglerna för de flesta system för matematiskt komplicerade för att lösa på det viset vilket har drivit forskare till att utveckla nya - mer generella inferensmetoder.

En sådan mer generell metod är Metropolis Hastings-Euler Murayama metoden som använder numeriska uppskattningar för att försöka göra samma sak som Metropolis Hastings algoritm gör. Den bygger på antagandet att bollen inte hinner flytta på sig så mycket mellan varje bild vilket gör att det är en bra metod när man har en bra tidsupplösning på datan man undersöker. Men om bollen hinner röra på sig mycket mellan bilder blir approximationerna dåliga och metoden ger dåliga uppskattningar för parametrarna. Därför behövs ytterligare andra metoder.

I vårt arbete undersöker vi en klass simuleringsdrivna inferensmetoder som går under samlingsnamnet ABC och använder ett helt annat sätt för att utföra inferens. De kan förklaras som om att vi låter en dator slå bollar med olika massor ett stort antal gånger. De simulerade banorna jämförs sedan med den verkliga bollbanan och de bollar som hade en bana som var tillräckligt lik den verkliga sparas och deras massor räknas då som mer sannolika.

Detta kommer inte att räcka för att förutse bollens rörelse eftersom det finns fler viktiga faktorer. Parametrar som gravitationen samt bollens form och tvärsnittsarea påverkar också hur bollen flyger. I teorin är det inga problem - vi får helt låta datorn simulera bollbanor där vi även varierar de parametrarna. Den enklaste ABC-metoden, som kallas ABC-R, går ut på att vi väljer parametrar slumpmässigt i ett valt intervall. Men denna naiva metod gör att jättemånga olika banor måste simuleras för att jämförelsen ska bli tillförlitlig och metoden blir snabbt mycket beräkningstung. Därför har andra metoder utvecklats som bygger på samma tanke men som bland annat kan ha ett smartare sätt att välja vilka parametrar som prövas. Istället för att bara välja parametrar helt slumpmässigt vore det exempelvis bra om vi kom ihåg vilka slag vi har gjort så att vi kan simulera fler av de slag som kom nära den undersökta banan. En metod som vi testade och bygger på den idén är ABC-SMC.

Våra resultat visar att ABC-metoderna fungerar bra även i de fall där MH-EM fallerar. Den smarta ABC-SMC visar sig dessutom kunna minska beräkningstiden hundrafalt jämfört med den naivare ABC-R. Metoderna kan alltså göra det möjligt att lösa massor av spännande problem som länge gäckat forskarna.

Så nästa gång du hör talas om spännande forskning inom ekonomi, fysik eller biologi, kom ihåg att stokastiska differentialekvationer är en viktig del av deras verktygslåda. Dessa matematiska detektiver hjälper oss att avslöja hemligheterna som står som grunden för en osäker och oförutsägbar värld och gör det möjligt för oss att hitta svar på många av de mysterier vi står inför.

Sammandrag

Stokastiska modeller, som ger tillförlitlig och användbar information om ett systems beteende, består ofta av stokastiska differentialekvationer (SDE) vars likelihoodfunktion inte är analytiskt tillgänglig. Mer traditionella Markov Chain Monte Carlo-metoder (MCMC) samt relativt nyligen utvecklade likelihood-fria Approximate Bayesian Computation-metoder (ABC) utgör populära angrepssätt för att utföra inferens på dessa typer av problem. För att bidra till utvecklingen av och förståelsen för ABC-metoder ger denna studie en överblick av två olika ABC baserade algoritmer, Rejection sampling (ABC-R) och Sequential Monte Carlo (ABC-SMC), och jämför dessa med Metropolis-Hastings Euler-Maruyama-metoden (MH-EM). Metodiken innefattar numerisk diskretisering, simulering och inferens av Ornstein-Uhlenbeck-modellen vars analytiska lösning är tillgänglig och används som referens för jämförelse av metodernas noggrannhet. Resultaten visar att ABC-metoderna kan användas för att utföra inferens med god noggrannhet, även på data med stora tidssteg för vilken MH-EM fallerade. De visar också fördelar med ABC-SMC jämfört med ABC-R då mått som noggrannhet och effektivitet sammanvägs.

Abstract

Stochastic models, that give reliable and useful information about the behaviour of a system, often consists of stochastic differential equations (SDE) with intractable likelihood functions. More traditional Markov Chain Monte Carlo (MCMC) methods as well as the relatively recently developed likelihood free Approximate Bayesian Compuation (ABC) methods are popular approaches for doing inference on these type of problems. To contribute to the development and understanding of ABC-methods, this study gives an overview of two different ABC-algorithms, Rejection sampling (ABC-R) and sequential Monte Carlo (ABC-SMC), and compares these with the Metropolis-Hastings Euler-Maruyama (MH-EM) method. The methodology includes numerical discretization, simulation and inference of the Ornstein-Uhlenbeck model of which an analytical solution exists and are used as a baseline for the results. Our findings demonstrate that ABC methods can provide accurate inference, even on data with large time steps where MH-EM fails. Moreover, they show benefits of using ABC-SMC compared with using ABC-R when considering accuracy and efficiency.

Innehållsförteckning

1 Inledning

Stokastisk modellering är ett viktigt område inom tillämpad matematik där man studerar beteendet hos system som präglas av slump och oförutsägbarhet [1][2]. Inom ämnet används statistiska metoder för att analysera och förutsäga det stokastiska systemets beteende, vilket gör det möjligt att lösa problem inom en rad olika vetenskapsområden såsom ekonomi, fysik och biologi.

Stokastiska differentialekvationer (SDE) är matematiska modeller som beskriver förändringen av ett stokastisk system över tid och kombinerar element från både deterministiska differentialekvationer och stokastiska processer [3]. På grund av slumpmässiga variationer i systemet, är det svårt att hitta analytiska lösningar till SDE:er, vilket gör det nödvändigt att använda numeriska lösningar. I denna rapport används Euler-Maruyama-metoden (EM) vilket är en generalisering av Eulers metod för ordinära differentialekvationer (ODE) för att simulera lösningsbanor till SDE:er. Även om numeriska lösningar inte kan ge exakta resultat, ger de oftast en tillräckligt noggrann bild av förväntat beteende för en given process och är således användbara verktyg.

I detta arbete ligger fokus på en specifik stokastisk modell känd som Ornstein-Uhlenbeck-processen (OU), som beskriver tidsutvecklingen av en stokastisk variabel med en tendens att pendla runt en jämviktsnivå [4], vilket är en användbar modell inom stokastisk modellering. Eftersom övergångstätheterna för OU kan uttryckas analytiskt är det möjligt att beräkna likelihood-funktionen som beskriver sannolikheten att den observerade datan genererats av olika modellparametrar. Det gör att OU kan användas som referensmodell för att jämföra olika likelihood-fria inferensmetoder mot ett tydligt facit i form av resultaten från den likelihood-baserade Metropolis-Hastings (MH-exakt). De likelihood-fria metoder som jämförs i denna rapport är Metropolis-Hastings Euler-Maruyama (MH-EM) samt de två Approximate Bayesian Computation-metoderna (ABC) Sequential Monte Carlo (ABC-SMC) och Rejection Sampling (ABC-R)

MH är en Markov Chain Monte Carlo-metod (MCMC) som bygger på markovkedjor och möjliggör dragningar från fördelningar som är svåra att beräkna direkt [5]. Markovkedjor beskriver system där det nuvarande tillståndet endast är beroende av det föregående tillståndet och som är oberoende av all tidigare information. I MH bestäms övergångsfördelningen mellan olika tillstånd i markovkedjan av en acceptanssannolikhet som beror på likelihood-funktionen. Vid beräkning når markovkedjan, efter en initial inställningsperiod, en stationär fördelning där varje nytt steg är en dragning från målfördelningen. I MH-EM används en EM-approximation av likelihood-funktionen istället för den exakta, och metoden kallas därför pseudo-likelihood-baserad.

ABC är en klass av metoder som utvecklats för att möta utmaningar som uppstår vid analys av dynamiska system. ABC-metoderna kringgår problemet med otillgänglig likelihood-funktion genom att använda simuleringar av modellen istället för att beräkna likelihood-funktionen direkt.

ABC-R är en av de tidigast utvecklade och mest grundläggande ABC-metoderna. Den introducerades av Pritchard et al. (1999) [6] och Tavaré et al. (1997) [7] som ett sätt att uppskatta a posteriori-fördelningen för en stokastisk modells parametrar genom att använda sig av en a priori-fördelning och jämföra den simulerade datan med den observerade. Parametrarna accepteras som en dragning ur den posteriora fördelningen om avståndet mellan simulerad och observerad data är mindre än en förutbestämd toleransnivå (ϵ). ABC-R är enkel att implementera och har acceptabel effektivitet för problem där det är möjligt att föreslå en a priori-fördelning med begränsat omfång. Detta kräver liten initial osäkerhet kring parametrarnas värden samt ett lågt antal modellparametrar. Beräkningstiden för ABC-R ökar dock snabbt när acceptanskvoten minskas och då a priori-fördelningen växer i omfång.

ABC-SMC är en vidareutveckling av ABC-R som syftar till att överträffa dess effektivitet och då särskilt i problem med omfångsrik a priori-fördelning. ABC-SMC introducerades av Sisson et al. (2007) [8] och Toni et al. (2009) [9] och bygger på en sekvens av ABC-R-algoritmer där toleransen minskas gradvis och kandidatfördelningen för parametrarna uppdateras i varje iteration. Parametrarna från föregående iteration viktas och störs för att utforska hela den posteriora fördelningen och för att acceptanskvoten ska hållas inom rimliga gränser. Algoritmen avslutas av ett förbestämd stoppkriterium, vilket kan vara en specifik acceptanskvot, toleransnivå eller en förändringsfaktor mellan fördelningar för två på varandra följande iterationer.

1.1 Syfte

Syftet med arbetet är att utvärdera hur metoderna ABC-R, ABC-SMC och MH-EM, presterar vid utförandet av likelihood-fri parameterinferens på realiseringar av SDE:er, i termer av noggrannhet och effektivitet. Rapporten ämnar därmed bidra till en fördjupad förståelse för metodernas begränsningar, samt belysa ABC-metodernas användbarhet i praktiska tillämpningar.

1.2 Frågeställning

Den övergripande fråga som rapporten syftar till att besvara är:

- Hur presterar metoderna MH-EM, ABC-R och ABC-SMC vid parameterinferens på realiseringar av OU i termer av noggrannhet och effektivitet?

Noggrannhet mäts som det statistiska avståndet mellan posteriori-tätheten genererad av den undersökta metoden och den som genererats av MH-exakt då de båda appliceras på en och samma realisering av OU-processen. Effektiviteten är en sammanvägning av metodernas noggrannhet och hur mycket datorkraft de använder. Vidare ska även följande frågor besvaras:

- Hur påverkas noggrannheten av tidstegets längd hos den data metoderna appliceras på?
- Hur mycket datorkraft kräver de olika ABC-metoderna?
- Hur påverkar konfigurationen av vald ABC-metod dess noggrannhet?

1.3 Avgränsningar

Trots att användningsområdet för de metoder som undersöks är problem där likelihood-funktionen inte är tillgänglig jämför denna studie metodernas prestanda på Ornstein-Uhlenbeck, vars likelihood-funktion faktiskt finns att tillgå. Anledningen till detta är att då likelihood-funktionen är tillgänglig kan exakt inferens göras med MCMC-metoder. Om exakt inferens inte kan utföras finns inget facit att jämföra prestandan av metoderna mot. Experimenten skulle lika väl ha kunnat utföras på andra SDE:er med tillgänglig likelihood-funktion, exempelvis Cox-Ingersoll-Rosignol eller Black-Scholes, två andra så kallade CKLS-processer, och just Ornstein-Uhlenbeck valdes därför att den är enkel att greppa intuitivt.

1.4 Rapportens disposition

I kapitel 2 ges en först en översiktlig beskrivning av relevanta begrepp och definitioner inom stokastisk analys och bayesiansk statistik för att sedan, med detta som grund, introducera de inferensmetoder och algoritmer som använts i arbetet. Kapitel 3 beskriver hur inferensmetoderna implementeras samt hur frågeställningarna besvaras med nödvändiga kvantifieringar. Kapitel 4 innehåller resultaten av de experiment som utförts, presenterat i både grafer och tabeller. Rapporten avslutas med diskussion och slutsats i kapitel 5.

2 Teori

Stokastiska differentialekvationer (SDE:er) skiljer sig från ordinära differentialekvationer (ODE:er) i det att förändringen i en punkt inte bara beror på funktionens värde utan också på en slumpvariabel. De "funktioner" som beskrivs av SDE:er förstås bäst som tidskontinuerliga stokastiska processer varför den gängse beteckningen är *processer*. För att förstå hur de genererade processerna beter sig och behandlas matematiskt ges en kort introduktion till stokastisk analys. Det centrala för rapporten är dock metoder för statistisk inferens. I det här fallet innebär det att, givet kända data X, identifiera parametrarnas värden i den stokastiska differentialekvation som antas ha genererat datan.

2.1 Stokastiska processer

En stokastisk process $X(\omega, t)$ är en samling slumpvariabler med ett tidsberoende och är ett matematiskt sätt att beskriva stokastiska fenomen som varierar i tiden. De är relevanta för rapporten eftersom lösningar till SDE:er, som de undersökta inferensmetoderna behandlar, kan betraktas som stokastiska processer.

Definition 1 (Stokastisk process [3]). *Låt $(\Omega, \mathcal{A}, \mathbb{P})$ vara ett utfallsrum. En reellvärd stokastisk process är en familj av slumpvariabler sådana att*

$$X(\omega, t) : \Omega \times T \to \mathbb{R}. \tag{1}$$

För ett fixt $\omega_0 \in \Omega$ så är $X(\omega_0, t)$ en funktion av t kallad en *realisering* av den stokastiska processen. Dessa realiseringar benämns *banor*. Om istället $t \in T$ fixeras så är $X(t)$ en stokastisk variabel. Om $T \subset \mathbb{Z}$ så sägs X vara en *tidsdiskret* stokastisk process och om $T \subset \mathbb{R}$ så sägs X vara *tidskontinuerlig*.

2.1.1 Brownsk rörelse beskrivs av Wienerprocessen

Brownsk rörelse betecknar ett fysikaliskt fenomen där en partikel hela tiden slumpmässigt bombarderas av ett stort antal partiklar. Partikelns olika möjliga banor över tid, beskrivs matematiskt realiseringar av Wienerprocessen.

Definition 2 (Wienerprocess [3]). *Låt $(\Omega, \mathcal{F}, \mathbb{P})$ vara ett sannolikhetsrum (se definition 8 i B.1) och antag att det för varje $\omega \in \Omega$ finns en process $\{W(t)\}_{t\geq 0}$ som besitter följande egenskaper:*

- $W(0) = 0$,
- *inkrementen $W(t) - W(s)$ för $t > s$ är oberoende,*
- *inkrementen $W(t) - W(s) \sim \mathcal{N}(0, t - s)$ för $t > s$, dvs. inkrementen är normalfördelade med väntevärde 0 och varians $t - s$ och*
- *$W(t)$, $t \geq 0$ är kontinuerliga funktioner av tid och är inte differentierbara någonstans.*

Då är $\{W(t)\}_{t\geq 0}$ en Wienerprocess.

Intuitivt kan vi se definitionen ovan på följande vis. Ett slumpexperiment utförs och utfallet är en bana ω av en Wienerprocess, så är $W(t)$ värdet på denna bana vid tiden t [10].

Definition 3 (Markovkedja [11]). *Låt $(\Omega, \mathcal{F}, \mathbb{P})$ vara ett sannolikhetsrum, $S = \{0, 1, 2, ...\}$ ett tillståndsrum och X_n en $\mathcal{F}$-mätbar funktion sådan att $X_n : \Omega \to S$. Då är $\{X_t\}_{t\geq 0}$ en Markovkedja om den besitter Markovegenskapen:*

$$\mathbb{P}(X_n = s | X_0 = x_0, X_1 = x_1, ..., X_{n-1} = x_{n-1}) = \mathbb{P}(X_n = s | X_{n-1} = x_{n-1})$$

för alla $n \geq 1$ och alla $s, x_1, ..., x_{n-1} \in S$.

I definitionen ser vi att en Markovkedjas nästa tillstånd endast beror på det omedelbart föregående tillståndet, och är oberoende av alla tidigare tillstånd i kedjan. En Wienerprocess besitter Markovegenskapen vilket vi formulerar i följande stats.

Sats 1 (En Wienerprocess är en Markovprocess [10]). *Låt $\{W(t)\}_{t\geq 0}$ vara en Wienerprocess och låt $\{\mathcal{F}(t)\}_{t\geq 0}$ vara en tillhörande filtration (se definition 9 i B.1). Då är $\{W(t)\}_{t\geq 0}$ en Markovprocess.*

2.2 Stokastisk analys

Första ordningens differentialekvationer är grundläggande i matematisk analys och med ett tillhörande initialvillkor $f(0) = x_0$ kan de som bekant skrivas på integralform:

$$df(t) = f'(t) = g(t, f(t))dt \iff f(t) = x_0 + \int_0^T g(t, f(t))dt. \tag{2}$$

I stokastisk analys tittar man på stokastiska processer istället för att endast titta på en deterministisk förändring av funktionen över tid, vilket innebär att funktionens förändring även påverkas av slumpmässiga faktorer.

Stokastisk analys är särskilt relevant i studiet av stokastiska dynamiska system och ger en teoretisk grund för att utveckla effektiva och pålitliga metoder för simulering av dem. De teoretiska insikter som stokastisk analys erbjuder medger utvecklingen av algoritmer som simulerar systemen med hög noggrannhet. Förståelse för matematiska egenskaper hos algoritmerna ger en grund för att utvärdera och jämföra deras prestanda.

2.2.1 Stokastiska Differentialekvationer

Processen

$$dX(t) = \mu(t, X(t))dt + \sigma(t, X(t))dW(t) \tag{3}$$

är en *stokastisk differentialekvation*. Funktionerna $\mu(t, x)$ och $\sigma(t, x)$ är givna och kallas för processens drift respektive diffusion. Den stokastiska differentialekvationen ovan saknar matematisk betydelse på grund av att en Wienerprocess inte är differentierbar någonstans - $dW(t)$ saknar alltså betydelse. För att få en matematisk uppfattning introducerar vi Itôprocesser och därmed Itôintegralen.

Definition 4 (Itôprocess [10]). *Låt $\{W(t)\}_{t\geq 0}$ vara en Wienerprocess med den tillhörande filtrationen $\{\mathcal{F}(t)\}_{t\geq 0}$. En Itôprocess är en stokastisk process av typen*

$$X(t) = X(0) + \int_0^T \mu(u, X(u))du + \int_0^T \sigma(u, X(u))dW(u), \qquad 0 \leq t \leq T. \tag{4}$$

Detta är (3) i integralform. Här är den första integralen är en vanlig Riemann-Stieltjesintegral och den andra är en Itôintegral.

Definition 5 (Itôintegral [10]). *Låt $0 = t_1 \leq t_2 \leq ... \leq t_n = T$ vara en partition av intervallet $[0, T]$. Itôintegralen av en mätbar process definieras då av*

$$I(t) = \int_0^T \sigma(t)dW(t) = \sum_{i=0}^{n-1} \sigma(t_i)(W(t_{i+1}) - W(t_i)). \tag{5}$$

Itôintegralen har även egenskapen att den är en martingal [3] (se definition 10 i B.1.2) och har alltså ingen tendens till att varken stiga eller sjunka med tiden.

Några viktiga egenskaper av Itôintegralen är att väntevärdet är noll och variansen ges av isometri egenskapen:

$$\mathbb{E}[I(t)] = \mathbb{E}\int_0^T \sigma(t)dW(t) = 0, \quad \text{och} \quad \mathbb{E}[I^2(t)] = \mathbb{E}\left(\int_0^T \sigma(t)dW(t)\right)^2 = \mathbb{E}\int_0^T \sigma^2(t)dt. \tag{6}$$

En annan viktig egenskap hos Itôintegralen är att dess kvadratiska variation inte är noll, vilket skiljer sig från vanlig analys där kvadratisk variation för alla funktioner med kontinuerlig derivata är noll [10]. Detta beror på att Itôintegralen inkluderar en slumpmässig komponent.

Sats 2 (Kvadratisk variation av Itôintegralen [10]). *Den kvadratiska variation som Itôintegralen samlar på intervallet* $[0, t]$ *är*

$$[I, I](t) = \left(\int_0^T \sigma(t) dW(t) \right)^2 = \int_0^T \sigma^2(t) dt. \tag{7}$$

Det följer av satsen att $dW(t)dW(t) = dt$, vilket vi tolkar som att Wienerprocessen samlar en enhet kvadratisk variation per tidsenhet. Värt att notera är att den kvadratiska variationen beräknas väg för väg, och därför är beroende av vägen processen tar medan variansen beräknas som ett medelvärde över alla vägar [10].

2.2.2 Itô-Doeblin formeln

Eftersom Brownsk rörelse har nollskild kvadratisk variation kan vi inte derivera på samma sätt vi gör vi vanlig analys. Därför introducerar vi Itô-Doeblinformlen.

Sats 3 (Itô-Doeblin formeln [10]). *Låt* $f(t, X(t))$ *där* $f(t, x) \in C^{1,2}$ *vara en funktion där* $\{X(t)\}_{t \geq 0}$ *är en Itôprocess. Då är*

$$f(t, X(t)) = f(0, X(0)) + \int_0^t f_t(u, X(u)) du + \int_0^t f_x(u, X(u)) dX(u) + \frac{1}{2} \int_0^t f_{xx}(u, X(u)) dX(u) dX(u). \tag{8}$$

Om $W(t)$ är en Wienerprocess så vet vi att dess kvadratiska variation ges av $dW(t)dW(t) = dt$ och vi kan skriva (8) ovan på differentialformen

$$\begin{aligned} df(t, W(t)) &= f_t(t, W(t))dt + f_x(t, W(t))dW(t) + \frac{1}{2} f_{xx}(t, W(t))dW(t)dW(t) \\ &= \left(f_t(t, W(t)) + \frac{1}{2} f_{xx}(t, W(t)) \right) dt + f_x(t, W(t))dW(t) \end{aligned} \tag{9}$$

vilket är mer intuitivt att förstå. Här kan $df(t, W(t))$ ses som förändringen i $f(t, W(t))$ när t förändras "lite" dt, och $dW(t)$ för förändringen i Wienerprocessen när t förändras "lite" dt. Eftersom "lite" inte har någon matematisk mening så låter vi sats 3 ge upphov till dess differentialform (9) [10].

2.3 CKLS-processer

Chan-Karolyi-Longstaff-Sanders (CKLS)-processer, är en familj av SDE:er definerade av ekvationen

$$dX(t) = \beta(\alpha - X(t))dt + \sigma X(t)^{\gamma} dW(t). \tag{10}$$

där $\{\alpha, \beta, \sigma, \gamma\} \in \boldsymbol{\theta}$ är parametrar och $\{W(t)\}_{t \geq 0}$ en Wienerprocess [12]. Definitionen kan skrivas på många sätt men detta sätt bjuder in till intuitiva förklaringar av processens egenskaper. Parametern α kan ses som jämviktsnivån processen pendlar runt och β är processens vikt, alltså hur snabbt processen rör sig mot α. Den högra termen beskriver volatiliteten där parametern γ beskriver hur slumpens påverkan förändras beroende på värdet av $X(t)$ och σ beskriver storleken på variationen [12].

2.3.1 Ornstein-Uhlenbeck

Ornstein-Uhlenbeckprocessen (OU) fås genom ansättningen $\gamma = 0$ i CKLS-processen (10) ovan, vilket innebär att parametervektorn blir $\boldsymbol{\theta} = [\theta_0, \theta_1, \theta_2] = [\alpha, \beta, \sigma]$, och vi skriver

$$dX(t) = \beta(\alpha - X(t))dt + \sigma dW(t). \tag{11}$$

Genom att använda Itô-Dobelinformeln (9) kan det visas att den analytiska lösningen (se B.2) till ekvationen med begynnelsevillkor $X_0 = x_0$ är

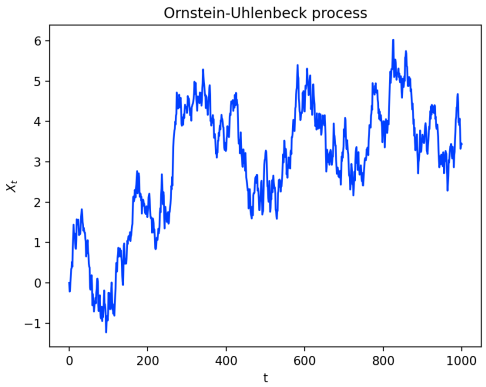
$$X(t) = \alpha + (x_0 - \alpha)e^{-\beta t} + \sigma \int_0^t e^{-\beta(t-s)} dW(s). \tag{12}$$

Existensen av en analytisk lösning till OU-processens begynnelsevärdesproblem är relativt unikt. De flesta stokastiska processer generellt kan inte lösas explicit och måste istället approximeras [13].

Ornstein-Uhlenbeckprocessen är den enda process som samtidigt besitter egenskaperna att vara en Markovprocess, en Gauss-process och stationär process [3]. Stationär betyder här att de ändligdimensionella fördelningarna inte påverkas av en förskjutning i tid.

2.4 Simulering av lösningar till SDE:er

Simuleringar av lösningar till stokastiska differentialekvationer är centrala för att undersöka ett systems dynamik när analytiska lösningar är svårtillgängliga eller inte existerar [13]. Svårigheter som uppkommer vid simulering av SDE:er inkluderar hanteringen av stokastiska termer, vilket resulterar i lösningar som är stokastiska processer istället för deterministiska funktioner. Numeriska metoder för att lösa en SDE kan vara känsliga för val av parametrar och begynnelsevillkor, vilket potentiellt kan leda till felaktiga eller instabila simuleringar [2]. Därför krävs lämpliga metoder och parametrar för att uppnå en tillförlitlig och korrekt simulering av lösningar till SDE:er.



Figur 1: Ornstein-Uhlenbeck process med $\alpha = 3$, $\beta = 1$ och $\sigma = 2$.

2.4.1 Euler-Maruyama-metoden

Euler-Murayama-metoden (EM) är en numerisk metod för att lösa SDE:er och metoden är en generalisering av Eulers stegmetod för ODE:er [14]. Den tar ett givet begynnelsevärdesproblem för en SDE och simulerar en diskretiserad Markovkedja som approximerar en lösningsbana till begynnelsevärdeproblemet (3). Låt X vara en stokastisk process enligt (3) formulerad som

$$dX(t) = \mu(t, X(t))dt + \sigma(t, X(t))dW(t) \tag{13}$$

med initialvillkor $X(0) = x_0$ som ska lösas på ett interall $[0, T]$ med tidsdiskretisering $0 = t_0 < t_1 < \ldots < t_N = T$ där $t_i - t_{i-1} = \Delta t = T/N$. EM-approximationen Y av den sanna processen X är definierad som den iterativt beräknade Markovkedjan

$$Y_{t_{i+1}} = Y_{t_i} + \mu(t_i, Y_{t_i})\Delta t + \sigma(t_i, Y_{t_i})\Delta W \tag{14}$$

för $0 \leq i \leq (N-1)$ där $Y_0 = X(0)$ och $\Delta W = W_{t_{i+1}} - W_{t_i}$ enligt definition 2 är oberoende av fördelningen $\mathcal{N}(0, \Delta t)$. EM-approximationen av övergångstätheten mellan två element i markovkedjan måste därför även den vara normalfördelad och kan därför skrivas som

$$\pi(Y_{t_{i+1}} | Y_{t_i} = y_{t_i}) \sim \mathcal{N}(y_{t_i} + \mu(t_i, y_i)\Delta t,\ \sigma(t_i, y_{t_i})^2 \Delta t). \tag{15}$$

EM delar samma svaghet som Eulers stegmetod vilket är att approximationen är bäst för små Δt, medan för större Δt, blir metoden instabil och mindre tillförlitlig [2]. Dessutom ackumuleras fel när tiden ökar, vilket leder till sämre approximationer för större t. Se appendix B.3 för mer om EM konvergens.

2.4.2 Dragning från övergångstätheter

Vill vi simulera en stokastisk process är det inte möjligt att representera den oändliga fraktalmässiga komplexiteten med hjälp av ett ändligt antal värden i en dator. Det bästa vi kan göra är att

simulera processen för diskreta tidpunkter $\{X_{t_0}, X_{t_1}, X_{t_2}, \ldots, X_{t_N} \mid i < j \implies t_i < t_j\}$. Detta kan enkelt göras då det finns en analytisk lösning för begynnelsevärdesproblemet (11). Eftersom den kan användas för ta fram övergångstätheten för X_t givet värdet på den tidigare tidpunken $X_s = x_s$ $(s < t)$ genom att sätta initialvärdet $x_0 = x_s$.

2.4.3 Simulering av Ornstein-Uhlenbeck

Eftersom varje inkrement av en Wienerprocess är normalfördelat $W_t - W_s \sim \mathcal{N}(0, t-s)$ $(s < t)$ har vi även att övergångstätheten för OU-processen X_t givet $X_s = x_s$ är normalfördelad med väntevärde och varians (se ekvation (36) i B.1)

$$\mathbb{E}(X_t | X_s = x_s, \boldsymbol{\theta}) = \alpha + (x_s + \alpha)e^{-\beta t} \quad \text{och} \quad \text{Var}(X_t | X_s = x_s, \boldsymbol{\theta}) = \frac{\sigma^2(1 - e^{-2\beta t})}{2\beta} \tag{16}$$

vilket ger övergångstätheten

$$\pi(X_t | X_s = x_s, \boldsymbol{\theta}) \sim \mathcal{N}\big(\alpha + (x_s - \alpha)e^{-\beta t},\ \frac{\sigma^2(1 - e^{-2\beta t})}{2\beta}\big) \tag{17}$$

som kan användas för att iterativt simulera processen [13] över diskreta tidpunkter över ett intervall $0 = t_0 < t_1 < \ldots < t_N = T$ genom att initiera startvillkoret $X_{t_0} = x_0$ och stegvis göra dragningar

$$X_{t_i} \sim \mathcal{N}\big(\alpha + (X_{t_{i-1}} - \alpha)e^{-\beta \Delta t},\ \frac{\sigma^2(1 - e^{-2\beta \Delta t})}{2\beta}\big) \tag{18}$$

där $\Delta t = t_i - t_{i-1},\ i \in [0, N-1]$. Alternativt kan EM användas för att approximera övergångstätheterna enligt (15). För en OU-process med startvillkor $X_{t_0} = x_0$ görs sedan dragningar enligt

$$X_{t_i} \sim \mathcal{N}(X_{t_{i-1}} + \beta(X_{t_{i-1}} - \alpha)\Delta t,\ \sigma^2 \Delta t). \tag{19}$$

2.5 Inferens av parametrar

Då vi nu har visat metoder för att simulera lösningar till SDE:er både approximativt med EM och genom att använda exakta övergångstätheter, kommer vi till nästa problem, att uppskatta parametrar $\boldsymbol{\theta}$ (som för OU-processen är $[\theta_0, \theta_1, \theta_2] = [\alpha, \beta, \sigma]$) så att simuleringar stämmer överens med någon observerad data $\boldsymbol{X}^{obs} = \{X_{t_0}, X_{t_1}, ..., X_{t_N}\}$.

2.5.1 Frekventistisk och bayesiansk statistik - en jämförelse

Det finns två olika paradigm inom statistik - det frekventistiska och det bayesianska [15]. Ett typiskt problem som behandlas inom det frekventistiska paradigmet är att data $\boldsymbol{X}^{obs}$ antas vara genererad från någon fördelning med likelihood-funktionen $L(\boldsymbol{X}^{obs}|\boldsymbol{\theta})$ där vi vill uppskatta $\boldsymbol{\theta}$ givet kända data $\boldsymbol{X}^{obs}$. Inom det frekventistiska paradigmet är alltså $\boldsymbol{\theta}$ ett tal som kan bestämmas med exempelvis metoden Maximum Likelihood Estimation (MLE) och för vilket vi kan beräkna ett konfidensintervall.

Inom det bayesianska paradigmet behandlas $\boldsymbol{\theta}$ istället som en slumpvariabel med a priorifördelning $\pi(\boldsymbol{\theta})$ [15]. För att generera data X från någon fördelning $L(\boldsymbol{X}^{obs}|\boldsymbol{\theta})$ genereras alltså först ett $\boldsymbol{\theta}$ från $\pi(\boldsymbol{\theta})$. Ett typiskt problem som kan behandlas inom det bayesianska paradigmet är att bestämma fördelningen av $\boldsymbol{\theta}$ givet någon data $\boldsymbol{X}^{obs}$ vilket alltså ger oss den så kallade a postieriorifördelningen $\pi(\boldsymbol{\theta}|\boldsymbol{X}^{obs})$. Värdet på $\boldsymbol{\theta}$ estimeras som väntevärdet [13]

$$\mathbb{E}(\boldsymbol{\theta}) = \int_{\boldsymbol{\theta}} \pi(\boldsymbol{\theta}|\boldsymbol{X}^{obs})\boldsymbol{\theta}\, d\boldsymbol{\theta} \tag{20}$$

där $\boldsymbol{\theta} \sim \pi(\boldsymbol{\theta}|\boldsymbol{X}^{obs})$ som definieras enligt Bayes formel

$$\pi(\boldsymbol{\theta}|\boldsymbol{X}^{obs}) = \frac{L(\boldsymbol{X}^{obs}|\boldsymbol{\theta})\pi(\boldsymbol{\theta})}{m(\boldsymbol{X}^{obs})} = \frac{L(\boldsymbol{X}^{obs}|\boldsymbol{\theta})\pi(\boldsymbol{\theta})}{\int_{\boldsymbol{\theta}} L(\boldsymbol{X}^{obs}|\boldsymbol{\theta})\pi(\boldsymbol{\theta})\, d\boldsymbol{\theta}}. \tag{21}$$

Densiteten i nämnaren, $m(\boldsymbol{X}^{obs}) = \int_{\boldsymbol{\theta}} L(\boldsymbol{X}^{obs}|\boldsymbol{\theta})\pi(\boldsymbol{\theta})\, d\boldsymbol{\theta}$, kallas *marginaldensiteten* av X [15]. Den är oftast svår att beräkna och inferensmetoder utformas därför så att man inte behöver det. $\pi(\boldsymbol{\theta})$ är priori-fördelningen av $\boldsymbol{\theta}$ vilket är den tidigare information eller gissning av vilka $\boldsymbol{\theta}$ som är möjliga. $L(\boldsymbol{X}^{obs}|\boldsymbol{\theta})$ är likelihood-funktionen som representerar sannolikheten att $\boldsymbol{X}^{obs} = \{x_{t_0}, x_{t_1}, ..., x_{t_N}\}$ är genererad av $\boldsymbol{\theta}$. I det diskreta fallet kan detta uttryckas som produkten

$$L(\boldsymbol{X}^{obs}|\boldsymbol{\theta}) = p(X_{t_0} = x_0) \prod_{i=1}^{N} f(X_{t_i}|\boldsymbol{\theta}) \tag{22}$$

där $f(X_{t_i}|\boldsymbol{\theta})$ är täthetsfunktionen för X_{t_i} och $p(X_{t_0} = x_0)$ är sannolikheten att startvärdet är x_0, denna term kan försummas om N är stort [13]. I fallet för Ornstein-Uhlenbeck, vars övergångstätheter är normalfördelade, är täthetsfunktionen normalfördelningens täthetsfunktion

$$f(X_{t_i} = x_i, \mu, \sigma^2|\boldsymbol{\theta}) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x_{t_i}-\mu)^2}{4\sigma^2}} \tag{23}$$

där väntevärdet μ och variansen σ^2 (notera att detta inte är parametern $\sigma \in \boldsymbol{\theta}$) är definierade enligt

$$\mu(X_{t_i}) = \begin{cases} X_{t_{i-1}} e^{-\beta\Delta t} + \alpha(1 - e^{-\beta\Delta t}) & \text{för exakt lösning,} \\ X_{t_{i-1}} + \beta(\alpha - X_{t_{i-1}})\Delta t & \text{för EM-approximation.} \end{cases} \tag{24}$$

$$\mathrm{Var}(X_{t_i}) = \begin{cases} \frac{\sigma^2}{2\beta}(1 - e^{-2\beta\Delta t}) & \text{för exakt lösning,} \\ \sigma^2\Delta t & \text{för EM-approximation.} \end{cases} \tag{25}$$

2.6 Markov Chain Monte Carlo

För att utföra inferens på någon given data vill vi kunna göra dragningar från a posteriorifördelningen $\pi(\boldsymbol{\theta}|\boldsymbol{X}^{obs})$ från ekvation (21), något som är utmanande då den sällan kan beskrivas analytiskt [16]. En lösning på detta problem är att använda MCMC-metoder utan att behöva uttrycka $\pi(\boldsymbol{\theta}|\boldsymbol{X}^{obs})$ explicit. Idén bakom dessa metoder är att behandla $\boldsymbol{\theta}^i$ som en stokastisk variabel där $i \in I$ och I är antalet iterationer som algoritmen körs. Därefter konstrueras en Markovkedja som, givet att antalet dragningar är tillräckligt stort, drar ur en målfördelning – i detta fall $\pi(\boldsymbol{\theta}|\boldsymbol{X}^{obs})$ [17]. En rigorös beskrivning av MCMC-algoritmer är utanför ramen av detta projekt och vi kommer översiktligt beskriva den metod som används i rapporten, Metropolis-Hastings algoritm, vilket är en av de mest etablerade MCMC-metoderna [18].

2.6.1 Monte Carlo-metoder

Monte Carlo-metoder är numeriska tekniker som löser statistiska och matematiska problem genom att generera stokastiska prover från en sannolikhetsfördelning [19]. De används ofta när problem har många dimensioner eller när analytiska lösningar är svåra att hitta.

En vanlig uppgift för en Monte Carlo-metod är att uppskatta väntevärden $\mathbb{E}[f(\boldsymbol{\theta})]$ med stokastiska variabler $\boldsymbol{\theta}$ och en fördelning $\pi(\boldsymbol{\theta})$. Genom att dra N oberoende prover $\boldsymbol{\theta}^i$ från $\pi(\boldsymbol{\theta})$, kan vi uppskatta $\mathbb{E}[f(\boldsymbol{\theta})]$ enligt [5]

$$\mathbb{E}[f(\boldsymbol{\theta})] \approx \frac{1}{N} \sum_{i=1}^{N} f(\boldsymbol{\theta}^i). \tag{26}$$

Monte Carlo-metoder används inom statistisk inferens, optimering och numerisk integrering. I MCMC-metoder genereras prover från en posteriorfördelning för att uppskatta parametrars medelvärden och osäkerheter [5].

2.6.2 Metropolis-Hastings algoritm (MH)

För att påminna om kontexten: vi vill göra dragningar från fördelningen

$$\pi(\boldsymbol{\theta}|\boldsymbol{X}^{obs}) \propto L(\boldsymbol{X}^{obs}|\boldsymbol{\theta})\pi(\boldsymbol{\theta}). \tag{27}$$

Idén med metoden är att genom en kandidatfördelning $g(\boldsymbol{\theta}^*|\boldsymbol{\theta}^i)$ generera kandidater $\boldsymbol{\theta}^*$ som potentiellt accepteras som nästa $\boldsymbol{\theta}^i$. Denna kandidatfördelning genererar $\boldsymbol{\theta}^*$ från en fördelning som baseras på det senast accepterade $\boldsymbol{\theta}^i$, vilket är en Markovegenskap. Anledningen till detta är att det är mer sannolikt att acceptera en ny kandidat nära det nuvarande $\boldsymbol{\theta}^i$. Inledningsvis väljs ett startvärde $\boldsymbol{\theta}^0$ inom priorfördelningen $\pi(\boldsymbol{\theta}^i)$. För enkelhetens skull används en likformig sannolikhetsfördelning över det intervall där $\boldsymbol{\theta}$ förväntas vara. Vårt mål är att acceptera $\boldsymbol{\theta}^*$ som $\boldsymbol{\theta}^{i+1}$ med en beslutsregel som, efter tillräckligt många iterationer, leder till att alla accepterade $\boldsymbol{\theta}^i$ kommer från fördelningen $\pi(\boldsymbol{\theta}|\boldsymbol{X}^{obs})$. Denna beslutsregel är att acceptera ett steg från $\boldsymbol{\theta}^i$ till $\boldsymbol{\theta}^{i+1} = \boldsymbol{\theta}^*$ med sannolikheten

$$\alpha(\boldsymbol{\theta}^*|\boldsymbol{\theta}^i) = \min\left(\frac{L(\boldsymbol{X}^{obs}|\boldsymbol{\theta}^*)\pi(\boldsymbol{\theta}^*)g(\boldsymbol{\theta}^i|\boldsymbol{\theta}^*)}{L(\boldsymbol{X}^{obs}|\boldsymbol{\theta}^i)\pi(\boldsymbol{\theta}^i)g(\boldsymbol{\theta}^*|\boldsymbol{\theta}^i)},\ 1\right) \in [0,1]. \tag{28}$$

Notera kvoten i denna sannolikhet är

$$\frac{\pi(\boldsymbol{\theta}^*|\boldsymbol{X}^{obs})g(\boldsymbol{\theta}^i|\boldsymbol{\theta}^*)}{\pi(\boldsymbol{\theta}^i|\boldsymbol{X}^{obs})g(\boldsymbol{\theta}^*|\boldsymbol{\theta}^i)}$$

där $m(\boldsymbol{X}^{\boldsymbol{obs}})$ kortas bort. Denna beslutsregel baseras på det faktum att den genererade Markovkedjan skall vara reversibel. För mer detaljerad förklaring se Chib och Greenberg (1995) [20].

Metropolis-Hastings algoritm [21]:

---

1. Initiera markovkedjan med något startvärde $\boldsymbol{\theta}^0$ och välj någon prior fördelning $\pi(\boldsymbol{\theta})$ för $\boldsymbol{\theta}$ samt sätt $i = 0$
2. Föreslå $\boldsymbol{\theta}^*$ genom dragning av från någon kandidat-fördelning $g(\boldsymbol{\theta}^*|\boldsymbol{\theta}^i)$ som ett möjligt nästa steg $\boldsymbol{\theta}^{i+1}$ i kedjan.
3. Beräkna likelihood-funktionerna $L(\boldsymbol{X}^{obs}|\boldsymbol{\theta}^*)$ och $L(\boldsymbol{X}^{obs}|\boldsymbol{\theta})$.
4. Acceptera $\boldsymbol{\theta}^*$ som $\boldsymbol{\theta}^{i+1}$ med sannolikheten

$$\alpha(\boldsymbol{\theta}^*|\boldsymbol{\theta}^i) = \min\left(\frac{L(\boldsymbol{X}^{obs}|\boldsymbol{\theta}^*)\pi(\boldsymbol{\theta}^*)g(\boldsymbol{\theta}^i|\boldsymbol{\theta}^*)}{L(\boldsymbol{X}^{obs}|\boldsymbol{\theta}^i)\pi(\boldsymbol{\theta}^i)g(\boldsymbol{\theta}^*|\boldsymbol{\theta}^i)},\ 1\right)$$

   annars förkasta det föreslagna $\boldsymbol{\theta}^*$ och sätt $\boldsymbol{\theta}^{i+1} = \boldsymbol{\theta}^i$.
5. Uppdatera $i = i + 1$ och återvänd till 2.

Notera att om kandidatfördelningen $g(\cdot|\cdot)$ är symmetrisk är $g(a|b) = g(b|a)$, för alla a, b vilket förenklar sannolikheten i steg 4. För att försäkra sig om att alla $\boldsymbol{\theta^i} \sim \pi(\boldsymbol{\theta}|\boldsymbol{X}^{obs})$ förkastas den första delen av kedjan innan konvergens har uppnåts. Denna mängd brukar kallas *burn-in* och varierar beroende på hur bra gissning $\boldsymbol{\theta}^0$ är.

2.6.3 Adaptiv Metropolis-Hastings

Valet av en effektiv kandidatfördelning för MH-algoritmen är av stor vikt för att uppnå tillfredsställande resultat är svårt att göra med en okänd målfördelning. Ett effektivt sätt att välja är att successivt anpassa kandidatfördelningen genom att utnyttja information om alla tidigare tillstånd i processen. Detta kallas Adaptiv Metropolis-Hastings. Nedan sammanfattas en variant föreslagen av Haarop, Saksman och Tammien (2001) [22].

Som kandidatfördelning $g(\boldsymbol{\theta}^*|\boldsymbol{\theta}^i)$ väljs en flerdimensionell normalfördelning med medelvärde i den senast accepterade punkten $\boldsymbol{\theta}^i$ och kovarians $\boldsymbol{C}_i(\boldsymbol{\theta}^0, ..., \boldsymbol{\theta}^{i-1})$. Efter ett visst antal iterationer I tillåts kovariansen att bero på alla tidigare värden i Markovkedjan genom att sätta $\boldsymbol{C}_i = s_d\text{Cov}(\boldsymbol{\theta}^0, ..., \boldsymbol{\theta}^{i-1}) + s_d\epsilon_d\boldsymbol{I}_d$. Parametern s_d beror på dimensionen d av $\boldsymbol{\theta}$ och $\epsilon_d > 0$, vilket är en vald konstant som är liten i jämförelse med storleken på mängden S ($\boldsymbol{\theta} \subset S$). $\boldsymbol{I}_d$ är den d-dimensionella identitetsmatrisen. Innan iteration I har passerats används en godtycklig initial kovarians $\boldsymbol{C}_0$ som är strikt positivt definit. Alltså definieras $\boldsymbol{C}_i$ enligt

$$\boldsymbol{C}_i = \begin{cases} \boldsymbol{C}_0, & \text{för } i \leq I, \\ s_d\text{Cov}(\boldsymbol{\theta}^0, ..., \boldsymbol{\theta}^{i-1}) + s_d\epsilon_d\boldsymbol{I}_d, & \text{för } i > I. \end{cases} \tag{29}$$

Valet av I beror mycket på hur bra det initiala valet $\boldsymbol{C}_0$ är. Rollen ϵ_d har är att försäkra att kovariansmatrisen har en nollskild determinant. Valet av $s_d = \frac{2.38^2}{d}$ rekommenderas av Gelman, Roberts och Gilks [23] för att uppnå en acceptanskvot i närheten av 0.234.

2.7 Approximate Bayesian Computation (ABC)

ABC och MCMC är båda samlingar av Bayesianska metoder som används för inferens om a posteriori-fördelningen $\pi(\boldsymbol{\theta}|\boldsymbol{X}^{obs})$, när direkt sampling från fördelningen inte är möjlig. En viktig distinktion mellan dessa metoder är att ABC inte kräver en explicit likelihood-funktion $L(\boldsymbol{X}^{obs}|\boldsymbol{\theta})$, vilket innebär att ABC kan hantera en betydligt bredare uppsättning processer. Denna rapport tar upp två av de mest populära ABC-algoritmerna: rejection sampler (ABC-R) och SMC (Sequential Monte Carlo).

2.7.1 ABC-Rejection

ABC-R går ut på att generera värden på parametrarna från a priori-fördelningen $\pi(\boldsymbol{\theta})$, och behålla dessa som dragningar från a posteriori-fördelningen $\pi(\boldsymbol{\theta}|\boldsymbol{X}^{obs})$ om de de kan generera data $\boldsymbol{X}^*$ som stämmer överens med den observerade datan $\boldsymbol{X}^{obs}$. Sannolikheten för att den genererade datan precis följer den observerade är emellertid nästan lika med noll, och en approximation görs där parametrar som ger upphov till data som är tillräckligt nära behålls. Skillnaden mellan datamängderna beräknas med en avståndsfunktion $\rho(\boldsymbol{X}^{obs}, \boldsymbol{X}^*)$, och tillräckligt nära definieras som att avståndet är mindre än en förbestämd toleransparameter ϵ. Därmed kommer de dragningar som görs inte att härröra från fördelningen $\pi(\boldsymbol{\theta}|\boldsymbol{X}^{obs})$, utan istället från $\pi_\epsilon(\boldsymbol{\theta} \mid \rho(\boldsymbol{X}^{obs}, \boldsymbol{X}^*) < \epsilon)$, en approximerad a priori-fördelning.

ABC-R algoritm [21]:

---

1. Generera en parametervektor $\boldsymbol{\theta}^* \sim \pi(\boldsymbol{\theta})$
2. Simulera data $\boldsymbol{X}^*$ med EM (14).
3. Beräkna avståndet mellan simulerad och observerad data utifrån någon avståndsfunktion: $\rho(\boldsymbol{X}^{obs}, \boldsymbol{X}^*)$
4. Acceptera $\boldsymbol{\theta}^*$ om $\rho(\boldsymbol{X}^{obs}, \boldsymbol{X}^*) < \epsilon$.
5. Börja om från 1.

2.7.2 ABC Sequantial Monte Carlo-algoritm

Ett problem med ABC-R är att acceptanskvoten för $\boldsymbol{\theta}^*$ ofta är väldigt låg. ABC-SMC är en metod som löser detta problem genom att ha en dynamisk toleransparameter ϵ. Inledningsvis sätts ett högt värde på ϵ, som sedan gradvis minskas för att se till att dragningarna är en bra approximation på dragningar från a posteriori-fördelningen. Vid varje runda t genereras en grupp kandidat-parametrar (partiklar) som sedan används som bas för dragningarna i nästa steg. Parametrarna som dras från tidigare population tilldelas en vikt w_{t-1} och en störning. ABC-SMC [21]:

---

1. Skapa $\epsilon_0 > 0$ och sätt $t = 0$.
2. Sätt partikelindikatorn $i = 1$.
3. (a) Om $t = 0$: generera parametervektor $\boldsymbol{\theta}^{**} \sim \pi(\boldsymbol{\theta})$
   (b) Annars: Dra parametervektor $\boldsymbol{\theta}^*$ från tidigare population $\{\boldsymbol{\theta}^i_{t-1}\}$ med vikter w_{t-1} och störning för att få $\boldsymbol{\theta}^{**} \sim K_t(\boldsymbol{\theta}|\boldsymbol{\theta}^*)$
   (c) Om $\pi(\boldsymbol{\theta}^{**}) = 0$, återvänd till 3.
   (d) Simulera data $\boldsymbol{X}^* \sim \pi(\boldsymbol{X}|\boldsymbol{\theta}^{**})$.
   (e) Om $\delta^*_t = \rho(\boldsymbol{X}, \boldsymbol{X}^*) \geq \epsilon_t$, återvänd till 3.
4. Sätt $\boldsymbol{\theta}^{(i)}_t = \boldsymbol{\theta}^{**}$, $\delta^{(i)}_t = \delta^*_t$ och beräkna vikterna $\tilde{w}^{(i)}_t$ för partikel $\boldsymbol{\theta}^{(i)}_t$, där

$$\tilde{w}^{(i)}_t = \begin{cases} 1 & \text{om } t = 0, \\ \frac{\pi(\boldsymbol{\theta}^{(i)}_t)}{\sum_{j=1}^N w^{(j)}_{t-1} K_t(\boldsymbol{\theta}^{(j)}_{t-1}, \boldsymbol{\theta}^{(i)}_t)} & \text{om } t > 0 \end{cases}.$$

Om $i < N$, sätt $i = i + 1$ och gå till 3.

5. Normalisera vikterna $w_t^i = \tilde{w}_t^{(i)} / \sum_i \tilde{w}_t^{(i)}$, och välj ϵ_{t+1} som q-percentilen av samlingen avstånd $\delta_t^{(i)}$, för något förbestämt $0 < q < 100$.
6. (Alternativt steg) Om vi uppfyller ett visst stoppkriterium så hoppar vi över nästa steg och metoden avslutas.
7. Om $t < T$, sätt $t = t + 1$ och gå till 2.

K_t i algoritmen är en så kallad *Transition kernel*. Den har samma syfte som kandidat-fördelningen i MH, det vill säga att den specificerar en fördelning för en variabel som adderas till (stör) varje kandidatparameter. Ett exempel på ett stoppkriterium för steg 6 är att avbryta om acceptanskvoten, andelen $\boldsymbol{\theta}^{**}$ som blir accepterade under en runda, är under 1.5%.

2.7.3 Neurala Nätverk som deskriptiv statistik

För att en ABC-metod ska fungera effektivt är det avgörande att kunna jämföra om simulerad data $\boldsymbol{X}^*$ från föreslagna $\boldsymbol{\theta}^*$ är tillräckligt lik observerad data $\boldsymbol{X}^{obs}$. Detta blir särskilt utmanande när den observerade datan är hög-dimensionell, ett fenomen som ofta kallas *the curse of dimensionality* [24]. En lösning på detta problem är att använda deskriptiv statistik av datan, $S(\boldsymbol{X})$, för att utvärdera $\rho(S(\boldsymbol{X}^{obs}), S(\boldsymbol{X}^*))$. I ABC-metoder innebär detta att dragningar görs från fördelningen $\pi_\epsilon(\boldsymbol{\theta} \mid \rho(S(\boldsymbol{X}^{obs}), S(\boldsymbol{X}^*)) < \epsilon)$ istället för $\pi(\boldsymbol{\theta}|\boldsymbol{X}^{obs})$, eftersom $S(\boldsymbol{X}^{obs})$ inte kan sammanfatta $\boldsymbol{X}^{obs}$ exakt med avseende på $\boldsymbol{\theta}$ och $\epsilon > 0$. För tillräckligt små ϵ görs dock approximationer av dragningar från $\pi_\epsilon(\boldsymbol{\theta}|S(\boldsymbol{X}^{obs}))$.

Neurala nätverk kan användas för att automatiskt konstruera deskriptiv statistik för ABC. Wiqvist et al. (2019) [25] har introducerat en arkitektur för neurala nätverk kallat Partially Exchangable Network (PEN) som är specifikt utvecklad för att hantera markoviansk data. PEN nätverket använder en regressionsmodell för att träna nätverket att hitta medelvärdet av a posteori-fördelningen $\mathbb{E}(\boldsymbol{\theta}|\boldsymbol{X})$ enligt

$$\boldsymbol{\theta^i} = \mathbb{E}(\boldsymbol{\theta}|\boldsymbol{X^i}) + \xi^i = \rho\beta_\rho \left(X^i_{1:d}, \sum_{l=1}^{M-d} \phi\beta_\phi(X^i_{l:l+d}) \right) + \xi^i \quad i \in T \tag{30}$$

där T är mängden träningsomgångar för nätverket, ξ^i någon störning med medelvärde 0, ϕ kallas det inre nätverket och avbildar en sekvens $X_{l:l+d}$ av längd d till någon representation $\phi(X_{l:l+d})$, ρ är det yttre nätverket som avbildar de första d värdena av in-datan $\boldsymbol{X}^i$ och summan av representationerna av längd d till $\mathbb{E}(\boldsymbol{\theta}|\boldsymbol{X})$. Vi har vikter β_ϕ och $\beta\rho$ för det inre respektive yttre nätverket och M representerar antalet datapunkter i $\boldsymbol{X}^i$. Vidare förklaring av nätverket är utom ramarna för rapporten, se Wiqvist et al. (2019) [25] för fördjupning.

2.8 Wasserstein-avstånd

Avstånd mellan fördelningar kan definieras på olika sätt, ett sådant som härstammar från transportteori är p-Wasserstein-avstånd.

Definition 6 (p-Wasserstein-avstånd [26])**.** *Låt P och Q vara fördelningar definierade på $\mathbb{R}^d$. Låt $\mathcal{J}(P, Q)$ vara mängden möjliga gemensamma fördelningar J av mängderna $X, Y \subset \mathbb{R}^d$ vars individuella fördelningar är P och Q. Då definieras p-Wasserstein-avståndet W_p mellan P och Q som transportproblemet*

$$W_p(P, Q) = \left(\inf_{J \in \mathcal{J}(P,Q)} \int |x - y|^p \, dJ(x, y) \right)^{1/p}, \tag{31}$$

där minimeraren J^* kallas den optimala transportplanen mellan P och Q. W_p är ett singulärt mått som representerar summan av den "massa" som måste förflyttas från P för att skapa Q. För det specifika fallet $p = 1$ kallas Wasserstein-avståndet även för "Earth Mover's Distance".

3 Metod

Programmeringsspråket Python användes för att undersöka inferensmetoderna med utgångspunkt i frågeställningen och för att visualisera resultatet. Se appendix B för fullständig källkod. Datamängderna som parameterinferens utfördes på var realiseringar av OU som genererats med övergångstätheten i ekvation (17). Tre datamängder simulerades med varierande storlek på tidssteg. Modell- och simuleringsparametrarna kan ses nedan i tabell 1. Se appendix B.4 för visualisering av realiseringarna.

Tabell 1: Beskrivning av simulerade datamängder där Δt är storleken på tidsstegen, $\boldsymbol{\theta} = [\alpha, \beta, \sigma]$ parametrarna för OU och N_{obs} är antalet datapunkter.

Datamängd	Δt	$\boldsymbol{\theta}$	N_{obs}
$\mathcal{D}_{\text{I}}$	0.1	[3, 1, 1]	100
$\mathcal{D}_{\text{II}}$	0.3	[3, 1, 1]	100
$\mathcal{D}_{\text{III}}$	1.0	[3, 1, 1]	100

På dessa datamängder utfördes inferens med MH-exakt, MH-EM, ABC-R och ABC-SMC algoritmerna för att generera a posteriori-fördelningar. Fördelningarna genererade med MH-exakt betraktades som sanna och användes för att jämföra de andra fördelningarna.

Implementeringen av ABC-R, ABC-SMC, samt träningen av det neurala nätverket, krävde simuleringar av OU vilket gjordes med EM trots att de exakta övergångstätheterna hade kunnat användas istället. Detta var för att demonstrera att ABC-metoderna verkligen är helt likelihood-fria och alltså inte kräver tillgång till de exakta övergångstätheterna. För att säkerställa stabiliteten i de approximativa EM-simuleringarna användes ett finare tidssteg, oberoende av datamängdens tidssteg. Storleken på tidssteg valdes till $\Delta t_{sim} = 0.01$ och antalet datapunkter till $N_{sim} = N_{\mathcal{D}} \cdot \Delta t_{\mathcal{D}} / \Delta t_{sim}$ där $N_{\mathcal{D}}$ och $\Delta t_{\mathcal{D}}$ är antalet datapunkter respektive storlek på tidssteg för given datamängd. Från de simulerade datamängderna $\boldsymbol{X}^i_{sim}$ plockades sedan datapunkter med steglängd $\Delta t_{\mathcal{D}} = N_{sim}/N_{\mathcal{D}}$ ut för att få simulerade banor med $N_{\mathcal{D}}$ datapunkter och tidssteg $\Delta t_{\mathcal{D}}$.

3.1 Implementering av MH

MH-exakt och MH-EM konstruerades enligt 2.6.2 med startvärde $\boldsymbol{\theta}^0 = [1,\ 0.5,\ 0.5]$. $\pi(\theta_j)$ sattes till den likformiga sannolikhetsfördelningen $\mathcal{U}_{[0,10]}$ för samtliga $j = 0, 1, 2$. Kandidatfördelningen $g(\boldsymbol{\theta}^*|\boldsymbol{\theta}^i)$ konstruerades enligt 2.6.3 som en adaptiv, multivariat normalfördelning med initial kovariansmatris

$$\boldsymbol{C}_0 = \begin{bmatrix} 0.1 & 0 & 0 \\ 0 & 0.1 & 0 \\ 0 & 0 & 0.1 \end{bmatrix}$$

och $\epsilon_d = 1 \cdot 10^{-8}$. För båda metoderna beräknades likelihood-funktionerna $L(\boldsymbol{X}^{obs}|\boldsymbol{\theta}^*)$ och $L(\boldsymbol{X}^{obs}|\boldsymbol{\theta}^i)$ som enligt ekvation (22) men för MH-exakt användes de exakta uttrycken för väntevärde och varians i ekvation (24) och (25) medan för MH-EM användes de approximativa motsvarigheterna.

Eftersom kandidatfördelningen $g(\boldsymbol{\theta}^*|\boldsymbol{\theta}^i)$ är symmetrisk reduceras sannolikhetsuttrycket i ekvation (28) till

$$\alpha(\boldsymbol{\theta}^*|\boldsymbol{\theta}^i) = \min\left(\frac{L(\boldsymbol{X}^{obs}|\boldsymbol{\theta}^*)\pi(\boldsymbol{\theta}^*)}{L(\boldsymbol{X}^{obs}|\boldsymbol{\theta}^i)\pi(\boldsymbol{\theta}^i)},\ 1\right).$$

Algoritmerna kördes i 100000 rundor och burn-in valdes till 60000.

3.2 Implementering av ABC-R

ABC-R utformades enligt 2.7.1. A priori-fördelningen $\pi(\boldsymbol{\theta})$ konstruerades genom att först låta det neurala nätverket uppskatta $\boldsymbol{\theta}$ och sedan bygga en uniform a priori-fördelning runt punktskattningen. Givet en observation $\boldsymbol{X}^{obs}$ uppskattades $\boldsymbol{\theta}$ som $\hat{\boldsymbol{\theta}} = S(\boldsymbol{X}^{obs})$ och $\pi(\boldsymbol{\theta})$ konstrueras då elementvis med

$$\pi(\theta_j) = \begin{cases} \mathcal{U}_{[\hat{\theta}_j - 3\ \hat{\theta}_j + 3]} & \text{för j} = 0, \\ \mathcal{U}_{[0,\ \hat{\theta}_j + 2]} & \text{för j} = 1, \\ \mathcal{U}_{[0,\ \hat{\theta}_j + 1]} & \text{för j} = 2. \end{cases}$$

EM-metoden användes sedan för att simulera datan med de dragna parametrarna från a priori-fördelningen och som avståndsfunktion $\rho(S(\boldsymbol{X}^{obs}), S(\boldsymbol{X}^*))$ användes euklidiskt avstånd. För varje datamängd genererades a posteriori-fördelningar med varje ϵ i listan [0.1, 0.4, 0.7].

3.3 Implementering av ABC-SMC

ABC-SMC utformades enligt 2.7.2. I undersökningen av hur konfigurationen av ABC-SMC påverkar resultaten genererades 5 a posteriori-fördelningar för respektive värde på N i listan $[100, 250, 500, 750, 1000]$, med konstant $q = 30\%$. För undersökning av påverkan av q sattes $N = 1000$ och 5 a posteriori-fördelningar genererades för varje q i listan [10%, 20%, 30%, 40%, 50%]. Båda ovanstående experiment upprepades för samtliga datamängder, med $T = 100$ och stoppkriterium: acceptanskvot $< 1.5\%$.

Slutligen genererades 10 a posteriori-fördelningar för varje datamängd med konfigurationen: $N = 750$, $q = 0.3$ och stoppkriterium: acceptanskvot $< 1.5\%$.

3.4 Konfiguration och träning av det neurala nätverket

Nätverket som användes för att generera deskriptiv statistik är av typen PEN, som beskrivs i 2.7.3. Se appendix B för inre konfiguration av nätverket..

För att träna ett neuralt nätverk att agera som $S(\cdot)$ och för att kunna uppskatta $\mathbb{E}(\boldsymbol{\theta}|\boldsymbol{X})$ simulerades en uppsättning av $k = 100000$ par av banor och parametrar $(\boldsymbol{X}^i, \boldsymbol{\theta}^i), i \in [1, k]$ så att det neurala nätverket fick $\boldsymbol{X}^i$ som input och det rätta svaret skulle vara $\boldsymbol{\theta}^i$. Av dessa 100000 par av data användes 80% till träning av nätverket och resten användes till validering av nätverket för att kontrollera att den lärt sig att generalisera. Dessa datapar $(\boldsymbol{X}^i, \boldsymbol{\theta}^i)$ simulerades genom att först slumpa parametrarna uniformt från $\theta_0 \sim \mathcal{U}^i_{[0,10]}$, $\theta_1 \sim \mathcal{U}^i_{[0,5]}$ och $\theta_2 \sim \mathcal{U}^i_{[0,2]}$. Banan $\boldsymbol{X}^i$ simulerades sen med EM (19) som en realisering av OU med parametrar $\boldsymbol{\theta}^i$ och en förfinad diskretisering användes för att öka noggrannheten såsom beskrivs ovan i 3. Grafer som visar prestandan av de neurala nätverken finns i appendix B.5.

3.5 Mått på noggrannhet och effektivitet

För att avgöra hur bra en fördelning genererad av de olika inferensmetoderna är jämfördes den med den fördelning som genererats från den analytiskt beräknade övergångstätheten med MH-exakt. p-Wasserstein-avståndet W_p användes för att jämföra de genererade a posteriori-fördelningarna enligt (31) med $p = 2$. Wasserstein måttet valdes framför andra avståndsmått som Hellinger eller L_2 på grund av att det har visats bättre sammanfatta skillnader mellan fördelningar på ett representativt sätt [26] samt att den sammanfattar den multivariata fördelningen av $\boldsymbol{\theta}$ istället för de marginella fördelningarna för varje enskild parameter.

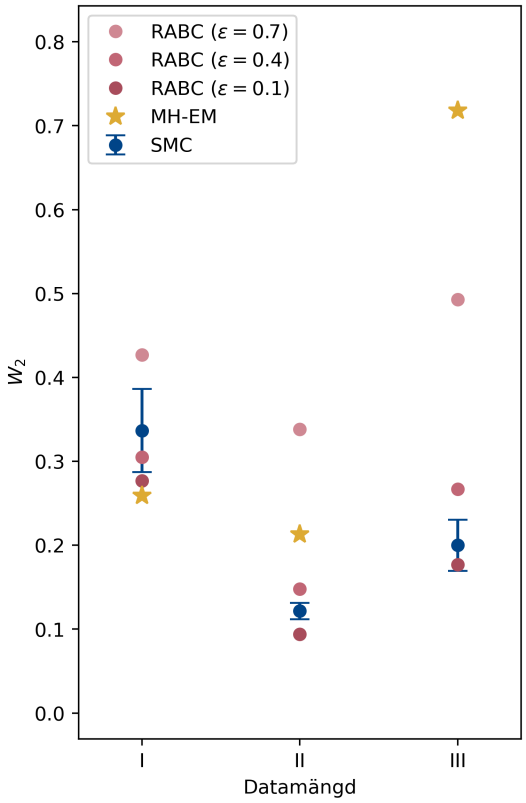
För att mäta effektivitet mättes också tidsåtgången för att köra algoritmerna. Men parameterinferens med MH-algoritmerna är mer eller mindre ögonblickligt vilket utklassar ABC-metoderna i termer av effektivitet. Därför jämfördes endast den relativa tidsåtgången för ABC-R mot ABC-SMC. När det kommer till tidsjämförelser av implementeringar av algoritmer finns flera utomstående parametrar som kan påverka tidsåtgången för ett program, såsom vilken maskin programmet körs på eller vilka andra saker maskinen gör i bakgrunden. Tidsåtgången för metoderna mättes därför på samma dator så att den relativa tidsåtgången mellan metoderna fortfarande kunde kvantifieras. Datorn som användes hade en 2,3 GHz Dual-Core Intel Core i5 CPU och 8 GB RAM-minne.

4 Resultat

I resultatavsnittet redovisas jämförelsen av MH-EM och ABC-metodernas noggrannhet samt ABC-metodernas effektivitet, mätt som tidsåtgång, för olika tidssteg. Även undersökningen av ABC-metodernas konfiguration redovisas i form av hur tolerensen ϵ påverkar ABC-R samt hur antal partiklar N och val av tolerenskvantil q påverkar ABC-SMC.

4.1 MH-EM och ABC-metodernas noggrannhet för olika tidssteg Δt

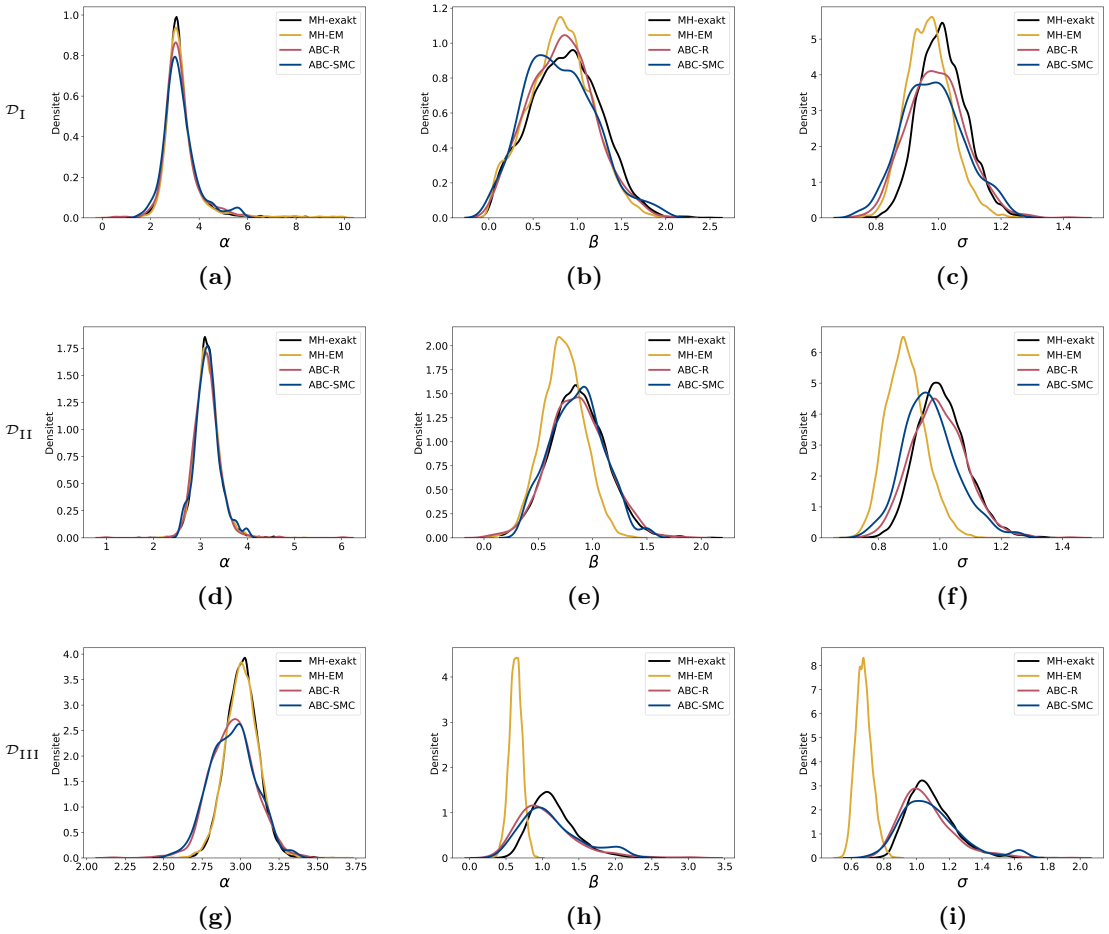
I figur 3 visas a posteriori-fördelningar $\pi_\epsilon(\boldsymbol{\theta}|S(\boldsymbol{X}^{obs}))$ genererade med ABC-R ($\epsilon = 0.1$) samt ABC-SMC ($N = 1000$) och $\pi(\boldsymbol{\theta}|\boldsymbol{X}^{obs})$ genererade med MH-Exakt och MH-EM för $\mathcal{D}_{\text{I}}$, $\mathcal{D}_{\text{II}}$ och $\mathcal{D}_{\text{III}}$. Här syns trenden att när Δt blir större genererar MH-EM a posteriori-fördelningar med allt större systematiskt fel - framförallt för volatilitets-parametern σ men även för β som representerar hastigheten för återgången till medelvärdet. ABC-R och ABC-SMC bibehåller däremot god noggrannhet för samtliga datamängder. I figur 2 visualiseras trenden genom att de jämförda metodernas a posteriori-fördelningars avvikelse från den den sanna fördelningen genererad med MH-exakt ritas ut. Avvikelsen kvantifieras med W_2-avståndet mellan fördelningarna och värdena kan avläsas i tabell 2. Tabellvärdena visar att i noggrannhet överträffade ABC-R med $\epsilon = 0.1$ ABC-SMC för alla datamängder. För $\mathcal{D}_{\text{I}}$ gav även ABC-R med $\epsilon = 0.4$ något högre noggrannhet än ABC-SMC. I figur 2 ses också hur W_2-avståndet mellan MH-EM-fördelningen och den sanna fördelningen ökar då tidssteget Δt växer; särskilt tydligt är detta för $\mathcal{D}_{\text{III}}$ där ABC-R till och med presterar bättre än MH-EM med den högsta toleransen $\epsilon = 0.7$.



Figur 2: W_2-avstånd för a posteriori-fördelningar genererade med ABC-R, ABC-SMC och MH-EM för $\mathcal{D}_{\text{I}}$, $\mathcal{D}_{\text{II}}$ och $\mathcal{D}_{\text{III}}$. Värden för ABC-SMC är medelvärden över tio körningar.

Tabell 2: W_2-avstånd för a posteriori-fördelningar genererade med ABC-R, ABC-SMC och MH-EM för $\mathcal{D}_{\text{I}}$, $\mathcal{D}_{\text{II}}$ och $\mathcal{D}_{\text{III}}$. Värden för ABC-SMC är medelvärden över tio körningar.

	W_2 $\mathcal{D}_{\text{I}}$	W_2 $\mathcal{D}_{\text{II}}$	W_2 $\mathcal{D}_{\text{III}}$
MH-EM	0.259	0.213	0.718
ABC-$R_{\epsilon=0.1}$	0.277	0.094	0.177
ABC-$R_{\epsilon=0.4}$	0.305	0.148	0.267
ABC-$R_{\epsilon=0.7}$	0.427	0.338	0.493
ABC-SMC	0.336 ± 0.050	0.122 ± 0.010	0.200 ± 0.030



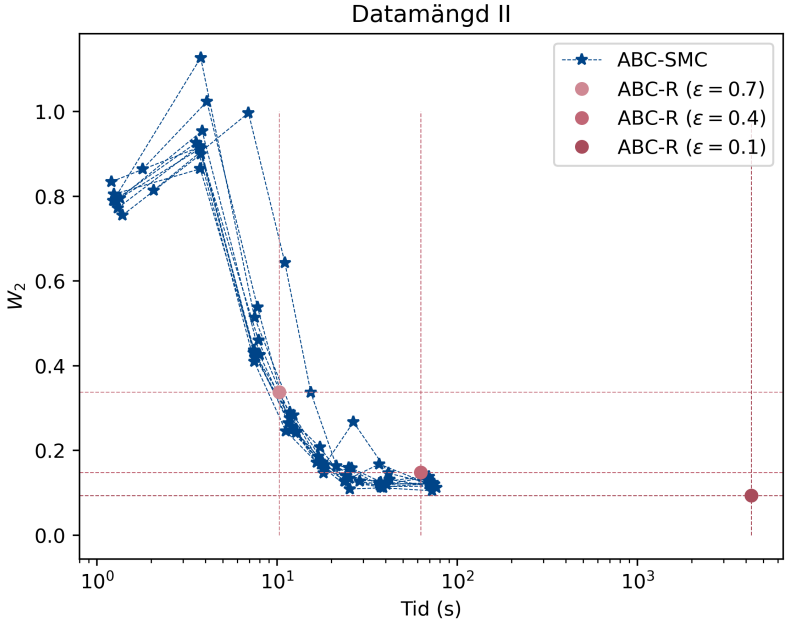
Figur 3: $\pi_\epsilon(\boldsymbol{\theta}|S(\boldsymbol{X}^{obs}))$ och $\pi(\boldsymbol{\theta}|\boldsymbol{X}^{obs})$ för de olika datamängderna. Översta raden: $\mathcal{D}_{\mathrm{I}}$ med $\Delta t = 0.1$. Mellersta raden: $\mathcal{D}_{\mathrm{II}}$ med $\Delta t = 0.3$. Nedersta raden: $\mathcal{D}_{\mathrm{III}}$ med $\Delta t = 1.0$

4.2 ABC-metodernas tidsåtgång för olika tidssteg Δt

I tabell 3 presenteras tidsåtgången för parameterinferens med ABC-R och ABC-SMC för respektive datamängd. I figur 4 ser vi tidsutvecklingen av W_2 för ABC-SMC och ABC-R för $\mathcal{D}_{\mathrm{II}}$. Se B.6 för samma visualisering för övriga datamängder. Tidsåtgången för ABC-R ökar kraftigt då toleransen ϵ sänks - särskilt från $\epsilon = 0.4$ till $\epsilon = 0.1$.

Tabell 3: Jämförelse av tidsåtgång för de olika ABC-metoderna där N_{sim} är antalet simulerade banor i algoritmerna. Värden för ABC SMC är medelvärden över tio körningar, med $N = 750$ och $q = 30\%$.

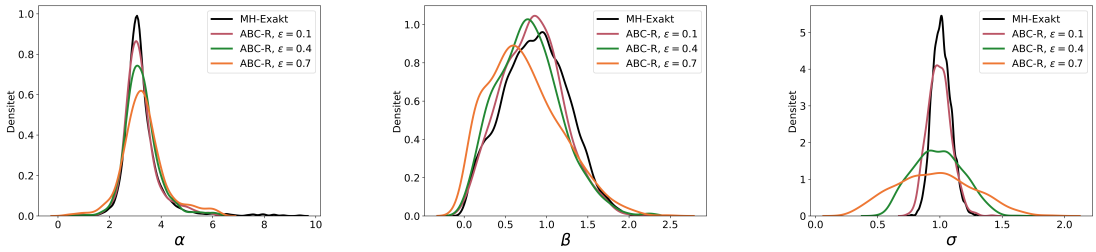
	Tid $(hh:mm:ss)$ $\mathcal{D}_{\mathrm{I}}$	Tid $(hh:mm:ss)$ $\mathcal{D}_{\mathrm{II}}$	Tid $(hh:mm:ss)$ $\mathcal{D}_{\mathrm{III}}$
ABC-$R_{\epsilon=0.1}$	00 : 39 : 57	01 : 11 : 15	02 : 52 : 40
ABC-$R_{\epsilon=0.4}$	00 : 00 : 33	00 : 01 : 03	00 : 02 : 11
ABC-$R_{\epsilon=0.7}$	00 : 00 : 06	00 : 00 : 10	00 : 00 : 17
ABC-SMC	$00:01:03 \pm 08$	$00:01:12 \pm 03$	$00:01:40 \pm 05$



Figur 4: 10 körningar av ABC-SMC med $N = 750$, $q = 30\%$ och stoppkriterium: acceptanskvot $< 1.5\%$, samt ABC-R med $\epsilon = [0.7, 0.4, 0.1]$ för $\mathcal{D}_{\text{II}}$. På y-axeln är W_2-avstånd och på x-axeln är tid (s) i logaritmerad skala.

4.3 Hur toleransen ϵ påverkar ABC-R

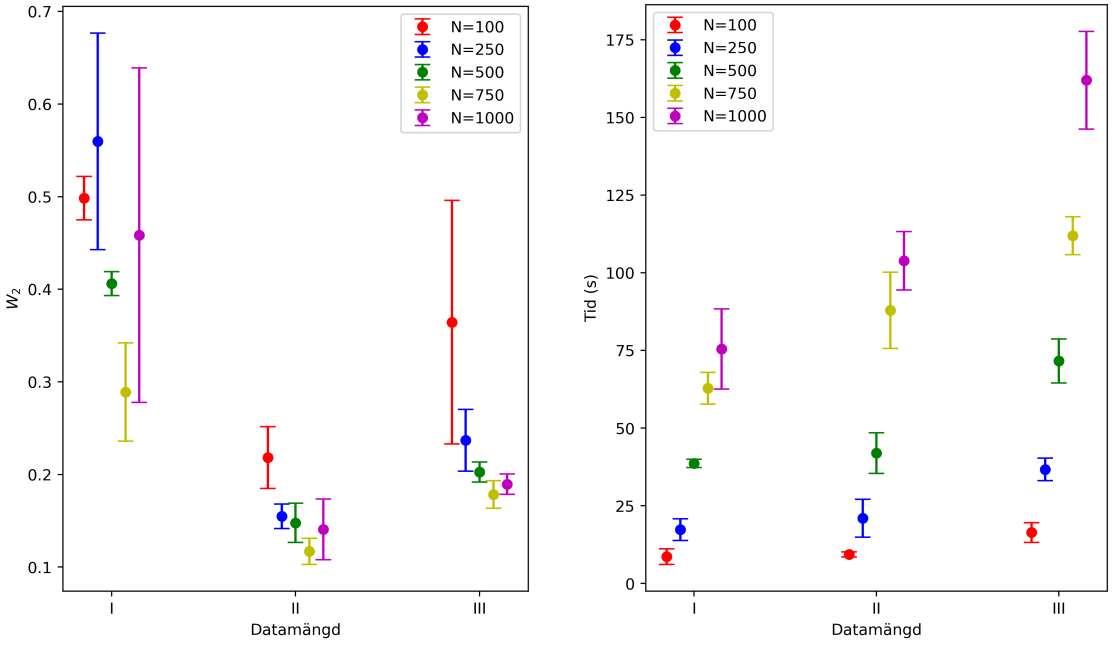
För ABC-R kan endast konstanten ϵ varieras. I figur 5 visas a posteriori-fördelningar, $\pi_\epsilon(\boldsymbol{\theta}|S(\boldsymbol{X}^{obs}))$, genererade med tolerans $\epsilon = [0.1,\ 0.4,\ 0.7]$ samt $\pi(\boldsymbol{\theta}|\boldsymbol{X}^{obs})$ genererade med MH-Exakt för $\mathcal{D}_{\text{I}}$. Resultatet för $\mathcal{D}_{\text{II}}$ och $\mathcal{D}_{\text{III}}$ följer samma trend och visas i appendix (B.6). Tydligt i graferna är att approximationen $\pi_\epsilon(\boldsymbol{\theta}|S(\boldsymbol{X}^{obs}))$ av $\pi(\boldsymbol{\theta}|\boldsymbol{X}^{obs})$ blir bättre med lägre ϵ och fördelningen närmar sig då referensfördelningen MH-Exakt.



Figur 5: $\pi_\epsilon(\boldsymbol{\theta}|S(\boldsymbol{X}^{obs}))$ och $\pi(\boldsymbol{\theta}|\boldsymbol{X}^{obs})$ för $\boldsymbol{X}^{obs} = \mathcal{D}_{\text{I}}$ med $\Delta t = 0.1$

4.4 Hur antal partiklar och val av toleranskvantil påverkar ABC-SMC

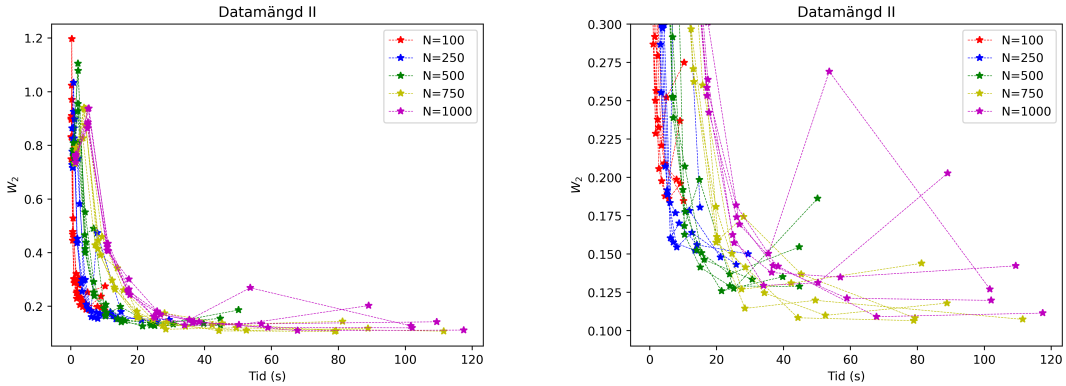
För ABC-SMC varierades två konstanter: Antalet partiklar N och toleranskvantilen q. I figur 6 (a) redovisas genomsnittliga W_2-avstånd, samt standardavvikelse, över 5 a posteriori-fördelningar genererade för varje N i listan $[100, 250, 500, 750, 1000]$. Med ökande N, från 100 till 750, sjunker överlag W_2. För N=1000 ökar dock W_2-avståndet igen, och $N = 750$ gav för samtliga datamängder upphov till fördelningar med lägst W_2-avstånd. I figur 6 (b) redovisas tidsåtgång för samma körningar. Vi ser en tydlig trend i att högre N ger större tidsåtgång.



(a) **(b)**

Figur 6: Låddiagram med medelvärden och standardavvikelse av W_2 (a) och Tid (s) (b), för $\mathcal{D}_{\text{I}}$, $\mathcal{D}_{\text{II}}$ och $\mathcal{D}_{\text{III}}$. För varje N i listan [100, 250, 500, 750, 1000] är medelvärde och standardavvikelse baserat på fem a posteriori-fördelningar. Tabell över datan finns i appendix (4)

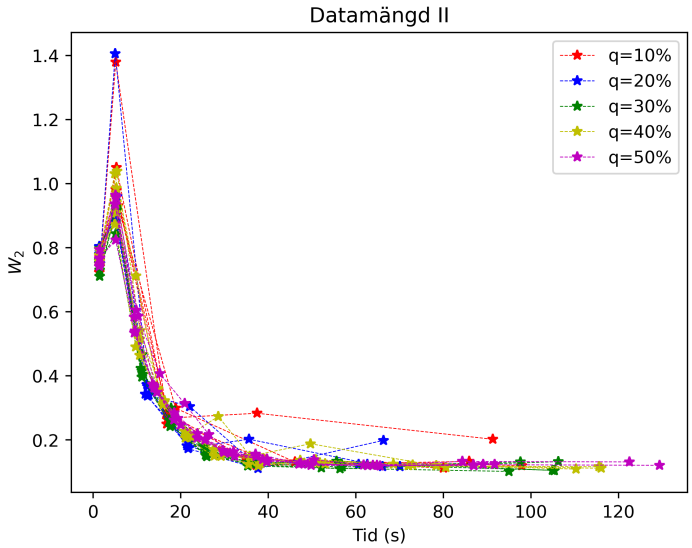
I figur 7 ses tidsutveckling av W_2-avstånd för $\mathcal{D}_{\text{II}}$. Motsvarande figurer för övriga datamängder återfinns i B.6.



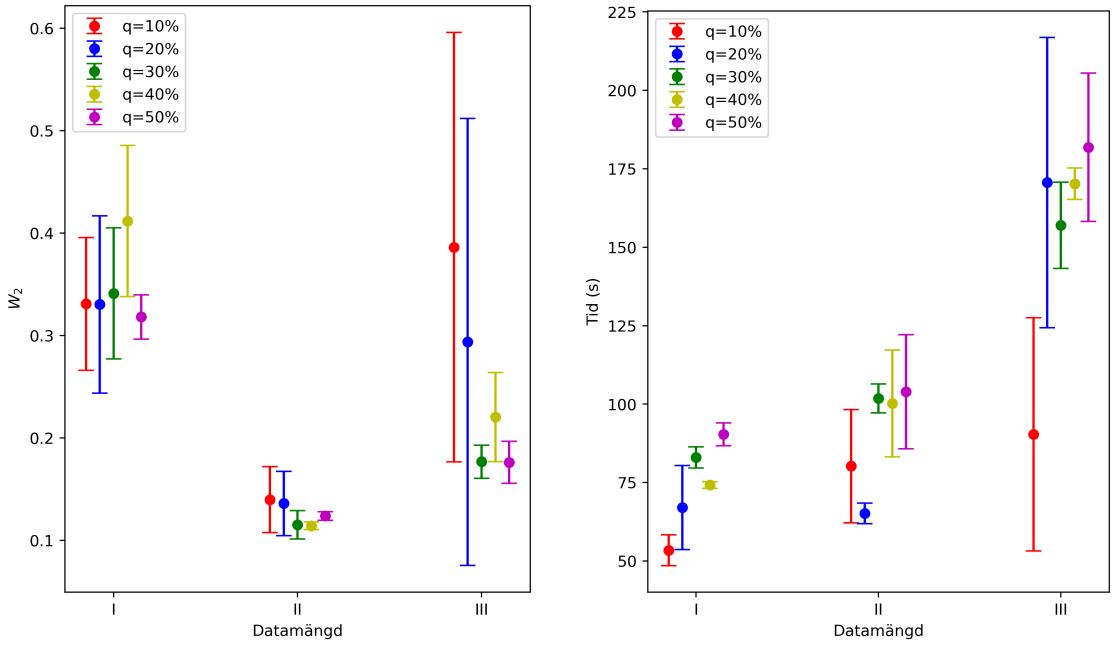
(a) **(b)**

Figur 7: (a) Jämförelse av W_2-avstånd mellan olika antal partiklar N som genereras varje omgång för för ABC-SMC utförd på $\mathcal{D}_{\text{II}}$. För varje N gjordes 5 körningar. Programmet stoppades då acceptanskvoten blev lägre än 1.5%. (b) förstoring vid lägre W_2-avstånd för att förtydliga skillnaden mellan var de olika banorna konvergerar

Toleranskvantilen q, som kontrollerar vilken kvantil av partiklar som accepteras varje runda, verkar inte påverka effektiviten vilket visas i figur 8. Det som syns i figur 9 är att med större q tar det lite mer tid att utföra inferensen men den blir också lite bättre. Effektiviteten är därför ungefär densamma och skillnaden är att lägre q ofta stoppar tidigare.



Figur 8: 5 körningar av ABC SMC på $\mathcal{D}_{\text{II}}$ med varje q i listan $[10\%, 20\%, 30\%, 40\%, 50\%]$. På x-axeln är tid i sekunder och på y-axeln är W_2-avståndet.



(a) **(b)**

Figur 9: Genomsnittligt W_2-avstånd (a) och tidsåtgång (b), när algoritmen stannats, för respektive q och $\mathcal{D}_{\text{I}}$, $\mathcal{D}_{\text{II}}$ och $\mathcal{D}_{\text{III}}$.

5 Diskussion och slutsats

Arbetet ämnade att utvärdera precisionen av likelihood-fria algoritmer för parameterinferens på SDE:er. Resultaten av experimenten i 4.1 visar att de likelihood-fria metoderna ABC-R och ABC-SMC gav a posteriori-fördelningar med god noggrannhet för data genererade med Ornstein-Uhlenbeck (OU) då de jämfördes med fördelningar från MH-exakt. ABC-metoderna visade sig även vara mer robusta vid hantering av datamängder med större Δt jämfört med MH-EM, vilket syns i figur 3 i bild (e), (f), (h) och (i). MH-EM förlorar snabbt tillförlitlighet då Δt ökar eftersom EM-approximationen för övergångstätheter är dålig för stora Δt vilket syns tydligt i figur 3. ABC-metoderna kan alltså vara ett komplement till MH-EM-metoden för att utföra likeliehood-fri parameterinferens på datamängder där Δt är stort.

I 4.1 och 4.2 ser vi att R-ABC med $\epsilon = 0.1$ ger högre noggrannhet än ABC-SMC för alla datamängder och med $\epsilon = 0.4$ får vi både lägre tidsåtgång och högre noggrannhet för $\mathcal{D}_{\text{I}}$. Detta resultat är inte helt rättvisande med avseende på tidsåtgången för ABC-R. Tidsåtgången är starkt beroende av valet av a priori-fördelningen och för syntetisk data där $\pi(\boldsymbol{\theta})$ är känt så kan a priori-fördelningen väljas till att vara smal och precis. När denna uppskattning av $\boldsymbol{\theta}$ inte är tillgänglig måste en bredare a priori-fördelning användas vilket betyder att acceptanskvoten blir mycket låg och ABC-R tar mer tid. ABC-SMC kringgår det problemet och är därmed mindre beroende av val av a priori-fördelning. Därför kan kan förvänta oss att parameterinferens med ABC-R på verklig data skulle ta betydligt längre tid än med ABC-SMC.

Toleransen ϵ var viktig för ABC-R:s noggrannhet och för ABC-SMC var antalet partiklar N avgörande medan valet av toleranskvantil q generellt spelar mindre roll. I 4.3 ser vi att lägre tolerans ger noggrannare inferens för ABC-R vilket var väntat då strängare krav på vilka förslag på parametrar som accepteras bör ge noggrannare resultat. Ur 4.1 och 4.2 framgår det dock att det krävdes markant mer datorkraft för köra algoritmen med låg tolerans för att uppnå lågt W_2-avstånd. Figur 7 visar att för ABC-SMC gav högre antal partiklar N, genererade varje omgång, noggrannare parameterinferens med ökad datorkraft som kostnad. I 4.4 ser vi däremot att tolerenskvantilen q, vilken avgör vilken kvantil som accepteras av varje runda, inte har någon signifikant påverkan på mängden använd datorkraft. Däremot påverkar valet av q med vår stoppregel när algoritmen stannar vilket förklarar den påverkan q får för noggrannheten vilket kan ses i figur 9.

I figur 2 ser vi att W_2-avståndet för MH-EM applicerad på $\mathcal{D}_{\text{II}}$ var lägre än för $\mathcal{D}_{\text{I}}$ vilket kan verka något motsägelsefullt då EM-approximationen generellt blir sämre med större Δt. Detta skulle kunna förklaras av att $\mathcal{D}_{\text{II}}$ slumpmässigt genererades på ett sätt som var mer informativt om parametrarna $\boldsymbol{\theta}$ än vad $\mathcal{D}_{\text{I}}$ var. Givet sådana slumpmässiga utfalls inverkan på resultatet är det inte givande att jämföra W_2-avståndet mellan olika datamängder. Det som kan jämföras är istället den relativa skillnaden i W_2-avstånd för olika metoder inom samma datamängd. Om istället densiteterna för MH-EM (gul) mot MH-Exakt (svart) i figur 3 jämförs visuellt ser avståndet mellan densiteterna ut att vara större för $\mathcal{D}_{\text{II}}$ än för $\mathcal{D}_{\text{I}}$. Detta motsäger alltså W_2-resultaten i tabell 2. Mot bakgrund av detta bör resultat som de i figur 2 tolkas med viss försiktighet och alltid jämföras med visualiseringar likt de i figur 3.

Ett något avvikande resultat som kan ses i figur 3 (a), (d) och (g) är att MH-EM är bra på att beräkna a posteriori-fördelningen för α, en av parametrarna i $\boldsymbol{\theta}$, även vid stora tidssteg. En realisering av OU kan delas upp i två delar där den första är en insvängningingstid och den andra ett jämviktstillstånd där grafen pendlar runt α (se figur 1). Om insvängningstiden är tillräckligt kort jämfört med tiden i jämvikt borde alltså redan en frekventistisk medelvärdesbildning ge en god uppskattning av α. Denna observation skulle kunna förklara varför även data med stora tidssteg är informativa för α. Eftersom vi lät antalet datapunkter N_{obs} vara fixt gav längre tidssteg längre total tid. Detta minskade insvängningstidens andel av bankurvan och kan därmed ha gjort datan mer informativ för α. Detta är också ett resultat vi kan skönja i bild (a), (d) och (g). Eftersom W_2-avståndet är ett tredimensionellt mått som sammanväger noggrannheten i uppskattningen av alla tre parametrarna skulle MH-EM:s goda förmåga att uppskatta α kunna uppväga för dess dåliga uppskattning av β och σ. Vi får alltså två trender som delvis tar ut varandra uppväger varandra i figur 2. Där kan vi se att misslyckandet i uppskattningen av β och σ ändå väger över. Att en parameter kan uppskattas med god noggrannhet med frekventiska metoder får antas vara ovanligt

och resultatet är därför inte något som väger upp för MH-EM:s svagheter.

Som tidigare nämnts i 2.7.3 approximerar ABC-R och ABC-SMC den sanna tätheten $\pi(\boldsymbol{\theta}|\boldsymbol{X})$ som $\pi_\epsilon(\boldsymbol{\theta}|S(\boldsymbol{X}^{obs}))$ med deskriptiv statistik $S(\boldsymbol{X})$ genererad av det neurala nätverket benämnt PEN, och kommer därför aldrig ge en helt exakt a posteori-fördelning. Algoritmerna är således mycket beroende av den deskriptiva statistiken vilket innebär att träningen av de neurala nätverken är avgörande för algoritmernas prestanda. Träningen behöver göras för varje datamängd med unik storlek på tidssteg, vilket är en ytterligare kostnad sett till datorkraft. För ABC-SMC kan banorna genererade varje runda användas för att förfina träningen av nätverket på banor med parametrar nära de sanna parametrarna [27]. Denna typ av sekvensiella träning implementerades inte för ABC-SMC i vårt arbete men det vore en självklar förbättring om algoritmen skulle vidareutvecklas. Sekvensiell träning kan inte implementeras för ABC-R eftersom metoden inte är sekvensiell.

5.1 Etiska aspekter

Eftersom all data som används i arbetet är simulerad har inga etiska aspekter behövt tas hänsyn till i det löpande arbetet. Den etiska dimensionen är alltså endast aktuell då de följdverkningar framsteg inom området kan leda till beaktas. Simuleringsdriven inferens av SDE:er är ett nytt område och studier likt vår, som undersöker metodernas tillförlitlighet, kan öppna upp för användandning av nya inferensmetoder för praktiska problem. Ett sådant kan vara att utföra inferens på börsdata för att förutsäga börsrörelser och prissätta finansiella derivat.

Att framsteg på området kan få stora samhälleliga konsekvenser kan ses i en historisk tillbakablick. Teoribildningen för Brownsk rörelse och stokastisk analys möjliggjorde härledningen av Black-Scholes formel som anger den förväntade avkastningen från optioner [28]. Black-Scholes öppnade från 80-talet och framåt upp för prissättning av och mer omfattande handel med optioner vilket var en disruption som revolutionerade finasmarknaderna. På grund av bristande insikt om modellernas begränsningar blåstes en spekulationsbubbla upp med lager av allt mer komplicerade finansiella derivat vilka alla kraftigt underskattade risken i de underliggande kapitalmarknaderna. När bubblan sprack resulterade det i bankkrisen 2008.

I vårt arbete ser vi att ABC-metoderna framförallt var bättre än MH-EM för stora tidssteg. Det finns många praktiska problem där antalet tillgängliga datapunkter är kraftigt begränsat och ABC-metoderna alltså hade varit användbara - exempelvis data som endast uppdateras en gång varje dag. Även om detta breddar tillämpningsområdena innebär det inte något nytt konceptuellt och förhoppningsvis är investerarnas förståelsen för de modellernas begränsningar större idag

5.2 Slutsats

Arbetet har jämfört parameterinferens med likelihood-fria algoritmer på SDE:en Ornstein-Uhlenbeck. Specifikt studerades ABC-R, ABC-SMC samt MH-EM. A posteriori fördelningarna jämfördes med de sanna fördelningarna från MH-exakt och noggrannheten kvantifierades som avvikelsen från de sanna fördelningarna mätt i Wasserstein-avstånd. Det tydligaste resultatet är att ABC-SMC klart är att föredra jämfört med de andra undersökta metoderna då mått som noggrannhet, effektivitet och anpassningsbarhet till olika datamängder sammanvägs. ABC-R gav något mer precisa resultat men metoden är ineffektiv i termer av använd datorkraft - speciellt för icke-syntetisk data. MH-EM visade sig endast fungera då utformningen av datamängderna var ideal med liten storlek på tidssteg mellan datapunkter. Resultatet utvecklar i viss mån förståelsen för ABC-R och ABC-SMC:s prestanda, begränsningar och användbarhet för parameterinferens på SDE:er. För vidare forskning skulle det vara av intresse att undersöka metodernas prestanda på andra SDE:er som, likt Ornstein-Uhlenbeck, har analytiska lösningar - exempelvis Cox-Ingersoll-Ross och geometrisk Brownsk rörelse.

Referenser

[1] H. M. Taylor and S. Karlin, *An Introduction to Stochastic Modeling*, p. preface. London: Academic Press Limited, 3 ed., 1998.

[2] D. J. Higham., "An algorithmic introduction to numerical simulation of stochastic differential equations," *SIAM Review*, vol. 43, no. 3, pp. 525–546, 2001.

[3] F. C. Klebaner, *Introduction to Stochastic Calculus with Applications.* London: Imperial College Press, 3 ed., 2012.

[4] G. E. Uhlenbeck and L. S. Ornstein, "On the theory of the brownian motion," *Phys. Rev.*, vol. 36, pp. 823–841, Sep 1930.

[5] C. P. Robert and G. Casella, *Monte Carlo Statistical Methods*, pp. 267–270. New York: Springer, 2 ed., 2004.

[6] J. K. Pritchard, M. T. Seielstad, A. Perez-Lezaun, , and M. W. Feldman, "Population growth of human y chromosomes: A study of y chromosome microsatellites," *Molecular Biology and Evolution*, vol. 16, no. 12, pp. 1791–1798, 1999.

[7] S. Tavaré, D. J. Balding, R. C. Griffiths, and P. Donnelly, "Inferring coalescence times from dna sequence data," *Genetics*, vol. 145, no. 2, pp. 505–518, 1997.

[8] S. A. Sisson, Y. Fan, and M. M. Tanaka, "Sequential monte carlo without likelihoods," *Proc Natl Acad Sci U S A*, vol. 104, no. 6, pp. 1760–1765, 2007.

[9] T. Toni, D. Welch, N. Strelkowaand, A. Ipsen, and M. P. Stumpf, "Approximate bayesian computation scheme for parameter inference and model selection in dynamical systems," *Journal of the Royal Society Interface*, vol. 6, pp. 187–202, 2009.

[10] S. E. Shreve, *Stochastic Calculus for Finance II.* Pittsburg: Springer, 8 ed., 2008.

[11] G. R. Grimmet and D. R. Stirzaker, *Probability and Random Processes*, pp. 213–214. New York: Oxford University Press, 3 ed., 2001.

[12] K. C. Chan, G. A. Karolyi, F. A. Longstaff, and A. B. Sanders, "An empirical comparison of alternative models of the short-term interest rate," *The Journal of Finance*, vol. 47, no. 3, pp. 1209–1227, 1992.

[13] S. M. Iacus, *Simulation and Inference for Stochastic Differential Equations.* Milan: Springer, 1 ed., 2008.

[14] E. Platen and P. E. Kloeden, *Numerical Solution of Stochastic Differential Equations*, pp. 305–307. Heidelberg: Springer Berlin, 1 ed., 2011.

[15] D. D. Boos and L. Stefanski, *Essential Statistical Inference, Theory and Methods*, pp. 163–164. Springer, 1 ed., 2008.

[16] O. Elerian, S. Chib, and N. Shephard, "Likelihood inference for discretely observed nonlinear diffusions," *Econometrica*, vol. 69, no. 4, pp. 959–993, 2001.

[17] C. Fuchs, *Inference for Diffusion Processes*, pp. 171–174. Heidelberg: Springer Berlin, 1 ed., 2013.

[18] D. B. Hitchcock, "A history of the metropolis-hastings algorithm," *The American Statistician*, vol. 57, no. 4, pp. 254–257, 2003.

[19] N. Metropolis and S. Ulam, "The monte carlo method," *Journal of the American Statistical Association*, vol. 44, no. 247, pp. 335–341, 1949. PMID: 18139350.

[20] S. Chib and E. Greenberg, "Understanding the metropolis-hastings algorithm," *The American Statistician*, vol. 49, no. 4, pp. 327–335, 1995.

[21] J. Owen, D. J. Wilkinson, and C. S. Gillespie, "Likelihood free inference for markov processes: A comparison," *Statistical Applications in Genetics and Molecular Biology*, vol. 14, no. 2, pp. 189–209, 2015.

[22] H. Haario, E. Saksman, and J. Tamminen, "An adaptive metropolis algorithm," *Bernoulli*, vol. 7, no. 2, pp. 223–242, 2001.

[23] A. Gelman, G. Roberts, and W. Gilks, "Efficient metropolis jumping rules," *Bayesian Statistics*, 1996.

[24] L. Chen, *Curse of Dimensionality*, pp. 545–546. Boston, MA: Springer US, 2009.

[25] S. Wiqvist, P.-A. Mattei, U. Picchini, and J. Frellsen, "Partially exchangeable networks and architectures for learning summary statistics in approximate bayesian computation," in *Proceedings of the 36th International Conference on Machine Learning* (K. Chaudhuri and R. Salakhutdinov, eds.), vol. 97 of *Proceedings of Machine Learning Research*, pp. 6798–6807, PMLR, 09–15 Jun 2019.

[26] S. Kolouri, S. Park, M. Thorpe, D. Slepčev, and G. K. Rohde, "Optimal mass transport: Signal processing and machine-learning applications," *IEEE signal processing magazine*, vol. 34, no. 1, pp. 43–59, 2017.

[27] Y. Chen, D. Zhang, M. Gutmann, A. Courville, and Z. Zhu, "Neural approximate sufficient statistics for implicit models," 2021.

[28] I. Stewart, "The Mathematical Equation That Caused the Banks to Crash," 2012.

A Appendix 1 – begrepp och akronymer

Begreppslista

A priori-fördelning	Sannolikhetsfördelning som uttrycker kunskap om parametrarna innan observationer beaktas
Likelihood-funktion	Sannolikheten för datan givet parametrarna
A posteriori-fördelning	Sannolikhetsfördelning för parametrarna betingat på data som uppdaterar a priori-fördelningen med information från likelihood-funktionen utifrån Bayes regel
Parameterinferens	Att dra slutsatser om parametrar utifrån sammanhang
Övergångstäthet	Täthetsfunktion för X_t givet tidigare värde X_{t-1}
Wasserstein-avstånd	Mått på olikheter mellan två flerdimensionella fördelningar.

Akronymer

ABC	Approximate Bayesian Computation
CIR-process	Cox-Ingersoll-Ross-process
CKLS-process	Chan-Karolyi-Longstaff-Sanders-process
OU-process	Ornstein-Uhlenbeck-process
EM	Euler-Maruyama
EMD	Earth Mover's Distance
MLE	Maximum Likelihood Estimation
MCMC-metod	Markov Chain Monte-Carlo-metod
MH-metod	Metropolis-Hastings-metod
SDE	Stokastiska Differentialekvationer
ODE	Ordinär Differentialekvation
ABC-SMC	Approximate Bayesian Computation Sequential Monte Carlo
ABC-R	Approximate Bayesian Computation Rejection Sampler
PEN	Partially Exchangeable Network

B Appendix 2 – teori och resultat

B.1 Sannolikhetsteori

Definition 7 (Sigma-algebra [10]). *Låt Ω vara en icke-tom mängd och $\mathcal{F}$ vara en familj av delmängder till Ω. Då är $\mathcal{F}$ en sigma-algebra om följande villkor är uppfyllda.*

- *Den tomma mängden, $\emptyset$, tillhör $\mathcal{F}$.*
- *Om en mängd $\mathcal{A}$ tillhör $\mathcal{F}$ så tillhör även dess komplement $\mathcal{A}^c$ mängden $\mathcal{F}$.*
- *Om en följd av mängder $\mathcal{A}_1, \mathcal{A}_2, ...$ tillhör $\mathcal{F}$ så tillhör även unionen $\bigcup_{n=1}^{\infty} \mathcal{A}_n$ mängden $\mathcal{F}$.*

Det är intuitivt att se en sigma-algebra som ett system för att organisera information av möjliga utfall ω i ett slumpexperiment. Sigma-algebran innehåller delmängder av utfallsrummet där varje delmängd representerar en viss händelse. När experimentet utförs får vi veta vilka av dessa händelser som inträffar utan att direkt veta det exakta värdet av utfallet ω. Ju mer detaljerade och omfattande delmängderna i sigma-algebran är, desto mer information får vi om det underliggande utfallet ω. Om sigma-algebran endast innehåller den tomma mängden $\emptyset$ och hela utfallsrummet Ω, så lär vi oss inget nytt om ω [10].

Definition 8 (Sannolkhetsmått [10]). *Låt Ω vara en icke-tom mängd och låt $\mathcal{F}$ vara en sigma-algebra till Ω. Ett sannolkhetsmått $\mathbb{P}$ är en funktion som för varje delmängd $\mathcal{A} \in \mathcal{F}$ tilldelar ett tal i intervallet $[0, 1]$. Detta kallas sannolikheten av $\mathcal{A}$ och skrivs $\mathbb{P}(\mathcal{A})$. Vi kräver även uppfyllnad av följande villkor*

- $\mathbb{P}(\Omega) = 1$.
- *En disjunkt följd av delmängder $\mathcal{A}_1, \mathcal{A}_2, ... \in \mathcal{F}$ implicerar att $\mathbb{P}\big(\bigcup_{n=1}^{\infty}\big) = \sum_{n=1}^{\infty} \mathbb{P}(\mathcal{A}_n)$.*

Trippeln $\big(\Omega, \mathcal{F}, \mathbb{P}\big)$ kallas för ett sannolikhetsrum.

Definition 9 (Filtration [10]). *Låt Ω vara en icke-tom mängd och antag att det för varje $t \in [0, T]$ finns en sigma-algebra $\mathcal{F}(t)$. Antag vidare att om $s \leq t$ så gäller $\mathcal{F}(s) \subseteq \mathcal{F}(t)$. Då kallas samlingen av sigma-algebror $\{\mathcal{F}(t)\}_{t\in[0,T]}$ för en filtration.*

B.1.1 Fördelnings- och täthetsfunktioner

Den kumulativa fördelningsfunktionen $F(x) = P(X \leq x) = P(X(\omega) \in (-\infty, x]$ är en ickeavtagande funktion s.a. $\lim_{x\to-\infty} F(x) = 0$, $\lim_{x\to\infty} F(x) = 1$ och F är högerkontinuerlig. Om F är absolut kontinuerlig kallas dess derivata $f(x)$ täthetsfunktionen vilket är en Lebesgueintegrerbar ickenegativ funktion s.a. $\int_{-\infty}^{\infty} f(x)dx = 1$.

B.1.2 Väntevärde och varians

Om $g(\cdot)$ är en mätbar funktion och X en kontinuerlig slumpvariabel med fördelningsfunktion F, så ges väntevärdet av $g(X)$ enligt

$$\mathbb{E}[g(X)] = \int_{\Omega} g(X(\omega))dP(\omega) = \int_{\mathbb{R}} g(x)dF(x) = \int_{\mathbb{R}} g(x)f(x)dx \tag{32}$$

givet att integralen är begränsad och där den sista likheten endast gäller om F är absolutkontinuerlig. Om Ω är uppräkneligt definieras väntevärdet som

$$\mathbb{E}[X] = \sum_{\omega\in\Omega} X(\omega)P(\omega) = \sum_{x\in I} xP(X = x) \tag{33}$$

där I är mängden av möjliga värden på X och den sista likheten endast gäller om X är en diskret slumpvariabel.

Variansen av X definieras som

$$\mathrm{Var}X = \mathbb{E}[(X - \mathbb{E}(X))^2] = \int_{\Omega} (X(\omega) - \mathbb{E}X)^2 dP(\omega). \tag{34}$$

För diskreta variabler är motsvarigheten till ekvation 32 endast approximativ och ges av de stora talen lag. Anta att vi kan dra tal $x_1, ..., x_n$ från X fördelning. Då kan $\mathbb{E}[g(X)]$ approximeras som

$$\mathbb{E}[g(X)] \approx \frac{1}{n}\sum_{i=1}^{n} g(x_i) = \overline{g}_n \xrightarrow{\mathrm{d}} N(\mathbb{E}[g(X)], \frac{1}{n}\mathrm{Var}[g(X)]). \tag{35}$$

Det sista ledet ges av centrala gränsvärdessatsen. När $n \to \infty$ går alltså variansen mot noll och det approximativa uttrycket blir exakt.

Definition 10 (Martingal [3])**.** *En stokastisk process $\{X(t)\}_{t\geq 0}$ är en martingal om den för alla t är integrerbar, $\mathbb{E}|X(t)| < \infty$, och det för alla $s > 0$ gäller att*

$$\mathbb{E}[X(t+s)|\mathcal{F}(t)] = X(t).$$

Här är $\mathcal{F}(t)$ processens associerade filtration.

Om en process besitter martingal-egenskapen är alltså vår bästa uppskattning om värdet vid tiden $t + s$, processens nuvarande värde.

B.2 Lösning av Ornstein-Uhlenbeck

Vi har definierat Ornstein-Uhlenbeckprocessen i (11) som

$$dX(t) = \beta(\alpha - X(t))dt + \sigma dW(t).$$

och vi löser denna genom att använda ansatsen $f(t, x) = e^{\beta t}x$ som har derivatorna $f_t = e^{\beta t}\beta x,\ f_x = e^{\beta t},\ f_{xx} = 0$. Enligt Itô-Doeblinformeln (9) får vi då

$$\begin{aligned} d(e^{\beta t}X(t)) &= e^{\beta t}\beta X(t)dt + e^{\beta t}dX(t) \\ &= e^{\beta t}\beta X(t)dt + e^{\beta t}\big[\beta(\alpha - X(t))dt + \sigma dW(t)\big] \\ &= \alpha\beta e^{\beta t}dt + \sigma e^{\beta t}dW(t). \end{aligned}$$

Integrering ger

$$\begin{aligned} e^{\beta t}X(t) &= X(0) + \alpha\beta\int_0^t e^{\beta t}ds + \sigma\int_0^t e^{\beta s}dW(s) \\ &= X(0) + \alpha(e^{\beta t} - 1) + \sigma\int_0^t e^{\beta s}dW(s) \end{aligned}$$

och vi får att

$$X(t) = \alpha + (X(0) - \alpha)e^{-\beta t} + \sigma\int_0^t e^{-\beta(t-s)}dW(s).$$

Det följer av Itôintegralens egenskaper (6) att OU-processens väntevärde och varians ges av

$$\mathbb{E}[X(t)] = \alpha + (X(0) - \alpha)e^{-\beta t}, \tag{36a}$$

$$\mathrm{Var}[X(t)] = \frac{\sigma^2}{2\beta}(1 - e^{-2\beta t}). \tag{36b}$$

B.3 Stark och svag konvergens för Euler-Maruyamametoden

Euler-Maruyamametoden uppvisar både stark och svag konvergens när den används för att lösa stokastiska differentialekvationer [2]. Stark konvergens innebär att metoden konvergerar i termer av väntevärde av absoluta skillnader mellan den approximerade och den sanna lösningen. För en metod med stark konvergensordning γ gäller att det finns en konstant C sådan att

$$\mathbb{E}\,|X_n - X(\tau)| \leq C\Delta t^{\gamma} \tag{37}$$

för alla $\tau = n\Delta t \in [0, T]$ och tillräckligt små Δt. Under vissa förutsättningar kan EM visas ha en stark konvergensordning $\gamma = \frac{1}{2}$ [2].

Svag konvergens innebär istället att metoden konvergerar i termer av väntevärden av en funktion p på lösningarna. En metod har svag konvergensordning γ om det finns en konstant C' sådan att

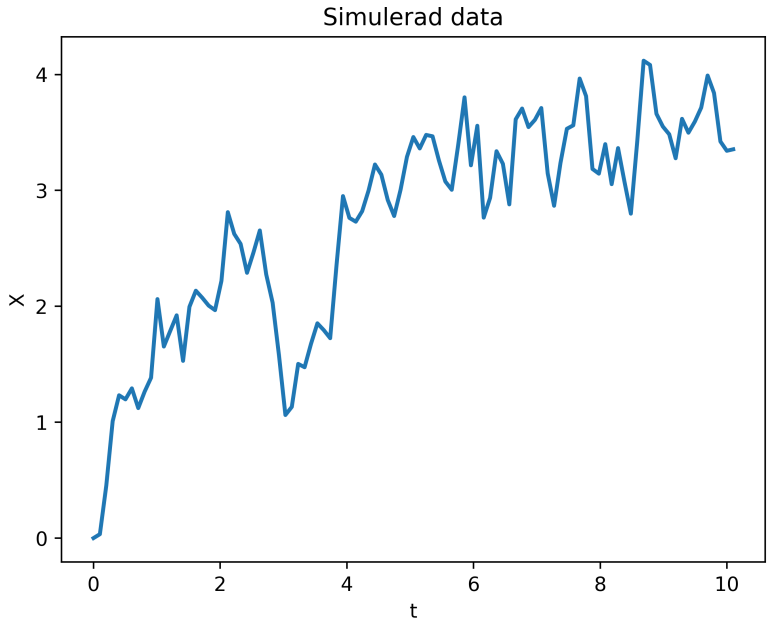
$$|\mathbb{E}p\left(X_n\right) - \mathbb{E}p\left(X(\tau)\right)| \leq C\Delta t^{\gamma} \tag{38}$$

för alla $\tau = n\Delta t \in [0, T]$ och tillräckligt små Δt, där funktionen p uppfyller vissa smidighets- och polynomiska tillväxtvillkor. EM kan visas ha en svag konvergensordning $\gamma = 1$ under vissa förutsättningar [2].

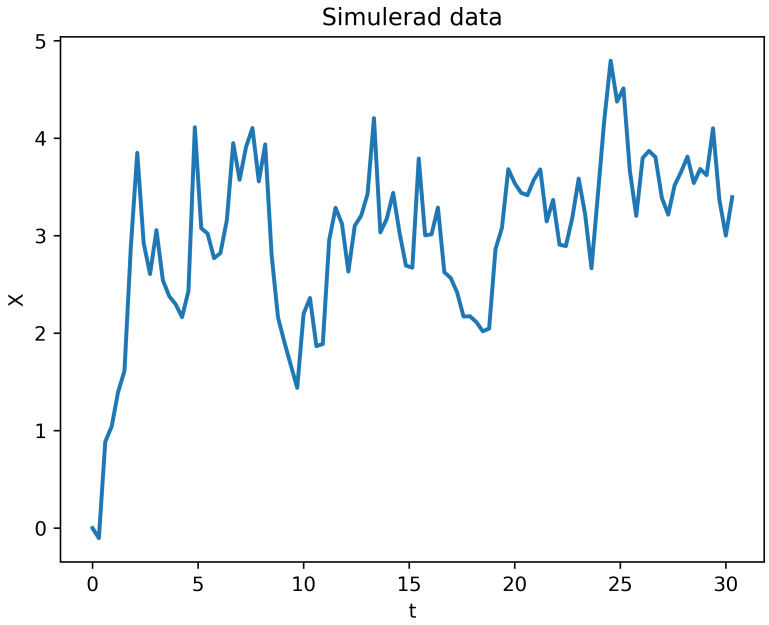
Vi noterar att stark konvergens implicerar svag konvergens, men tvärtom gäller inte. Beroende på det specifika problemet kan antingen stark eller svag konvergens vara mer relevant för att bedöma noggrannheten och stabiliteten i den numeriska metoden.

B.4 Syntetisk data

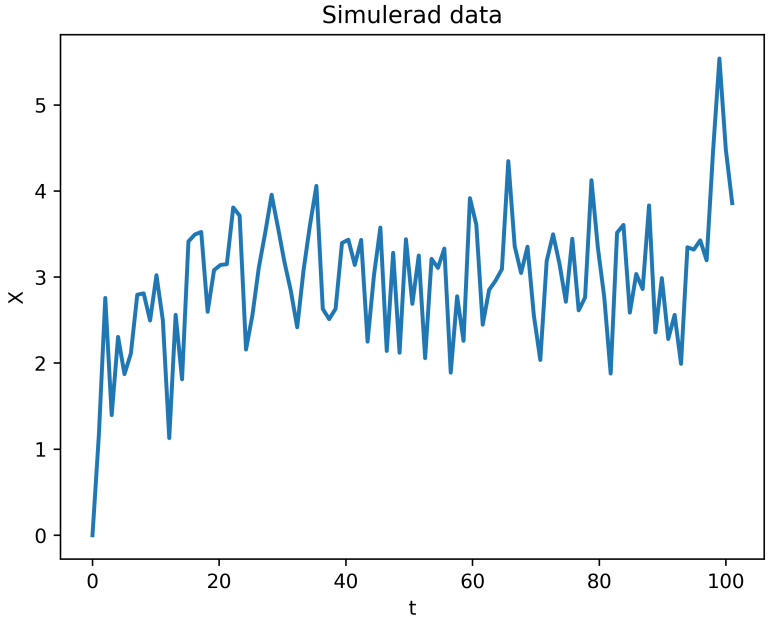
Figur 10, 11 och 12 visar syntetisk data för Ornstein-Uhlenbeck processer genererat med exakta övergångstätheter enligt (12).



Figur 10: Datamängd $\mathcal{D}_{\text{I}}$ med parametrar $\boldsymbol{\theta} = [3, 1, 1]$, storlek på tidssteg $\Delta t = 0.1$ och antal observationer $N_{obs} = 100$.

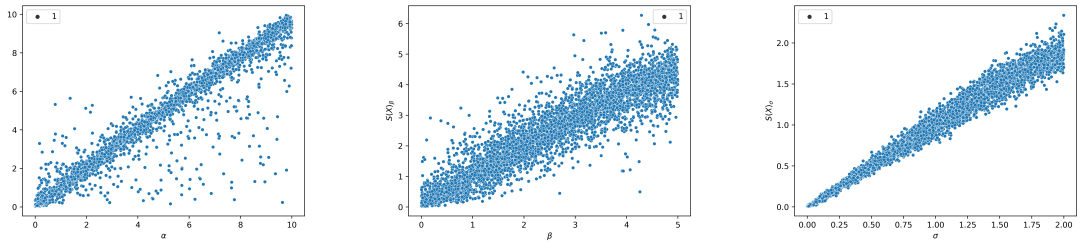


Figur 11: Datamängd $\mathcal{D}_{\text{II}}$ med parametrar $\boldsymbol{\theta} = [3, 1, 1]$, storlek på tidssteg $\Delta t = 0.3$ och antal observationer $N_{obs} = 100$.



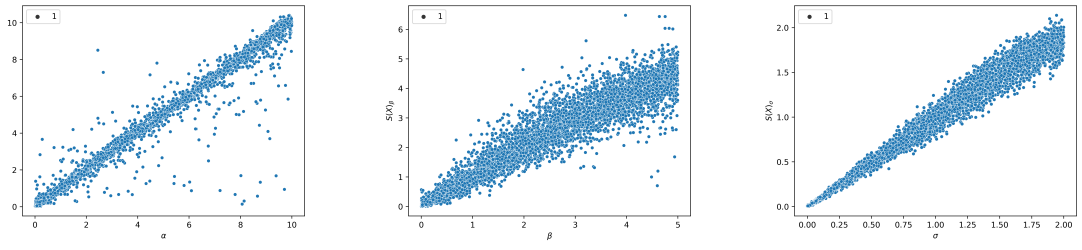
Figur 12: Datamängd $\mathcal{D}_{\text{III}}$ med parametrar $\boldsymbol{\theta} = [3, 1, 1]$, storlek på tidssteg $\Delta t = 1.0$ och antal observationer $N_{obs} = 100$.

B.5 Neurala nätverk



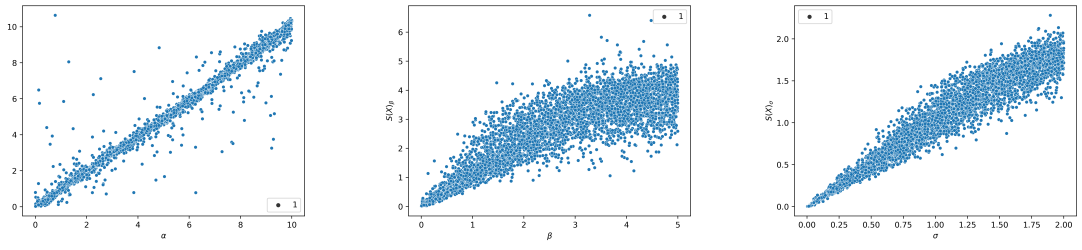
(a) **(b)** **(c)**

Figur 13: Neurala nätverkets deskriptiva statistik för 5000 simulerade datamängder med $\Delta t = 0.1$. På y-axeln visas $S(\boldsymbol{X})$ och på x-axeln visas parametrarna som datamängderna simulerats med



(a) **(b)** **(c)**

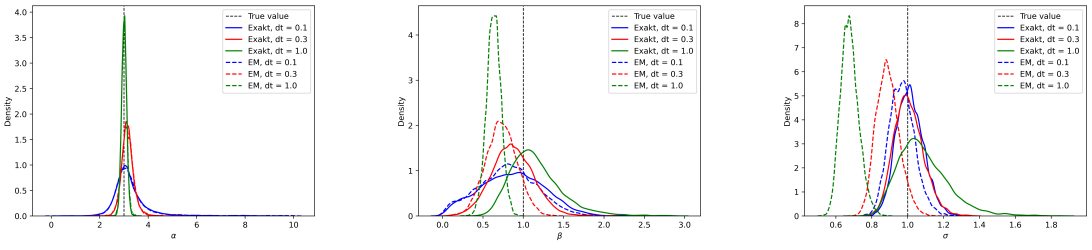
Figur 14: Neurala nätverkets deskriptiva statistik för 5000 simulerade datamängder med $\Delta t = 0.3$. På y-axeln visas $S(\boldsymbol{X})$ och på x-axeln visas parametrarna som datamängderna simulerats med



(a) **(b)** **(c)**

Figur 15: Neurala nätverkets deskriptiva statistik för 5000 simulerade datamängder med $\Delta t = 1.0$. På y-axeln visas $S(\boldsymbol{X})$ och på x-axeln visas parametrarna som datamängderna simulerats med

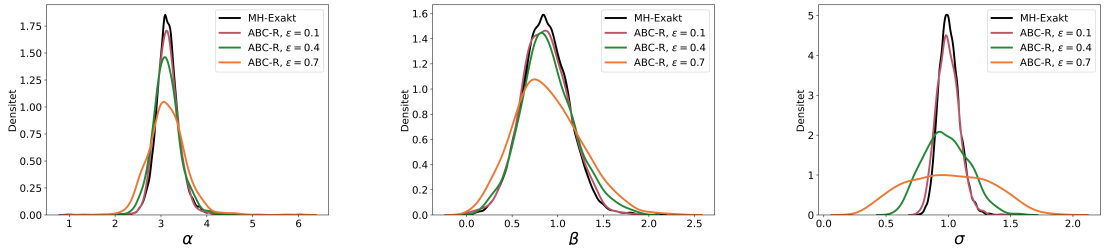
B.6 Resultat



(a) **(b)** **(c)**

Figur 16: $\pi(\boldsymbol{\theta}|\boldsymbol{X}^{obs})$ för MH-EM och $\pi(\boldsymbol{\theta}|\boldsymbol{X}^{obs})$ för MH-Exakt för $\Delta t = [0.1, 0.3, 1.0]$

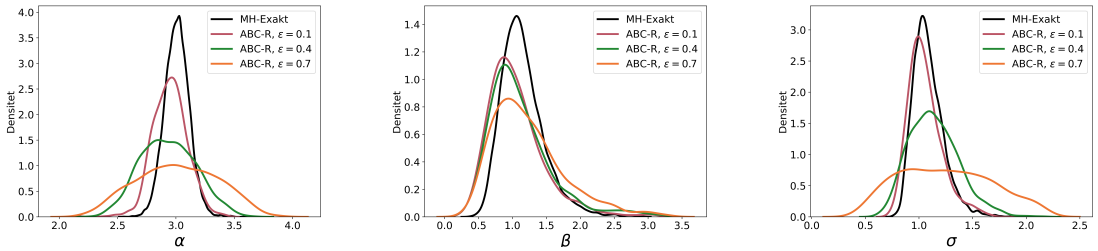
$$W_{2\epsilon=0.1} = 0.091,\ W_{2\epsilon=0.4} = 0.162,\ W_{2\epsilon=0.7} = 0.354$$



(a) **(b)** **(c)**

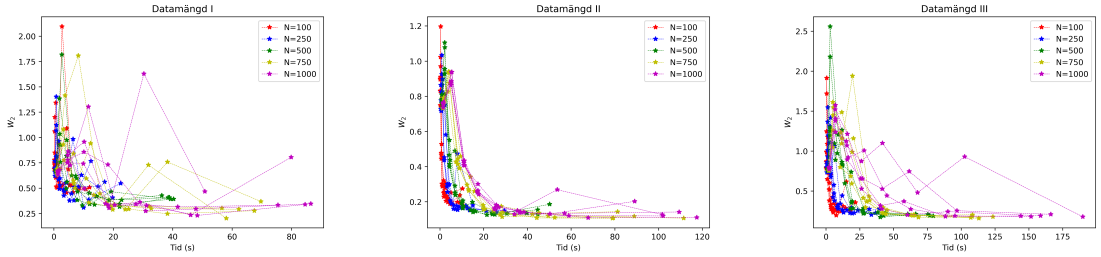
Figur 17: $\pi_\epsilon(\boldsymbol{\theta}|S(\boldsymbol{X}^{obs}))$ genererad av ABC-R med varierande tolerans ϵ jämfört med $\pi(\boldsymbol{\theta}|\boldsymbol{X}^{obs})$ för MH-Exakt där $\boldsymbol{X}^{obs} \in \mathcal{D}_{\text{II}}$ med $\Delta t = 0.3$

$$W_{2\epsilon=0.1} = 0.197,\ W_{2\epsilon=0.4} = 0.337,\ W_{2\epsilon=0.7} = 0.551$$



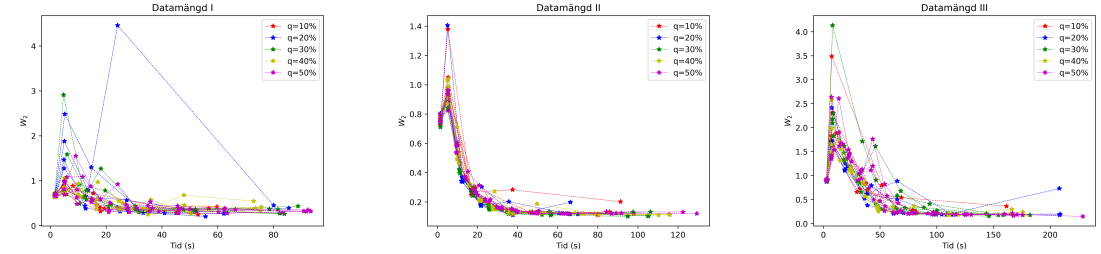
(a) **(b)** **(c)**

Figur 18: $\pi_\epsilon(\boldsymbol{\theta}|S(\boldsymbol{X}^{obs}))$ genererad av ABC-R med varierande tolerans ϵ jämfört med $\pi(\boldsymbol{\theta}|\boldsymbol{X}^{obs})$ för MH-Exakt där $\boldsymbol{X}^{obs} \in \mathcal{D}_{\text{III}}$ med $\Delta t = 1.0$



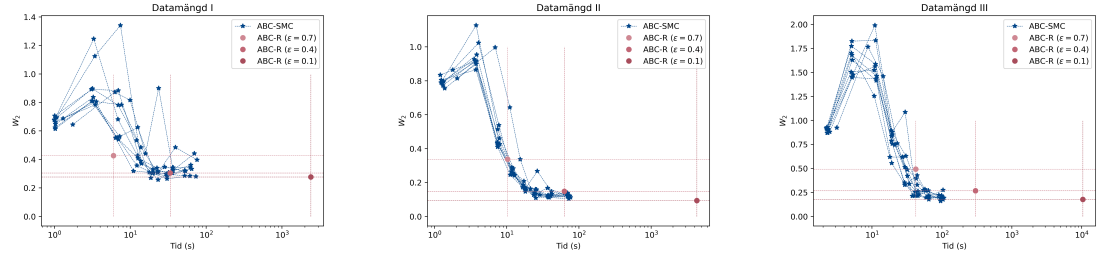
(a) **(b)** **(c)**

Figur 19: Jämförelse av W_2-avstånd mellan olika antal partiklar N som genereras varje omgång för ABC-SMC utförd på samtliga datamängder. För varje N gjordes 5 körningar. Programmet stoppades då acceptanskvoten blev lägre än 1.5%.



(a) **(b)** **(c)**

Figur 20: 5 körningar av ABC-SMC på samtliga datamängder med varje q i listan [10%, 20%, 30%, 40%, 50%]. På x-axeln är tid i sekunder och på y-axeln är W_2-avståndet.



(a) **(b)** **(c)**

Figur 21

Figur 22: 10 körningar av ABC-SMC med $N = 750$, $q = 30\%$ och stoppkriterium: acceptanskvot $< 1.5\%$, samt ABC-R med $\epsilon = [0.7,\ 0.4,\ 0.1] = [0.7, 0.4, 0.1]$ för samtliga datamängder. På y-axeln är W_2-avstånd och på x-axeln är tid (s) i logaritmerad skala

Tabell 4: För varje N i listan [100, 250, 500, 750, 1000] och datamängd sammanställdes medelvärde och standardavvikelse för W_2-måttet och tidsåtgång baserat på fem a posteriori-fördelningar. Datan visualiseras i figur (7).

N	100	250	500	750	1000
D1 W_2	0.498 ± 0.023	0.560 ± 0.117	0.406 ± 0.013	0.289 ± 0.053	0.458 ± 0.181
D2 W_2	0.218 ± 0.033	0.155 ± 0.013	0.148 ± 0.021	0.117 ± 0.014	0.141 ± 0.033
D3 W_2	0.364 ± 0.132	0.237 ± 0.033	0.203 ± 0.011	0.178 ± 0.015	0.189 ± 0.011
D1 Tid (s)	8.611 ± 2.50	17.328 ± 3.533	38.656 ± 1.321	62.816 ± 5.116	75.443 ± 12.887
D2 Tid (s)	9.336 ± 0.824	20.998 ± 6.073	41.984 ± 6.562	87.934 ± 12.284	103.907 ± 9.413
D3 Tid (s)	16.412 ± 3.167	36.716 ± 3.647	71.609 ± 7.100	111.946 ± 6.080	162.001 ± 15.724

C Appendix 3 – källkod

C.1 Simuleringar

```python
import numba as nb
import numpy as np
from math import sqrt, exp
from random import normalvariate


@nb.njit
def OU_Exact(x0: float, N: int, dt: float, theta: np.ndarray):
    """Simulate trajectory of the Ornstein-Uhlenbeck model using exact transition densities.

    Args:
        x0 (float): Initial state.
        n (int): Number of observations (without x0).
        dt (float): Timestep
        theta (np.ndarray): Parameter.

    Returns:
        np.ndarray: Ornstein-Uhlenbeck trajectory.
    """

    # Extract parameters.
    a, b, s = theta

    # Allocate space for the forward trajectory.
    X = np.zeros(N + 1)
    X[0] = x0

    # Incrementally sample from the transition density.
    for i in range(1, N + 1):
        mu = a + (X[i - 1] - a) * exp(-b * dt)
        var = (s**2) * (1 - exp(-2 * b * dt)) / (2 * b)
        X[i] = normalvariate(mu, sqrt(var))
    return X

@nb.njit
def OU_EM( x0: float, N: int, dt: float, theta: np.ndarray):
    """Simulate trajectory of the Ornstein-Uhlenbeck model using Euler-Marayama approximatimated transition densities.

    Args:
        N (int): Number of samples
        theta (ndarray): Parameters
        x0 (float): Initial state
        dt (float): Timestep

    Returns:
        ndarray: Ornstein-Uhlenbeck trajectory.
    """
    X = np.zeros(N + 1)
    X[0] = x0
    a, b, s = theta
    for i in range(N):
        X[i + 1] = (
            X[i]
            + b*(a - X[i])*dt
            + s*np.random.normal()*np.sqrt(dt)
        )
    return X
```

C.2 Wasserstein distance

```python
import numpy as np
import ot


def calc_emd(ref_data_set: np.ndarray, data_set: np.ndarray, p: int = 2):
    """Calculates multivariate Wasserstein distance (Earth Mover's Distance) between two
    multivariate data sets.

    Args:
        ref_data_set (ndarray): Reference data set.
        data_set (ndarray): Data set to compare to reference data set.
        p (int, optional): Square root of squared euclidean distance if 2, Euclidean distance if 1. Defaults to 2.

    Returns:
        float: Wasserstein distance.
    """

    n = 1000

    # Slice data sets to same size
    if n < ref_data_set.shape[0]:
        ref_data_set = ref_data_set[:: int(ref_data_set.shape[0] / n), :]
    if n < data_set.shape[0]:
        data_set = data_set[:: int(data_set.shape[0] / n), :]

    a, b = np.ones((n,)) / n, np.ones((n,)) / n
    if p == 2:
        M = ot.dist(ref_data_set, data_set)
        return np.sqrt(ot.emd2(a, b, M))
    elif p == 1:
        M = ot.dist(ref_data_set, data_set, metric="euclidean")
        return ot.emd2(a, b, M)
```

C.3 Metropolis-Hastings

```python
import numba as nb
import numpy as np
from math import exp, log, sqrt, pi
import time


@nb.njit
def norm_logpdf(x: float, mu: float, sigma: float):
    """Compute the log-pdf of a normal distribution.

    Args:
        x (float): Evaluation point.
        mu (float): Mean.
        sigma (float): Standard deviation.

    Returns:
        float: Log-pdf.
    """
    sigmasquared = sigma * sigma
    return -log(sqrt(2 * pi * sigmasquared)) - ((x - mu) * (x - mu)) / (
        2 * sigmasquared
    )


@nb.njit
def euler_transition_density(X1: float, X2: float, theta: np.ndarray, dt: float):
    """Compute the transition density of the OU process using the Euler-Maruyama
    approximation.

    Args:
        X1 (float): Current state.
        X2 (float): Next state.
        theta (ndarray): Parameters.
        dt (float): Timestep.

    Returns:
        _type_: Transition density from X1 to X2.
    """
    Alpha, Beta, Sigma = theta
    mean = X1 + Beta * (Alpha - X1) * dt
    var = Sigma**2 * dt
    return norm_logpdf(X2, mean, sqrt(var))


@nb.njit
def exact_transition_density(X1: float, X2: float, theta: np.ndarray, dt: float):
    """Compute the exact transition density of the OU process.

    Args:
        X1 (float): Current state.
        X2 (float): Next state.
        theta (ndarray): Parameters.
        dt (float): Timestep.

    Returns:
        _type_: Transition density from X1 to X2.
    """
    Alpha, Beta, Sigma = theta
    mean = Alpha + (X1 - Alpha) * exp(-Beta * dt)
    var = (Sigma**2) / (2 * Beta) * (1 - exp(-2 * Beta * dt))
    return norm_logpdf(X2, mean, sqrt(var))


@nb.njit
def loglik_OU(theta: np.ndarray, X: np.ndarray, dt: float, exact=True):
    """Compute the log-likelihood of the OU process for some parameters.

    Args:
        theta (ndarray): Parameters.
```

```
        X (ndarray): Ornstein-Uhlenbeck trajectory.
        dt (float): Timestep.
        exact (bool, optional): Using exact transition densities if True, using Euler-Maruyama approximation if False. Defaults to True.

    Returns:
        float: Log-likelihood.
    """
    N = len(X)
    f = np.zeros(N - 1)
    for i in range(N - 1):
        x1 = X[i]
        x2 = X[i + 1]
        if exact:
            f[i] = exact_transition_density(x1, x2, theta, dt)
        else:
            f[i] = euler_transition_density(x1, x2, theta, dt)
    return np.sum(f)


@nb.njit
def sample_mvnorm(
    theta: np.ndarray,
    chol: np.ndarray,
):
    """Randomly pick a particle and perturb it by a Gaussian

    Args:
        theta (ndarray): Parameter.
        chol (ndarray): Cholesky decomposition of a covariance matrix.

    Returns:
        ndarray: Parameter proposal.
    """
    # Resample and perturb.
    d = len(theta)
    theta = chol @ np.random.randn(d) + theta
    return theta


@nb.njit
def logpdf_unif(x: float, a: float, b: float):
    """Computes the log of the uniform probability density function.

    Args:
        x (float): evaluation point
        a (float): lower bound
        b (float): upper bound

    Returns:
        float: likelihood of x
    """
    if a <= x <= b:
        return -log(b - a)
    return -np.inf


def MH_OU(
    X: np.ndarray,
    dt: float,
    theta: np.ndarray,
    prior_bounds: np.ndarray,
    I: int = 100000,
    exact=True,
    timecomp=False,
):
    """Metropolis Hastings algorithm for Ornstein-Uhlenbeck process

    Args:
        X (ndarray): Trajectory of the Ornstein-Uhlenbeck process.
```

```
        dt (float): Time step of the process.
        theta (ndarray): Initial guess of the parameters.
        prior_bounds (2x3 ndarray): Prior bounds for the parameters.
        I (integer, optional): Number of iterations. Defaults to 100000.
        exact (bool, optional): Use exact transition densities if True or use Euler-Maruyama approximation if False. Defaults to True.
        timecomp (bool, optional): Record elapsed time. Defaults to False.

    Returns:
        Ix3 ndarray:  Markov chains of the parameters.
    """

    # For adaptive covariance
    p = len(theta)
    eps = 1e-8
    s_d = (2.38**2) / p
    cov = np.diag([0.1] * 3)

    # Cholensky decomposition of covariance matrix
    chol = np.linalg.cholesky(cov)

    # Initilize markov chain
    chain = np.zeros((I, p))
    chain[0] = theta

    # Initilize likelihood
    lik = loglik_OU(theta, X, dt, exact)

    start_time = time.time()
    time_dict = {}
    for i in range(1, I):
        # Generate new theta from candidate distribution
        theta_new = sample_mvnorm(theta, chol)

        # Compute log-likelihood from transition densities
        lik_new = loglik_OU(theta_new, X, dt, exact)

        # Compute prior likelihoods
        prior_old = 0
        prior_new = 0
        for k in range(p):
            prior_old += logpdf_unif(theta[k], prior_bounds[k, 0], prior_bounds[k, 1])
            prior_new += logpdf_unif(
                theta_new[k], prior_bounds[k, 0], prior_bounds[k, 1]
            )

        # Accept next step or stay
        accept_prob = (lik_new + prior_new) - (lik + prior_old)

        # Accept if true, else stay
        if np.random.uniform(0, 1) < exp(accept_prob):
            theta = theta_new
            lik = lik_new

        # Add theta to markov chain
        chain[i] = theta

        # Adaptive covariance
        if i > (I / 10) and i % (I / 100) == 0:
            cov = s_d * np.cov(chain[:i].T) + s_d * eps * np.eye(p, p)
            chol = np.linalg.cholesky(cov)

        # Store elapsed time every 2000 points
        if timecomp and i % 2000 == 0:
            elapsed = time.time() - start_time
            time_dict.update({i: elapsed})
    if not timecomp:
        return chain
    else:
```

```
206             return chain, time_dict
```

C.4 ABC-R

```python
import torch
import numpy as np
import tqdm
from standardSimuleringar import OU_EM
import time


def summarize(x: torch.Tensor, net):
    """Summarizes markov chain using neural network as summary statistics

    Args:
        x (torch.Tensor): Ornstein-Uhlenbeck trajectory
        net (neuralnetworks.MarkovExchangeableNeuralNetwork): Trained neural network

    Returns:
        torch.Tensor: 3x1 tensor with summary statistics for trajectory
    """
    if net:
        with torch.no_grad():
            return net(x)
    else:
        return x


def euclid(a, b):
    """Calculates euclidian distance between two vectors"""
    return np.linalg.norm(np.array(a) - np.array(b))


def RABC(X: np.ndarray, dt: float, tol: float, accepted: int = 1000, nn=None):
    """Rejection ABC algorithm for Ornstein-Uhlenbeck process, using neural network summary statistics

    Args:
        X (ndarray): Ornstein-Uhlenbeck trajectory
        dt (float): Size of time step in trajectory
        accepted (int, optional): Number of accepted samples to return. Defaults to 1000
        tol (float, optional): Tolerance for sample acceptance. Defaults to 0.1.
        nn (_type_, optional): Trained neural network. Defaults to None.

    Returns:
        tuple[ndarray, float]: Approximated posterior distribution and time taken to run algorithm
    """
    theta_chain = np.zeros((accepted, 3))
    theta_sample = np.zeros(3)
    accepted_thetas = 0
    x0 = X[0]
    dt_sim = 0.01  # time step for simulated data
    N_obs = len(X - 1)  # Number of data points in observed data
    N_sim = int(dt * N_obs / dt_sim)  # Number of data points in simulated data

    # Summarize observed data
    X = torch.Tensor(X)
    X_summary = np.exp(summarize(X, nn))

    # Prior bounds
    temp = X_summary.numpy()
    temp_prior = temp.copy()
    prior_a = [temp_prior[0] - 3, temp_prior[0] + 3]
    prior_b = [0, temp_prior[1] + 2]
    prior_s = [0, temp_prior[2] + 1]
    prior_bounds = np.array([prior_a, prior_b, prior_s])

    # Begin algiorithm
    print("Running RABC algorithm with tolerance: ", tol)
    pbar = tqdm.tqdm(total=accepted)
```

```
    t = time.time()
    N_paths = 0
    while accepted_thetas < accepted:
        # Sample proposal theta from uniform prior
        for i in range(3):
            theta_sample[i] = np.random.uniform(prior_bounds[i, 0], prior_bounds[i,
 1])

        # Simulate path from sampled theta
        X_sim = OU_EM(theta=theta_sample, dt=dt_sim, x0=x0, N=N_sim)
        X_sim = torch.Tensor(X_sim[:: int(N_sim / N_obs)])
        # Update number of paths simulated
        N_paths += 1

        # Summarize simulated data
        X_sim_summary = np.exp(summarize(X_sim, nn))

        # Check distance against observed data, accept if below tolerance
        summary_distance = euclid(X_sim_summary, X_summary)
        if summary_distance <= tol:
            theta_chain[accepted_thetas, :] = theta_sample
            accepted_thetas += 1
            pbar.update(1)

    pbar.close()
    tot_time = time.time() - t
    print("RABC algorithm finished.")
    return theta_chain, tot_time, N_paths
```

C.5 ABC-SMC

```python
import torch
import numpy as np
import numba as nb
import random
import tqdm
from numpy.linalg import inv, det, cholesky
from math import log
import time
import ot


def summarize(x, net):
    """Summarize markov chain using neural network as summary statistic.

    Args:
        x (np.ndarray): Ornstein-Uhlenbeck trajectory.
        net (torch.nn.Module): Neural network.

    Returns:
        torch.Tensor: 3x1 tensor with summary statistics for the trajectory.
    """

    x = torch.Tensor(x)
    if net:
        with torch.no_grad():
            return net(x)
    else:
        return x


def euclid(a, b):
    """Euclidean distance between two vectors."""
    return np.linalg.norm(a - b)


def calc_emd(ref_data_set, data_set, p=2):
    """Calculate multivariate Wasserstein distance between two datasets.

    Args:
        ref_data_set (np.ndarray): Reference dataset.
        data_set (np.ndarray): Dataset to compare with reference.
        p (int, optional): Order of the Wasserstein distance. Defaults to 2.

    Returns:
        float: Wasserstein distance."""
    n = ref_data_set.shape[0]
    a, b = np.ones((n,)) / n, np.ones((n,)) / n
    if p == 2:
        M = ot.dist(ref_data_set, data_set)
        return np.sqrt(ot.emd2(a, b, M))
    elif p == 1:
        M = ot.dist(ref_data_set, data_set, metric="euclidean")
        return ot.emd2(a, b, M)



@nb.njit
def prior_pdf(theta,bounds):
    """Prior density.

    Args:
        theta (np.ndarray): Parameters.
        bounds (np.ndarray): Bounds for the prior.

    Returns:
        float: Prior density evaluated at theta."""
    for i in range(len(theta)):
        if not bounds[i][0] < theta[i] < bounds[i][1]:
            return 0
```

```
    return 1  # Weights are normalized


@nb.njit
def sample_mvnorm(
    theta: np.ndarray,
    chol: np.ndarray,
):
    """Randomly pick a particle and perturb it by a Gaussian

    Args:
        theta (ndarray): Parameter.
        chol (ndarray): Cholesky decomposition of a covariance matrix.

    Returns:
        ndarray: Parameter proposal.
    """
    # Resample and perturb.
    d = len(theta)
    theta = chol @ np.random.randn(d) + theta
    return theta


@nb.njit
def multivariate_normal_pdf(x, mean, cov):
    """Multivariate normal density function.

    Args:
        x (np.ndarray): Point at which to evaluate the density (particle).
        mean (np.ndarray): Mean of the multivariate normal (oldtheta[i]).
        cov (np.ndarray): Covariance matrix of last rounds particles (oldCov).

    Returns:
        float: Density evaluated at x."""
    dim = len(x)
    x_minus_mean = x - mean
    cov_inv = inv(cov)
    cov_det = det(cov)
    exponent_term = -0.5 * np.dot(x_minus_mean.T, np.dot(cov_inv, x_minus_mean))

    normalization_term = 1.0 / (np.power(2 * np.pi, dim / 2) * np.sqrt(cov_det))

    return normalization_term * np.exp(exponent_term)

@nb.njit
def compute_nWeight(prior_theta, oldthetas, oldweights, oldCov, prior_bounds):
    """Compute the weight for a particle.

    Args:
        prior_theta (np.ndarray): Particle.
        oldthetas (np.ndarray): Old particles.
        oldweights (np.ndarray): Old weights.
        oldCov (np.ndarray): Covariance matrix of old particles.
        prior_bounds (np.ndarray): Bounds for the prior.

    Returns:
        float: New weight for particle."""
    prior_value = prior_pdf(prior_theta, prior_bounds)
    pdf_values = np.empty(len(oldweights))

    for i in range(len(oldthetas)):
        pdf_values[i] = multivariate_normal_pdf(prior_theta, mean=oldthetas[i], cov
    =oldCov)

    weighted_sum = np.dot(oldweights, pdf_values)
    nWeight = prior_value / weighted_sum

    return nWeight


```

```
def ABCSMC(X_obs, dt_obs, dt_sim, simulator, prior_bounds, N, T,mcmc, nn, omega
    =0.3, SR="acceptance"):
    """ ABC SMC algorithm

    Args:
        X_obs (np.ndarray): Observed trajectory.
        dt_obs (float): Time step of the observed trajectory.
        dt_sim (float): Time step of the simulated trajectory.
        simulator (function): Function that simulates the process.
        prior_bounds (np.ndarray): Bounds for the prior.
        N (int): Number of particles.
        T (int): Number of iterations.
        mcmc (function): MH posterior. (with N points)
        nn (torch.nn.Module): Neural network.
        omega (float, optional): q/100. Defaults to 0.3.
        SR (str, optional): Stopping rule. Defaults to "acceptance".

    Returns:
        SMC posterior, list of: [EMD, simluated paths, time] (for each iteration)
    """

    N_obs = int((len(X_obs)) - 1)
    N_sim = int(dt_obs * N_obs / dt_sim)
    step = int(N_sim / N_obs)
    X0 = X_obs[0]
    X_summary = torch.exp(summarize(X_obs, nn))
    EMD_list = []
    sim_paths_list = []
    time_list = []

    @nb.njit
    def prior():
        theta = []
        for i in prior_bounds:
            param = np.random.uniform(i[0],i[1])
            theta.append(param)
        return np.array(theta)

    I = 100000
    BurnIn = 60000
    if N == 750:
        I = 100000
        BurnIn = 50500
    points = N
    slice_size = int((I - BurnIn) / points)
    mcmc = mcmc[BurnIn::slice_size, :]

    sim_paths = 0
    k = 5
    start_time = time.time()
    for t in range(T):

        if t > 0:
            k = 1
        sim_paths_round = 0
        accepted_thetas = []
        weights = []
        deltas = []
        pbar = tqdm.tqdm(total=k*N)

        while len(accepted_thetas) < k*N:

            if t == 0:
                prior_theta = prior()
                X_sim = simulator(X0, N_sim, dt_sim, prior_theta)
                sim_paths += 1
                sim_paths_round += 1
                X_sim = X_sim[::step]
                X_sim_summary = torch.exp(summarize(X_sim, nn))
                summary_distance = euclid(X_summary,X_sim_summary)
```

```
                accepted_thetas.append(prior_theta)
                deltas.append(summary_distance)
                pbar.update(1)
                nWeight = 1
                weights.append(nWeight)

            if t > 0:
                draws = random.choices(oldthetas, weights=oldweights, k=N)
                draws_array = np.array(draws)
                prior_thetas = np.array([sample_mvnorm(draw, chol) for draw in
    draws_array])

                X_sims = [simulator(X0, N_sim, dt_sim, prior_theta) for prior_theta
     in prior_thetas]
                X_sims = np.array([X_sim[::step] for X_sim in X_sims])
                X_sims_summary = np.array(list(map(lambda x: torch.exp(summarize(x,
     nn)).numpy(), X_sims)))
                summary_distances = np.array([euclid(X_summary, X_sim_summary) for
    X_sim_summary in X_sims_summary])


                accepted_indices = np.where(summary_distances <= epsilon)[0]
                if len(accepted_indices) > N:
                    accepted_indices = accepted_indices[:N]

                accepted_prior_thetas = prior_thetas[accepted_indices]
                accepted_deltas = summary_distances[accepted_indices]

                for prior_theta, delta in zip(accepted_prior_thetas,
    accepted_deltas):
                    accepted_thetas.append(prior_theta)
                    deltas.append(delta)
                    pbar.update(1)
                    nWeight = compute_nWeight(prior_theta, oldthetas, oldweights,
    oldCov, prior_bounds)
                    weights.append(nWeight)
                sim_paths += len(prior_thetas)
                sim_paths_round += len(prior_thetas)

        pbar.close()
        s = sum(weights) ##normalise sum to 1 done
        weights = [w/s for w in weights]

        if t == 0:
            accepted_thetas = [x for _,x in sorted(zip(deltas,accepted_thetas), key
    =lambda pair: pair[0])]
            accepted_thetas = accepted_thetas[:N]
            deltas.sort()
            epsilon = deltas[round(len(deltas)*N/(k*N))]
            weights = weights[:N]
        else:
            deltas.sort()
            epsilon = deltas[round(len(deltas)*omega)]

        oldweights = np.array(weights)
        oldthetas = np.array(accepted_thetas)
        oldCov = np.cov(np.array(accepted_thetas).T)
        chol = np.linalg.cholesky(oldCov)

        posterior = np.array(random.choices(accepted_thetas, weights=weights, k=N))
        EMD = calc_emd(mcmc, posterior)
        EMD_list.append(EMD)
        sim_paths_list.append(sim_paths)
        acceptance_rate = k*N/sim_paths_round
        passed_time = time.time() - start_time
        time_list.append(passed_time)

        print("Round:", t,"| Time: ", passed_time, "| Simulated paths: ", sim_paths
    ,
              "| Acceptance rate = ", acceptance_rate, "| EMD: ", EMD,"| Epsilon: "
```

```
    , epsilon)
        if SR == "acceptance":
            if acceptance_rate < 0.015:
                break
        elif SR == "paths":
            if sim_paths > 100000:
                break
    EMD_paths_time = np.column_stack((EMD_list, sim_paths_list, time_list))
    return posterior, EMD_paths_time
```

C.6 PEN

```python
import torch
from typing import Optional
import pytorch_lightning as pl
from torch.utils.data import DataLoader, TensorDataset


class PENDataModule(pl.LightningDataModule):
    def __init__(
        self,
        train_paths: torch.Tensor,
        train_params: torch.Tensor,
        val_paths: torch.Tensor,
        val_params: torch.Tensor,
    ):
        super().__init__()
        self.train_paths = train_paths
        self.train_params = train_params
        self.val_paths = val_paths
        self.val_params = val_params

    def setup(self, stage: str):
        self.train = TensorDataset(self.train_paths, self.train_params)
        self.validation = TensorDataset(self.val_paths, self.val_params)

    def train_dataloader(self):
        return DataLoader(self.train, batch_size=1024, num_workers=1)

    def val_dataloader(self):
        return DataLoader(self.validation, batch_size=256, num_workers=1)


class MarkovExchangeableNeuralNetwork(pl.LightningModule):
    def __init__(self, input_shape: int = 2, nparams: int = 3, hidden_size: int =
    100):
        super().__init__()
        self.l1 = torch.nn.Linear(input_shape, hidden_size)
        self.l2 = torch.nn.Linear(hidden_size, hidden_size)
        self.l3 = torch.nn.Linear(hidden_size + 1, hidden_size)
        self.l4 = torch.nn.Linear(hidden_size, nparams)
        self.loss = torch.nn.MSELoss()

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        if len(x.shape) == 1:
            x0 = x[0]
            x = x.unfold(0, 2, 1)
            x = self.l1(x).clamp(min=0)
            x = self.l2(x)
            x = torch.mean(x, 0)
            x = self.l3(torch.cat((x0.reshape(1), x))).clamp(min=0)
            x = self.l4(x)
        else:
            x = x.unfold(1, 2, 1)
            x0 = x[:, 0, 0]
            length = x0.shape[0]
            x0 = torch.reshape(x0, (length, 1))
            x = self.l1(x).clamp(min=0)
            x = self.l2(x)
            x = torch.mean(x, 1)
            x = self.l3(torch.cat((x0, x), 1)).clamp(min=0)
            x = self.l4(x)
        return x

    def configure_optimizers(self):
        return torch.optim.Adam(self.parameters())

    def training_step(self, batch, batch_idx):
        x, y = batch
        output = self(x)
        J = self.loss(output, y)
```

```
        self.log("train_loss", J)
        return J

    def validation_step(self, batch, batch_idx):
        x, y = batch
        output = self(x)
        J = self.loss(output, y)
        self.log("val_loss", J)
        return J
```