Masterproef scriptie

Recursive Monte Carlo for linear ODEs

Auteur: *Isidoor Pinillo Esquivel*

Promotor: *Wim Vanroose*

Academiejaar 2022-2023

Contents

Abstract

This thesis explores applying recursive Monte Carlo for solving linear ordinary differential equations with a vision towards partial differential equations. The proposed algorithms capitalize on the appropriate combination of Monte Carlo techniques. These Monte Carlo techniques get introduced with examples and code.

1 Introduction

1.1 Related Work

The primary motivating paper for this work is the work by Sawhney et al. (2022) [Saw+22], which introduces the Walk-on-Sphere (WoS) method for solving second-order elliptic PDEs with varying coefficients and Dirichlet boundary conditions. Their techniques have shown high accuracy even in the presence of geometrically complex boundary conditions. We were inspired to apply the underlying mechanics of these Monte Carlo (MC) techniques to ODEs to explore parallel in time and the possibility of extending their techniques to other types of PDEs.

We made an interactive data map of the literature read mainly in the function of this thesis available at `https://huggingface.co/spaces/ISIPINK/zotero_map`. It may take 10 seconds to load.

The latest paper that we found on an unbiased Initial Value Problem (IVP) solver is by Ermakov and Smilovitskiy's 2021 [ES21] that study an unbiased method for a Cauchy problem for large systems of linear ODEs for describing queuing systems. Similarly to us, they base their solver on Volterra integral equations.

Other literature is a bit further away. The most important fields we draw from are:

- rendering and WoS/first passage literature which contain many practical recursive MC techniques,
- stochastic gradient descent literature that is connected through continuous gradient descent see [Hua+17] for an introduction,
- Information-Based Complexity (IBC) literature, which was unexpected to us, there are some interesting biased algorithms applied on ODEs that achieve optimal IBC rates for RMSE for some smoothness classes, similar to us Daun's 2011 [Dau11] uses control variates to achieve optimal IBC.

A recurrent theme in these fields is that optimal IBC algorithms and unbiased algorithms are of theoretical importance.

1.2 Contributions

A significant part of this thesis is dedicated to informally introducing Recursive Monte Carlo (RMC) and applying variance reduction techniques for ODEs.

The key contribution is an unbiased MC method see example 3.2.4 for linear IVPs for systems of ODEs by using recursion in recursion and variance reduction techniques. We conjecture that this method achieves optimal IBC.

2 Background

2.1 Monte Carlo Integration

In this subsection, we review basic MC theory.

Notation 2.1.1 (Random Variables)
Random variables (RVs) will be denoted with capital letters, e.g., X, Y or Z.

MC integration is any way to use random sampling to estimate an integral.

Definition 2.1.2 (Uniform Monte Carlo Integration)
We define uniform MC integration of $f : \mathbb{R}^n \to \mathbb{R}^m$ over $\Omega \subset \mathbb{R}^n$ as an estimation of the average of $f(S)$, with $S \sim \text{Uniform}(\Omega)$. Combined with the Best Linear Unbiased Estimators (BLUEs) for the average, MC Integration in that case, can be summarized in the following formula:

$$\int_\Omega f(s)ds \approx \frac{1}{n}\sum_{i=1}^{n} f(S_j), \tag{1}$$

where n is the amount of samples used and S_j i.i.d. Uniform(Ω).

Because estimators are random variables (RVs), the cost and error are also random variables. In most cases, obtaining these RVs is difficult to impossible. Directly comparing estimators based on these RVs can be challenging; there is no Pareto front. Instead, comparisons can be made using statistics.

Accuracy comparisons between estimators are typically conducted with (root-)mean-square error (RMSE).

Definition 2.1.3 (Root-Mean-Square Error)
We define the Root-Mean-Square Error (RMSE) of an estimator $\tilde{\theta}$ for θ as follows:

$$\text{RMSE}(\tilde{\theta}) = \sqrt{E[||\tilde{\theta} - \theta||_2^2]}. \tag{2}$$

Even comparisons based on RMSE can be counterintuitive; consider Stein's paradox, for example. We will almost always limit ourselves to 1-dimensional unbiased estimators, making MSE equivalent to variance. Estimating variance is simple and can be used to calculate confidence intervals using Chebyshev's inequality or an approximate normal distribution argument.

Average floating point operations or time per simulation are common cost statistics. It may be useful to consider 'at risk' (analogous to 'value at risk') in terms of memory or wall time.
If we limit ourselves to (1) with a big sample size and finite variance assumption on error and simulation time. Simulations can be computed in parallel, making them

well-suited for GPU implementation. When these assumptions are close to optimal, they are very practical. Under these assumptions, there is a linear trade-off between average simulation time and variance which motivate the definition of MC efficiency for comparing estimators.

Definition 2.1.4 (Monte Carlo Efficiency)
Define MC efficiency of an estimator F as follows:

$$\epsilon[F] = \frac{1}{\text{Var}(F)T(F)}, \tag{3}$$

with T the average simulation time.

Related Work 2.1.5 (Monte Carlo Efficiency)
For a reference see [Vea97] page 45.

For smooth 1 dimensional integration, the linear trade-off between variance and average simulation time is not even close to optimal see theorem 2.4.6.
When it comes to comparing better trade-offs, Information-Based Complexity (IBC) is often employed. It's worth mentioning that IBC primarily serves as a qualitative measure and does not necessarily imply the practicality of an algorithm. While we won't delve into a rigorous definition of IBC here, it plays a vital role in assessing the efficiency of algorithms.

Definition 2.1.6 (Information-Based Complexity)
IBC is a way to describe asymptotically (for increasing accuracy/function calls) the trade-off between the average amount of function calls (information) needed and accuracy.

Example 2.1.7 (IBC of (1))
In (1) the function calls trades of linearly with variance. For n function calls, the RMSE $= O\left(\frac{1}{\sqrt{n}}\right)$ or equivalently, if we want a RMSE of ε we would need $O\left(\frac{1}{\varepsilon^2}\right)$ function calls.

2.2 Recursive Monte Carlo

In this subsection, we introduce Recursive Monte Carlo (RMC) with the following initial value problem:

$$y_t = y, \quad y(0) = 1. \tag{4}$$

By integrating both sides of (4), we obtain:

$$y(t) = 1 + \int_0^t y(s)ds. \tag{5}$$

(5) represents a recursive integral equation, specifically, a linear Volterra integral equation of the second type. By estimating the recursive integral in (5) using MC, we derive the following estimator:

$$Y(t) = 1 + ty(Ut). \tag{6}$$

Notation 2.2.1 (U)
We will be using the uniform distribution very often. So we abbreviate it.

$$U \sim \text{Uniform}(0, 1). \tag{7}$$

If y is well-behaved, then $E[Y(t)] = y(t)$. However, we cannot directly simulate $Y(t)$ without access to $y(s)$ for $s < t$. Nevertheless, we can replace y with an unbiased estimator without affecting $E[Y(t)] = y(t)$, by the law of total expectation ($E[X] = E[E[X|Z]]$). By replacing y with Y itself, we obtain a recursive expression for Y:

$$Y(t) = 1 + tY(Ut). \tag{8}$$

(8) is a Recursive Random Variable Equation (RRVE).

Definition 2.2.2 (Recursive Random Variable Equation (RRVE))
A Recursive Random Variable Equation (RRVE) is an equation that defines a family of random variables in terms of itself.

If one were to try to simulate Y with (8), it would recurse indefinitely. To stop the recursion approximate $Y(t) \approx 1$ near $t = 0$ introducing minimal bias. Later, we will discuss Russian roulette; see definition 2.3.2, which can be used as an unbiased stopping mechanism.

Python Code 2.2.3 (implementation of (8))

```python
from random import random as U
def Y(t, eps): return 1 + t*Y(U()*t, eps) if t > eps else 1
def y(t, eps, nsim):
    return sum(Y(t, eps) for _ in range(nsim))/nsim
print(f"y(1) approx {y(1,0.01,10**3)}")
# y(1) approx 2.710602603240193
```

To gain insight into the realizations of a RRVE, it can be helpful to plot all recursive calls $(t, Y(t))$, as shown in Figure 1 for this implementation.

2.3 Modifying Monte Carlo

In this subsection, we discuss techniques for modifying RRVEs in a way that preserves the expected value of the solution while acquiring more desirable properties. These techniques are only effective when applied smartly by using prior information about the problem or computational costs.

We will frequently interchange RVs with the same expected values. This is why we introduce the following notation.

Notation 2.3.1 ($\cong$)

$$X \cong Y \iff E[X] = E[Y].$$

Russian roulette is a MC technique commonly employed in rendering algorithms. The concept behind Russian roulette is to replace a RV with a less computationally expensive approximation sometimes.

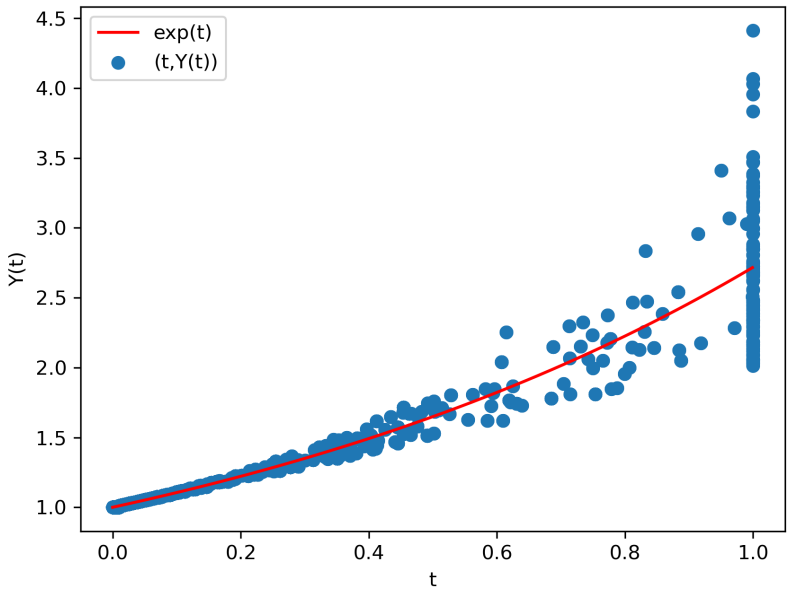

Figure 1: Recursive calls of code 2.2.3

Definition 2.3.2 (Russian roulette)
We define Russian roulette on X with free parameters $Y_1 \cong Y_2$, $p \in [0,1]$ and U independent of Y_1, Y_2, X as follows:

$$X \cong \begin{cases} \frac{1}{p}(X - (1-p)Y_1) & \text{if } U < p \\ Y_2 & \text{else} \end{cases}. \tag{9}$$

Notation 2.3.3 ($B(p)$)
Often Russian roulette will be used with $Y_1 = Y_2 = 0$. In that case, we use Bernoulli variables to shorten notation.

$$B(p) \sim \text{Bernoulli}(p) = \begin{cases} 1 & \text{if } U < p \\ 0 & \text{else} \end{cases}. \tag{10}$$

Example 2.3.4 (Russian roulette)
Let us consider the estimation of $E[Z]$, where Z is defined as follows:

$$Z = U + \frac{f(U)}{1000}. \tag{11}$$

Here, $f : \mathbb{R} \to [0,1]$ is an expensive function to compute. Directly estimating $E[Z]$ would involve evaluating f for each sample, which can be computationally costly. To address this, we can modify Z to:

$$Z \cong U + B\left(\frac{1}{100}\right)\frac{f(U)}{10}. \tag{12}$$

This requires calling f on average once every 100 samples. This significantly reduces the computational burden while increasing the variance slightly thereby increasing

the MC efficiency.

Related Work 2.3.5 (example 2.3.4)
In example **??**, it is also possible to estimate the expectations of the 2 terms of Z separately. Given the variances and computational costs of both terms, you can calculate the asymptotically optimal division of samples for each term. However, this is no longer the case with RMC. In [Rat+22], a method is presented to estimate the optimal Russian roulette/splitting factors for rendering.

Example 2.3.6 (Russian roulette on (8))
To address the issue of indefinite recursion in (8), Russian roulette can be employed by approximating the value of Y near $t = 0$ with 1 sometimes. Specifically, we replace the coefficient t in front of the recursive term with $B(t)$ when $t < 1$. The modified recursive expression for $Y(t)$ becomes:

$$y(t) \cong Y(t) = \begin{cases} 1 + B(t)Y(Ut) & \text{if } t < 1 \\ 1 + tY(Ut) & \text{else} \end{cases}. \tag{13}$$

Python Code 2.3.7 (implementation of (13))

```python
from random import random as U
def Y(t):
    if t>1: return 1 + t*Y(U()*t)
    return 1 + Y(U()*t) if U() < t else 1
def y(t, nsim): return sum(Y(t) for _ in range(nsim))/nsim
print(f"y(1) approx {y(1,10**3)}")
# y(1) approx 2.698
```

Interestingly, $\forall t \leq 1 : Y(t)$ is the number of recursion calls to sample $Y(t)$ such that the average number of recursion calls equals e^t.

Splitting is a technique that has almost the reverse effect of Russian roulette. Instead of reducing the number of simulations of a RV as Russian roulette does, we increase it by using more samples (i.e. splitting the sample) which reduces the variance.

Definition 2.3.8 (splitting)
Splitting X refers to utilizing multiple $X_j \sim X$ (not necessarily independent) to reduce variance by taking their average:

$$X \cong \frac{1}{N} \sum_{j=1}^{N} X_j. \tag{14}$$

Splitting the recursive term in a RRVE can result in additive branching recursion, necessitating cautious management of terminating the branches promptly to prevent exponential growth in computational complexity. To accomplish this, termination strategies that have been previously discussed can be employed. Subsequently, we will explore the utilization of coupled recursion as a technique to mitigate additive branching recursion in RRVEs (see example 3.1.7).

Example 2.3.9 (splitting on (13))

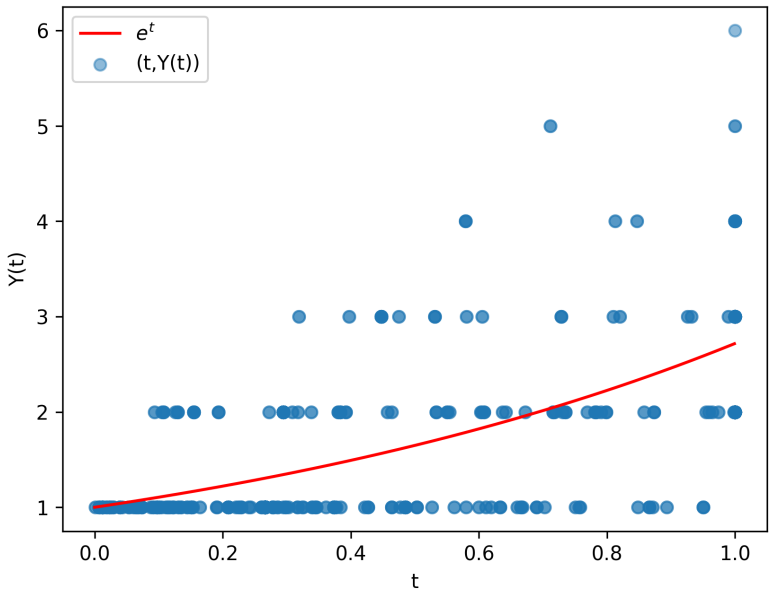

Figure 2: Recursive calls $(t, Y(t))$ of code 2.3.7

We can "split" the recursive term of (13) into two parts as follows:

$$y(t) \cong Y(t) = \begin{cases} 1 + \frac{B(t)}{2}(Y_1(Ut) + Y_2(Ut)) & \text{if } t < 1 \\ 1 + \frac{t}{2}(Y_1(Ut) + Y_2(Ut)) & \text{else} \end{cases}. \tag{15}$$

where $Y_1(t)$ and $Y_2(t)$ are i.i.d. with $Y(t)$.

Python Code 2.3.10 (implementation of (15))

```
from random import random as U
def Y(t):
    u = U()
    if t > 1: return 1 + t*(Y(u*t)+Y(u*t))/2
    return 1 + (Y(u*t)+Y(u*t))/2 if U() < t else 1
def y(t, nsim): return sum(Y(t) for _ in range(nsim))/nsim
print(f"y(1) approx {y(1,10**3)}")
# y(1) approx 2.73747265625
```

Definition 2.3.11 (control variates)
Define control variating $f(X)$ with $\tilde{f}$ an approximation of f as:

$$f(X) \cong f(X) - \tilde{f}(X) + E[\tilde{f}(X)]. \tag{16}$$

Note that control variating requires the evaluation of $E[\tilde{f}(X)]$. When this is estimated we refer to it as 2-level MC.

Example 2.3.12 (control variate on (8))
To create a control variate for (8) that effectively reduces variance, we employ the approximation $y(t) \approx \tilde{y} = 1 + t$ and define the modified recursive term as follows:

$$Y(t) = 1 + E[\tilde{y}(Ut)] + t(Y(Ut) - \tilde{y}(Ut)) \tag{17}$$

$$= 1 + t + \frac{t^2}{2} + t(Y(Ut) - 1 - Ut). \tag{18}$$

Note that while we could cancel out the constant term of the control variate, doing so would have a negative impact on the Russian roulette implemented later.

Python Code 2.3.13 (implementation of example 2.3.12)

```
from random import random as U
def Y(t):
    u = U()
    if t > 1: return 1+t**2/2 + t*(Y(u*t)-u*t)
    return 1 + t + t**2/2 + (Y(u*t)-1-u*t if U() < t else 0)
def y(t, nsim): return sum(Y(t) for _ in range(nsim))/nsim
print(f"y(1) approx {y(1,10**3)}")
# y(1) approx 2.734827303480301
```

Related Work 2.3.14 (MC modification)
For further reference on these techniques see [Vea97].

2.4 Monte Carlo Trapezoidal Rule

In this subsection, we introduce a MC trapezoidal rule that exhibits similar convergence behavior to the methods discussed later. The MC trapezoidal rule is essentially a regular Monte Carlo method enhanced with control variates based on the trapezoidal rule.

Definition 2.4.1 (MC trapezoidal rule)
We define the MC trapezoidal rule for approximating the integral of function f over the interval $[x, x+\Delta x]$ with a Russian roulette rate l and $\tilde{f}$ the linear approximation of f corresponding to the trapezoidal rule as follows:

$$\int_x^{x+\Delta x} f(s)ds \tag{19}$$

$$= \int_x^{x+\Delta x} \tilde{f}(s)ds + \int_x^{x+\Delta x} f(s) - \tilde{f}(s)ds \tag{20}$$

$$= \Delta x \frac{f(x) + f(x+\Delta x)}{2} + E\left[f(S) - \tilde{f}(S)\right] \tag{21}$$

$$\cong \Delta x \frac{f(x) + f(x+\Delta x)}{2} + \Delta x l B\left(\frac{1}{l}\right)\left(f(S) - f(x) - \frac{S-x}{\Delta x}\left(f(x+\Delta x) - f(x)\right)\right), \tag{22}$$

where $S \sim \text{Uniform}(x, x+\Delta x)$.

Lemma 2.4.2 (RMSE MC Trapezoidal Rule)

The MC trapezoidal rule for a twice differentiable function has

$$\text{RMSE} = O\left(\Delta x^3\right). \tag{23}$$

Proof. Start from (22). The MSE is the variance so we can ignore addition by constants.

$$\text{MSE} = \text{Var}\left(\Delta x l B\left(\frac{1}{l}\right)\left(f(S) - f(x) - \frac{S-x}{\Delta x}\left(f(x+\Delta x) - f(x)\right)\right)\right) \tag{24}$$

We substitute $S = \Delta x U + x$ and then apply Taylor's theorem finishing the proof:

$$\begin{aligned}
\text{MSE} &= \text{Var}\left(\Delta x l B\left(\frac{1}{l}\right)\left(f(\Delta x U + x) - f(x) - U\left(f(x+\Delta x) - f(x)\right)\right)\right) && (25)\\
&= \text{Var}\left(\Delta x l B\left(\frac{1}{l}\right)\left(U\Delta x f'(x) + \frac{U^2\Delta x^2}{2} f''(Z_1) - U\left(\Delta x f'(x) + \Delta x^2 f''(z_2)\right)\right)\right) && (26)\\
&= \text{Var}\left(\Delta x l B\left(\frac{1}{l}\right)\left(\frac{U^2\Delta x^2}{2} f''(Z_1) - \frac{U\Delta x^2}{2} f''(z_2)\right)\right) && (27)\\
&= \Delta x^6 \text{Var}\left(l B\left(\frac{1}{l}\right)\left(\frac{U^2}{2} f''(Z_1) - \frac{U}{2} f''(z_2)\right)\right), && (28)
\end{aligned}$$

for some $Z_1 \in [x, S], z_2 \in [x, x+\Delta x]$. The variance term is bounded because the variance of a bounded RV is bounded. Note that the proof doesn't rely on Russian roulette ($l = 1$). □

Related Work 2.4.3 (proof 2.4.2)
An easier to generalize proof to other types of control variates can be made by using "separation of the main part" see lemma 4 of [MH].

Definition 2.4.4 (composite MC trapezoidal rule)
Define the composite MC trapezoidal rule for approximating the integral of function f over the interval $[a, b]$ with a uniform grid $(x_j) = \text{linspace}(a, b, n)$ with n intervals and a Russian roulette rate l as follows:

$$\begin{aligned}
\int_a^b f(s)ds \cong \Delta x \sum_{j=1}^{n} &\frac{f(x_j) + f(x_j + \Delta x)}{2} \\
&+ lB\left(\frac{1}{l}\right)\left(f(S_j) - f(x_j) - \frac{S_j - x_j}{\Delta x}(f(x_j + \Delta x) - f(x_j))\right),
\end{aligned} \tag{29}$$

where $S_j \sim \text{Uniform}(x, x + \Delta x)$.

Python Code 2.4.5 (implementation of (29))
We implement (29) for $\int_0^1 e^s ds$.

```python
from random import random as U
from math import exp
import numpy as np
def f(x): return exp(x)
def trapezium(n): return sum((f(j/n)+f((j+1)/n))/2
    for j in range(n))/n
def MCtrapezium(n, l=100):
    sol = 0
    for j in range(n):
        if U()*l < 1:
            x, xx = j/n, (j+1)/n
            S = x + U()*(xx-x)  # \sim Uniform(x,xx)
            sol += l*(f(S)-f(x)-(S-x)*(f(xx)-f(x))*n)/n
    return sol+trapezium(n)
def exact(a, b): return exp(b)-exp(a)
def error(s): return (s-exact(0, 1))/exact(0, 1)
print(f"  error:{error(trapezium(10000))}")
print(f"MCerror:{error(MCtrapezium(10000,100))}")
#   error:8.333344745642098e-10
# MCerror:-1.5216231703870405e-10
```

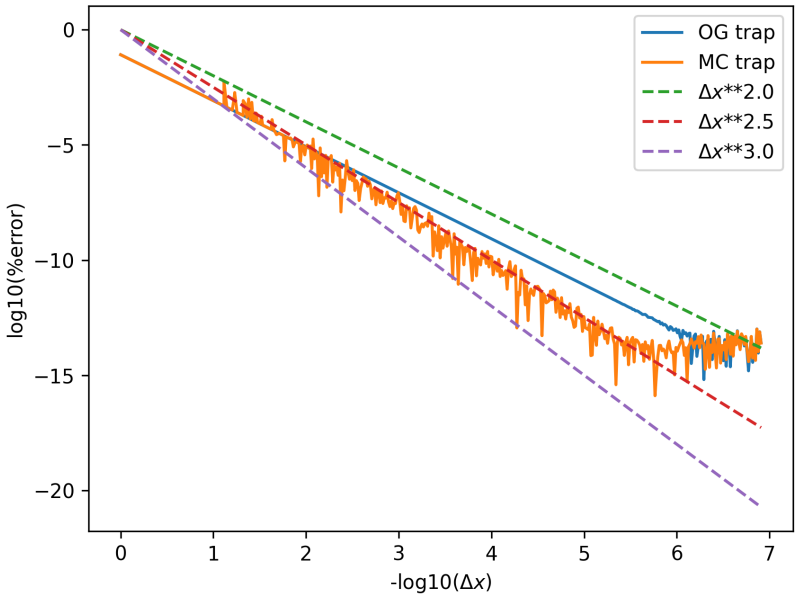

Figure 3: Log-log plot of the error of (29) for $\int_0^1 e^s ds$ *with* $l = 100$*. At floating point accuracy, the convergence ceases.*

Figure 3 suggests that the order of convergence of RMSE of the composite MC trapezoidal rule is better by 0.5 than the normal composite trapezoidal rule. The MC trapezoidal rule has on average $\frac{1}{l}$ more function calls than the normal trapezoidal rule. For the composite rule with n intervals, there are Binomial$(n, \frac{1}{l})$ additional function calls (repeated Bernoulli experiments). So they use up to a constant the same amount of function calls on average. This means an advantage in IBC measured in RMSE.

Theorem 2.4.6 (RMSE Composite Trapezoidal MC Rule)
The composite trapezoidal MC rule wth n intervals for a twice differentiable function has

$$\text{RMSE} = O\left(\frac{1}{n^{2.5}}\right). \tag{30}$$

The proof uses lemma 2.4.2 and is similar to the proof of the normal trapezoidal rule with the main difference being the accumulation of "local truncation" errors into "global truncation" error. Normally there is a loss of one order but the MC trapezoidal loses only a half order because the accumulation happens in variance instead of bias.

$$\sqrt{\text{Var}\left(\sum_{j=1}^{n} \Delta x^3 U_j^2\right)} = \Delta x^3 \sqrt{\sum_{j=1}^{n} \text{Var}(U_j^2)} \tag{31}$$

$$= \Delta x^3 \sqrt{n \text{Var}(U^2)} \tag{32}$$

$$= O(\Delta x^{2.5}). \tag{33}$$

Note that the meaning of a bound on the error that behaves as $O(\Delta x^2)$ and a bound on the RMSE that behaves as $O(\Delta x^2)$ is different. A bound on the error implies a bound on the RMSE, but the reverse is not true.

Related Work 2.4.7 (Monte Carlo Trapezoidal Rule)
With Stein's paradox, it is always possible to bias the composite MC trapezoidal rule to achieve lower RMSE. The optimal IBC for the deterministic, random and quantum case are known see [HN01] for some smoothness classes.

2.5 Unbiased Non-Linearity

In this subsection, we present techniques for handling polynomial non-linearity. The backbone for this is using independent samples $y^2 \cong Y_1 Y_2$ with Y_1 independent of Y_2 and $Y_1 \cong Y_2 \cong y$.

Example 2.5.1 ($y_t = y^2$)
Consider the following ODE:

$$y_t = y^2, \quad y(1) = -1. \tag{34}$$

The solution to this equation is given by $y(t) = -\frac{1}{t}$. By integrating both sides of (34), we obtain the following integral equation:

$$y(t) = -1 + \int_1^t y(s)y(s)ds. \tag{35}$$

To estimate the recursive integral in (34), we use i.i.d. $Y_1, Y_2 \sim Y$ in following RRVE:

$$y(t) \cong Y(t) = -1 + (t-1)Y_1(S)Y_2(S), \tag{36}$$

where $S \sim \text{Uniform}(1, t)$. Branching RRVEs are typical when dealing with non-linearity.

Python Code 2.5.2 (implementation of example 2.5.1)

```python
from random import random as U
def Y(t):
    if t>2: raise Exception("doesn't support t>2")
    S = U()*(t-1)+1
    # Y(u)**2 != Y(u)*Y(u) !!!
    return -1 + Y(S)*Y(S) if U()<t-1 else -1
def y(t, nsim): return sum(Y(t) for _ in range(nsim))/nsim
print(f"y(2) approx {y(2,10**3)}")
# y(2) approx -0.488
```

In this implementation $Y(t)$ only takes values $\{-1, 0\}$.

Example 2.5.3 ($e^{E[X]}$)

$e^{\int x(s)ds}$ is a common expression encountered when studying ODEs. In this example, we demonstrate how you can generate unbiased estimates of $e^{E[X]}$ with simulations of X. The Taylor series of e^x is:

$$e^{E[X]} = \sum_{n=0}^{\infty} \frac{E^n[X]}{n!} \tag{37}$$

$$= 1 + \frac{1}{1}E[X]\left(1 + \frac{1}{2}E[X]\left(1 + \frac{1}{3}E[X]\left(1 + ...\right)\right)\right). \tag{38}$$

Change the fractions of (38) to Bernoulli processes and replace all X with independent $X_j \cong X$.

$$e^{E[X]} = E\left[1 + B\left(\frac{1}{1}\right)E[X_1]\left(1 + B\left(\frac{1}{2}\right)E[X_2]\left(1 + B\left(\frac{1}{3}\right)E[X_3]\left(1 + ...\right)\right)\right)\right] \tag{39}$$

$$\cong 1 + B\left(\frac{1}{1}\right)X_1\left(1 + B\left(\frac{1}{2}\right)X_2\left(1 + B\left(\frac{1}{3}\right)X_3\left(1 + ...\right)\right)\right). \tag{40}$$

Sampling (40) requires a finite amount of samples from X_j's with probability 1.

Related Work 2.5.4 (example 2.5.3)

NVIDIA has a great paper on optimizing example 2.5.3 [Ket+21].

2.6 Recursion

In this subsection, we discuss recursion-related techniques.

Technique 2.6.1 (coupled recursion)

The idea behind coupled recursion is sharing recursion calls of multiple RRVEs for simulation. This does make them dependent.

Example 2.6.2 (coupled recursion)

Consider calculating the sensitivity of following ODE to a parameter a:

$$y_t = ay, y(0) = 1 \Rightarrow \tag{41}$$
$$\partial_a y_t = y + a\partial_a y \tag{42}$$

Turn (41) and (42) into RRVEs. To emphasize that they are coupled, that they should recurse together we write them in a matrix equation:

$$\begin{bmatrix} y(t) \\ \partial_a y(t) \end{bmatrix} \cong \begin{bmatrix} Y(t) \\ \partial_a Y(t) \end{bmatrix} = X(t) = \begin{bmatrix} 1 \\ 0 \end{bmatrix} + t \begin{bmatrix} a & 0 \\ 1 & a \end{bmatrix} X(Ut). \tag{43}$$

Observe how this eliminates the additive branching recursion present in (42).

Python Code 2.6.3 (implementation of (43))

```
from random import random as U
import numpy as np
def X(t, a):  # only supports t<1
    q, A = np.array([1, 0]), np.array([[a, 0], [1, a]])
    return q + A @ X(U()*t, a) if U() < t else q
def sol(t, a, nsim): return sum(X(t, a) for _ in
   range(nsim))/nsim
print(f"x(1,1) = {sol(1,1,10**3)}")
# x(1,1) = [2.7179 2.7104]
```

Related Work 2.6.4 (coupled recursion)

Example 2.6.2 is inspired by [VSJ21]. [VSJ21] propose an efficient unbiased backpropagation algorithm for rendering.

Technique 2.6.5 (recursion in recursion)

Recursion in recursion is like proving an induction step of an induction proof with induction. Recursion in recursion uses an inner recursion in the outer recursion.

Related Work 2.6.6 (recursion in recursion)

Beautiful examples of recursion in recursion are the next flight variant of WoS in [Saw+22] and epoch-based algorithms in optimization [GH21].

Most programming languages do support recursion, but it often comes with certain limitations such as maximum recursion depth and potential performance issues. There are multiple ways to implement recursion we will discuss tail recursion and do an example using a stack.

Technique 2.6.7 (non-branching tail recursion)

Tail recursion involves reordering all operations so that almost no operation needs to happen after the recursion call. This allows us to return the answer without retracing all steps when we reach the last recursion call and it can achieve similar speeds to a forward implementation.

The non-branching recursion presented in the RRVEs can be implemented with tail recursion due to the associativity of all operations ($(xy)z = x(yz)$) involved.

Python Code 2.6.8 (tail recursion on (43))

We implement (43) but this time with tail recursion. We collect addition operations in a vector *sol* and multiplication in a matrix W. W may be referred to as accumulated weight or throughput.

```python
from random import random as U
import numpy as np
def X(t, a) -> np.array:
    q, A = np.array([1.0, 0.0]), np.array([[a, 0.0], [1.0, a]])
    sol, W = np.array([1.0, 0.0]), np.identity(2)
    while U() < t:
        W = W @ A if t < 1 else t * W @ A
        sol += W @ q
        t *= U()
    return sol
def sol(t, a, nsim): return sum(X(t, a) for _ in
   range(nsim))/nsim
print(f"x(1,1) = {sol(1,1,10**3)}")
# x(1,1) = [2.7198 2.7163]
```

Tail recursion is not always desirable as it discards intermediate values of the recursion calls and can increase computational cost. To retain some of these intermediate values, it is possible to use tail recursion partially. In example 2.6.8 it would be more efficient to avoid matrix multiplication on line 7. An alternative to tail recursion is implementing the recursion with a stack.

Python Code 2.6.9 (stack recursion on (43))

We implement (43) but this time with a stack. We want to avoid matrix multiplication and only use matrix-vector multiplications. To do this on (43) we need to know $X(Ut)$ when it doesn't get Russian rouletted away. If we sample Ut we can recurse on our reasoning until Russian roulette termination so we need the path of all the samples.

```python
from random import random as U
from collections import deque
import numpy as np
def sample_path(t):
    res = deque([t])
    while U() < t:
        t *= U()
        res.append(t)
    return res
def X(t, a) -> np.array:
    q, A = np.array([1.0, 0.0]), np.array([[a, 0.0], [1.0, a]])
    X, path = np.zeros(2), sample_path(t)
    while path:
        t = path[-1]
        X = q + (A @ X if t < 1 else t * A @ X)
        path.pop()
    return X
def sol(t, a, nsim): return sum(X(t, a) for _ in
   range(nsim))/nsim
print(f"x(1,1) = {sol(1,1,10**3)}")
# x(1,1) = [2.721 2.725]
```

3 Ordinary Differential Equations

3.1 Green's Functions

In this subsection, we discuss how to turn ODEs into integral equations and then solving them. Our main tool for this is Green's functions.

Roughly speaking Green's functions are a type of kernel function used to solve linear problems with linear conditions (it is also possible to absorb less important non-linear terms in the source term). In this context the Green's functions are like homogenous and particular solutions to a specific problem that are combined with integration to solve more general problems.

Related Work 3.1.1 (Green's function)
Our notion of Green's function is similar to that in [HGM01].

Notation 3.1.2 (H)
We denote the Heaviside step function with:

$$H(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{else} \end{cases}. \tag{44}$$

Notation 3.1.3 (δ)
We denote the Dirac delta function with $\delta(x)$.

To clarify Green's functions let's look at the following example.

Example 3.1.4 ($y_t = y$ average condition)
We will solve the equation:

$$y_t = y, \tag{45}$$

but this time with a different condition:

$$\int_0^1 y(s)ds = e - 1. \tag{46}$$

The solution to this equation remains the same: $y(t) = e^t$. We define the corresponding source Green's function $G(t, x)$ for y_t and this type of condition as follows:

$$G_t = \delta(x - t), \quad \int_0^1 G(s, x)ds = 0. \tag{47}$$

Solving this equation yields:

$$G(t, x) = H(t - x) + x - 1. \tag{48}$$

We define the corresponding boundary Green's function $G(t, x)$ for y_t and this type of condition as follows:

$$P_t = 0, \quad \int_0^1 P(s)ds = e - 1. \tag{49}$$

Solving this equation yields:

$$P(t) = e - 1. \tag{50}$$

The Green's functions are constructed such that we can form the following integral equation for (45):

$$y(t) = P(t) + \int_0^1 G(t,s)y(s)ds. \tag{51}$$

Converting (51) into a RRVE using RMC, we obtain:

$$Y(t) = e - 1 + 2B\left(\frac{1}{2}\right) Y(S)(H(t-S) + S - 1), \tag{52}$$

where $S \sim U$. We plot realizations of (52) in Figure 4.

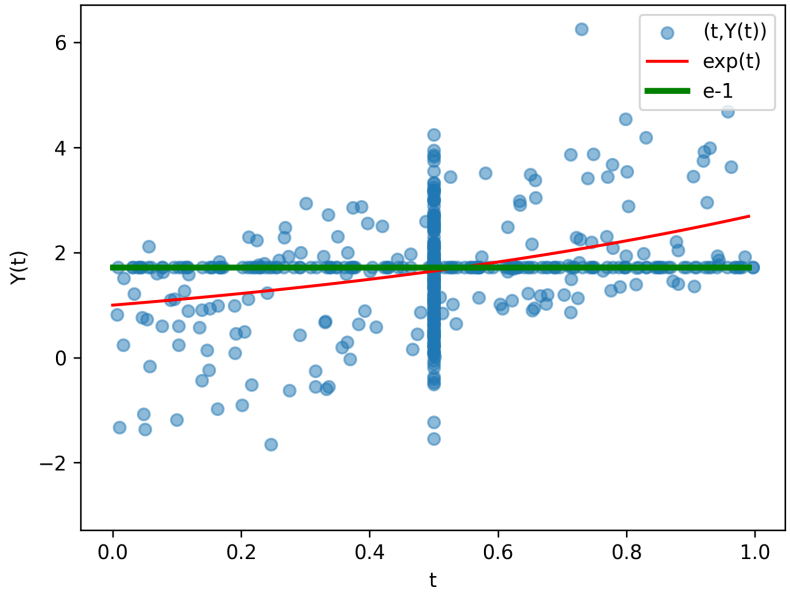

Figure 4: Recursive calls of (52) when calling $Y(0.5)$ 300 times. Points accumulate on the Green's line due to the Russian roulette, and at $t = 0.5$ because it is the starting value of the simulation.

(51) is a Fredholm integral equations of the second kind.

Definition 3.1.5 (Fredholm equation of the second kind)
A Fredholm equation of the second kind for φ is of the following form:

$$\varphi(t) = f(t) + \lambda \int_a^b K(t,s)\varphi(s)ds. \tag{53}$$

Given the kernel $K(t,s)$ and $f(t)$.

If both K and f satisfy certain regularity conditions, then for sufficiently small λ, it is relatively straightforward to establish the existence and uniqueness of solutions using a fixed-point argument.

Example 3.1.6 (Dirichlet $y_{tt} = y$)
We turn the following ODE into Fredholm integral equation of the second kind for testing:

$$y_{tt} = y, \quad y(b_0), y(b_1). \tag{54}$$

The Green's functions corresponding to y_{tt} and Dirichlet conditions are:

$$P(t,x) = \begin{cases} \frac{b_1-t}{b_1-b_0} & \text{if } x = b_0 \\ \frac{t-b_0}{b_1-b_0} & \text{if } x = b_1 \end{cases}, \tag{55}$$

$$G(t,s) = \begin{cases} -\frac{(b_1-t)(s-b_0)}{b_1-b_0} & \text{if } s < t \\ -\frac{(b_1-s)(t-b_0)}{b_1-b_0} & \text{if } t < s \end{cases}. \tag{56}$$

Straight from these Green's functions, you get the following integral equation and RRVE:

$$y(t) = P(t,b_0)y(b_0) + P(t,b_1)y(b_1) + \int_{b_0}^{b_1} G(t,s)y(s)ds, \tag{57}$$

$$Y(t) = P(t,b_0)y(b_0) + P(t,b_1)y(b_1) + lB\left(\frac{1}{l}\right)(b_1-b_0)G(t,S)y(S), \tag{58}$$

where $l \in \mathbb{R}$ the Russian roulette rate is and $S \sim \text{Uniform}(b_1, b_0)$. In Figure 5 we tested convergence of (58).

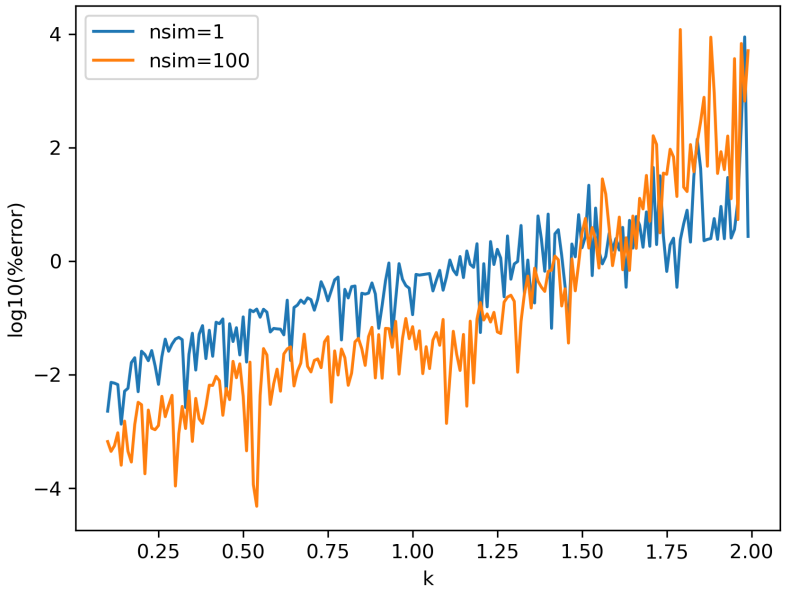

Figure 5: The logarithmic percentage error of $Y(0)$ for (58), with $l = 1.2$ and initial conditions $y(-k) = e^{-k}$ and $y(k) = e^{k}$, displays an exponential increase until approximately $k = 1.5$, beyond which additional simulations fail to reduce the error, indicating that the variance doesn't exist.

Coupled splitting is one of the ideas that we tested on example 3.1.6. Coupled splitting deals with additive branching of splitting by coupling (reusing) samples.

Example 3.1.7 (coupled splitting on example 3.1.6)
In addition to normal splitting (see definition 2.3.8), we can also split the domain in (57) as follows:

$$y(t) = P(t,b_0)y(b_0) + P(t,b_1)y(b_1) + \frac{1}{2}\int_{b_0}^{b_1} G(t,s)y(s)ds + \frac{1}{2}\int_{b_0}^{b_1} G(t,s)y(s)ds, \tag{59}$$

$$y(t) = P(t,b_0)y(b_0) + P(t,b_1)y(b_1) + \int_{b_0}^{\frac{b_1+b_0}{2}} G(t,s)y(s)ds + \int_{\frac{b_1+b_0}{2}}^{b_1} G(t,s)y(s)ds. \tag{60}$$

By coupling, we can eliminate the additive branching recursion in the RRVEs corresponding to (59) and (60). This results in the following RRVE:

$$X(t_1,t_2) = \begin{bmatrix} P(t_1,b_0) & P(t_1,b_1) \\ P(t_2,b_0) & P(t_2,b_1) \end{bmatrix} \begin{bmatrix} y(b_0) \\ y(b_1) \end{bmatrix} + W \begin{bmatrix} G(t_1,S_1) & G(t_1,S_2) \\ G(t_2,S_1) & G(t_2,S_2) \end{bmatrix} X(S_1,S_2), \tag{61}$$

where W the right weighting matrix is (see code 3.1.8), S_1 and S_2 can be chosen in various ways. (61) is unbiased in the following way: $X(t_1,t_2) \cong [y(t_1) \quad y(t_2)]^T$.

Python Code 3.1.8 (implementation of (61))
We implemented (61) with recursion but in this case, it is better to implement it forwardly.

```
from random import random as U
from math import exp
import numpy as np
def Pb0(t, b0, b1): return (b1-t)/(b1-b0)
def Pb1(t, b0, b1): return (t-b0)/(b1-b0)
def G(t, s, b0, b1): return - (b1-s)*(t-b0)/(b1-b0) if t < s
    else - (b1-t)*(s-b0)/(b1-b0)
def X(T, y0, y1, b0, b1):
    yy = np.array([y0, y1])
    bb = np.diag([(b1-b0)/len(T)]*len(T))
    PP = np.array([[Pb0(t, b0, b1), Pb1(t, b0, b1)] for t in T])
    sol = PP @ yy
    l = 1.2  # russian roulette rate
    if U()*l < 1:
        u = U()
        SS = [b0+(u+j)*(b1-b0)/len(T) for j in range(len(T))]
        GG = np.array([[G(t,S,b0,b1) for S in SS] for t in T])
        sol += l*GG @ bb @ X(SS, y0, y1, b0, b1)
    return sol
```

Related Work 3.1.9 (coupled splitting)
Coupled splitting is partly inspired by how [SM09] reduces variance by using bigger submatrices. Reusing samples for WoS is discussed in [Mil+23] and [BP23].

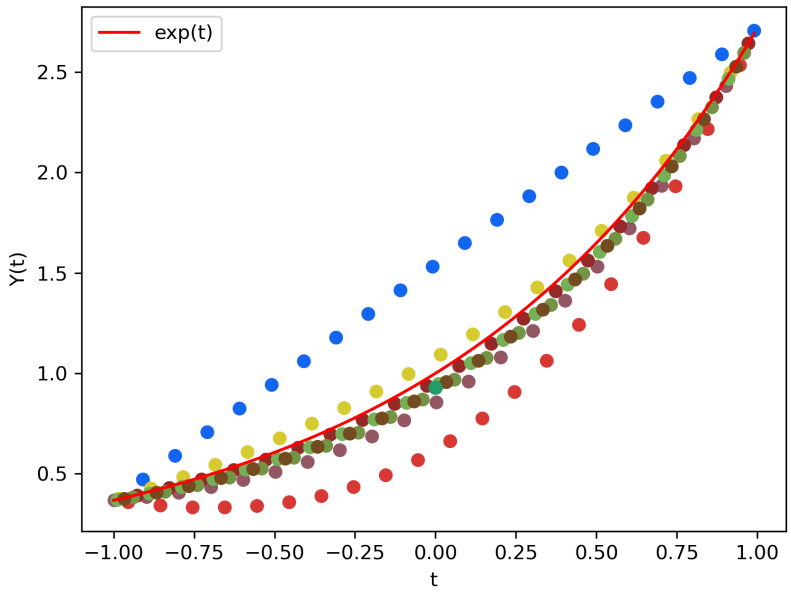

Figure 6: Recursive calls of (61) when calling $X(0)$ once. We chose X to have 20 points and colored each call. The S_j are coupled such that they are equally spaced. The initial conditions for this call are $y(-1) = e^{-1}$ and $y(1) = e^1$, with Russian roulette rate $l = 1.2$.

Figure 6 resembles fixed-point iterations, leading us to hypothesize that the convergence speed is very similar to fix-points methods until the accuracy of the stochastic approximation of the operator is reached (the approximate operator bottleneck). The approximation of the operator can be improved by increasing the coupled splitting amount when approaching the bottleneck. Alternatively when reaching the bottleneck it is possible to rely on MC convergence.

Related Work 3.1.10 (convergence coupled splitting)
See [GH21] for a discussion on the convergence of recursive stochastic algorithms.

Related Work 3.1.11 (IBC integral equations)
Optimal IBC is known for integral equations see [Hei] for the solution in 1 point and the global solution.

3.2 Initial Value Problems

Classic IVP solvers rely on shrinking the time steps for convergence. In this subsection, we introduce Recursion in Recursion MC (RRMC) for IVPs that tries to emulate this behavior.

Example 3.2.1 (RRMC $y_t = y$)
We demonstrate RRMC for IVPs with

$$y_t = y, \quad y(0) = 1. \tag{62}$$

Imagine we have a time-stepping scheme $(t_n), \forall n : t_{n-1} < t_n$ then the following integral equations hold:

$$y(t) = y(t_n) + \int_{t_n}^{t} y(s)ds, \quad t > t_n. \tag{63}$$

Turn these in the following class of RRVEs:

$$y(t) \cong Y_j(t) = y(t_j) + (t - t_j)Y_j((t - t_j)U + t_j), \quad t > t_j. \tag{64}$$

A problem with these RRVEs is that we do not know $y(t_j)$. Instead, we can replace it with an unbiased estimate y_j which we keep frozen in the inner recursion:

$$y(t) \cong Y_j(t) = y_j + (t - t_j)Y_j((t - t_j)U + t_j), \quad t > t_j \tag{65}$$

$$y(t_j) \cong y_j = \begin{cases} Y_{j-1}(t_j) & \text{if } j \neq 0 \\ y(t_0) & \text{if } j = 0 \end{cases}. \tag{66}$$

We refer to (65) as the inner recursion and (66) as the outer recursion of the recursion in recursion.

Python Code 3.2.2 (implementation of example 3.2.1)

```
from random import random as U
def Y_in(t, tn, yn, h):
    S = tn + U()*(t-tn)  # \sim Uniform(T,t)
    return yn + h*Y_in(S, tn, yn, h) if U() < (t-tn)/h else yn
def Y_out(tn, h): # h is out step size
    TT = tn-h if tn-h > 0 else 0
    return Y_in(tn, TT, Y_out(TT, h), h) if tn > 0 else 1
```

We measured the convergence speed of example 3.2.1 to be $O\left(h^{1.5}\right)$ in RMSE with h the step size. We used a scaled version of the time process of example 2.3.6 for the inner recursion such that the average amount of total inner recursion calls is en with n the total amount of outer recursion calls. Our intuition for this behavior is based on how classical solvers work and the MC trapezoidal rule.

Conjecture 3.2.3 (local RMSE RMC)
Consider a general linear IVP

$$y_t(t) = A(t)y(t) + g(t), y(t_0), \tag{67}$$

with A a matrix and g a vector function each once differentiable with the corresponding RRVE

$$y(t) \cong Y(t) = y(t_0) + hB\left(\frac{t - t_0}{h}\right)A(S)Y(S) + g(S), \tag{68}$$

with $S \sim \text{Uniform}(t_0, t)$ there holds

$$E[||y(t_1) - Y(t_1)||^2] = O(h^2), \tag{69}$$

with $t_1 = t_0 + h$.

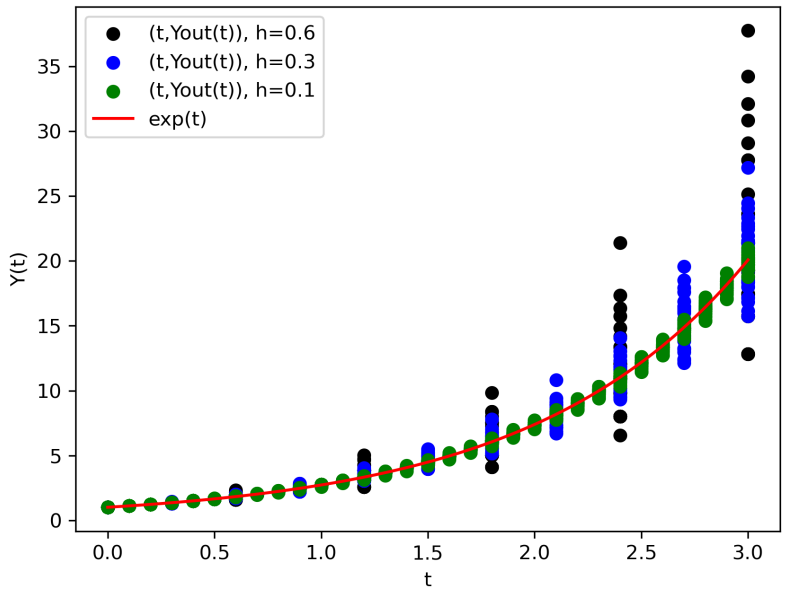

Figure 7: Recursive calls of (66) when calling $Y_{out}(3, h)$ 30 times for different h.

We can improve RRMC with control variates.

Example 3.2.4 (CV RRMC $y_t = y$)
Let us control variate example 3.2.1.

$$y(t) = y(t_n) + \int_{t_n}^{t} y(s)ds, \quad t > t_n. \tag{70}$$

We build a control variate with a lower-order approximation of the integrand:

$$\begin{aligned} y(s) &= y(t_n) + (s - t_n)y_t(t_n) + O((s - t_n)^2) && (71)\\ &\approx y(t_n) + (s - t_n)f(y(t_n), t_n) && (72)\\ &\approx y(t_n) + (s - t_n)\left(\frac{y(t_n) - y(t_{n-1})}{t_n - t_{n-1}}\right) && (73)\\ &\approx y(t_n)(1 + s - t_n). && (74) \end{aligned}$$

Using the last one as a control variate for the integral:

$$\begin{aligned} y(t) &= y(t_n) + \int_{t_n}^{t} y(s)ds && (75)\\ &= y(t_n) + \int_{t_n}^{t} y(s) - y(t_n)(1 + s - t_n) + y(t_n)(1 + s - t_n)ds && (76)\\ &= y(t_n)\left(1 + (1 - t_n)(t - t_n) + \frac{t^2 - t_n^2}{2}\right) + \int_{t_n}^{t} y(s) - y(t_n)(1 + s - t_n)ds. && (77) \end{aligned}$$

We translate this to an RRVE and plot the error for different step sizes on Figure 8.

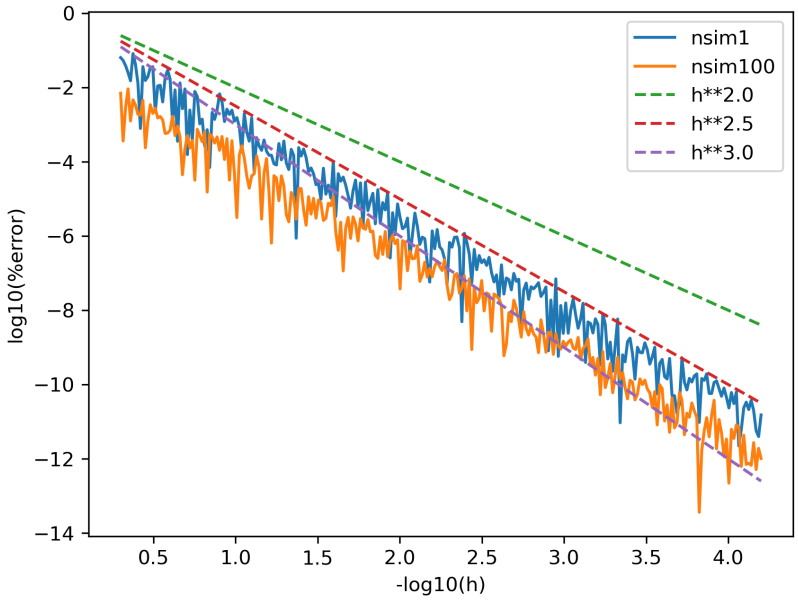

Figure 8: Log-log plot of the error for example 3.2.4 at $Y(10)$.

Related Work 3.2.5 (CV RRMC)

[Dau11] similarly uses control variates to achieve a higher order of convergence.

Similar to explicit solvers, RRMC performs poorly on stiff problems. We attempted to make RRMC more like implicit solvers but this proved difficult. Instead, we think that an approach more like exponential integrators is more viable. We experimented with Diagonal RRMC in this direction with little success.

Definition 3.2.6 (Diagonal RRMC)

Consider a general linear ODE IVP problem:

$$x' = Ax + g, \quad x(0) = x_0. \tag{78}$$

Sometimes repeatedly multiplying by A is unstable. Diagonal RRMC adds a positive diagonal matrix D to A and hopes that it stabilizes.

$$x' + Dx = (A + D)x + g. \tag{79}$$

The following integral equation can be derived by using integrating factor:

$$x(t) = e^{D(t_n - t)}x(t_n) + \int_{t_n}^{t} e^{D(s-t)}(A(s) + D)x(s)ds + \int_{t_n}^{t} e^{D(s-t)}g(s)ds. \tag{80}$$

Remember that the exponential of a diagonal matrix is the exponential of its elements. The recursive integral has the following trivial control variate:

$$\int_{t_n}^{t} e^{D(s-t)}(A(t_n) + D)x(t_n)ds = D^{-1}(I - e^{D(t_n - t)})(A(t_n) + D)x(t_n). \tag{81}$$

Note that D may be chosen differently for every outer recursion.

Example 3.2.7 (DRRMC)
Consider:

$$x' = Ax, x(0) = \begin{bmatrix} 1 \\ 0 \end{bmatrix}. \tag{82}$$

With

$$A = \begin{bmatrix} 0 & 1 \\ -1000 & -1001 \end{bmatrix}. \tag{83}$$

This has the following solution:

$$x(t) = \frac{1}{999} \begin{bmatrix} -e^{-1000t} + 1000e^{-t} \\ 1000e^{-1000t} - 1000e^{-t} \end{bmatrix}. \tag{84}$$

We choose D fixed over all outer recursions:

$$D = \begin{bmatrix} 1 & 0 \\ 0 & 1000 \end{bmatrix}. \tag{85}$$

We make the convergence plot for this example with (80) with control variate (81) implemented with recursion in recursion on Figure 9.

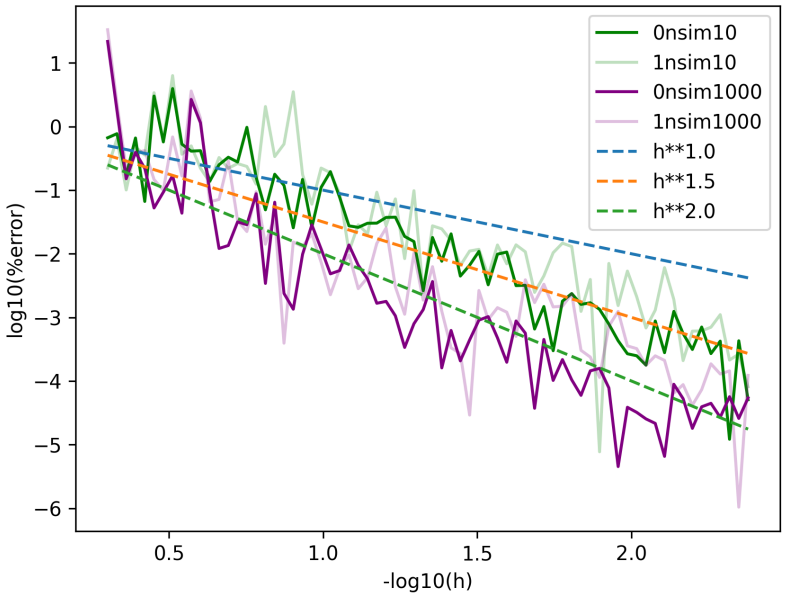

Figure 9: Log-log plot of the error of example 3.2.7. We plotted the second component of the error transparent.

Related Work 3.2.8 (DRRMC)
DRRMC is inspired by $\bar{\sigma}$ parameter in [Saw+22] but instead of importance sampling, we use control variates to deal with nonlinearity introduced by the exponential because it needs to work over an entire vector at the same time.

4 Limitations and Future Work

Our goal for this thesis was to learn about and explore different Monte Carlo techniques. We ended up limiting the scope to unbiased linear ODE solvers. In this direction, there is still much obvious work to be done.

We think that understanding and optimizing unbiased and deterministic linear ODE solvers is the key to developing better randomized ODE/PDE solvers. Randomized ODE/PDE solvers are useful for cases with little structure where the advantage of IBC is significant or in the case of a linear trade-off between cost and variance is close to optimal.

Besides that, it is sometimes convenient to have low-bias access to solutions of ODEs. For example, when integrating over a high dimensional parametric ODE problem we would need unbiased solutions to do MC integration. The following example is a toy problem where unbiased estimates are convenient.

Example 4.0.1
Consider the following parametric IVP:

$$y_t = ay, \quad y(0) = 1, \tag{86}$$

with a a parameter. The solution to this problem is given by $y(t, a) = e^{ta}$. Imagine we have a belief about a quantized in the following way $a \sim U$. If we want to estimate $E[y(t, U)]$ or in a more general case $E[f(y(t, U))]$ with f analytic directly we need samples of $y(t, U)$. If we don't have a solution for the parametric IVP we can't sample $y(t, U)$ instead, we use unbiased estimates $(Y(t, u))$ of samples $(y(t, u))$ of $y(t, U)$ in the following way using the total law of expectation:

$$E[f(y(t, U))] = E[f(y(t, u)) \mid U = u] \tag{87}$$
$$= E[f(E[Y(t, u)]) \mid U = u]. \tag{88}$$

To estimate $E[f(E[Y(t, u)]) \mid U = u]$, we use the approach outlined in example 2.5.3. For this example, we can compute the first two moments of $y(t, U)$:

$$E[y(t, U)] = \frac{e^t}{t} - \frac{1}{t}, \tag{89}$$
$$E[y^2(t, U)] = \frac{e^{2t}}{2t} - \frac{1}{2t}. \tag{90}$$

Python Code 4.0.2 (implementation of example 4.0.1)

```
from random import random as U
from math import exp
def Y(t, a):
    if t < 1: return 1+a*Y(U()*t, a) if U() < t else 1
    return 1+t*a*Y(U()*t, a)
def YU(t): return Y(t, U())
def Y2U(t):
```

```
    a = U()
    return Y(t, a)*Y(t, a)
t, nsim = 3, 10**4
sol = sum(YU(t) for _ in range(nsim))/nsim
sol2 = sum(Y2U(t) for _ in range(nsim))/nsim
s = exp(t)/t - 1/t                # analytic solution
s2 = exp(2*t)/(2*t) - 1/(2*t)     # analytic solution
print(f"E(YU({t}))   is approx {sol},%error = {(sol - s)/s}")
print(f"E(Y2U({t})) is approx {sol2},%error = {(sol2 - s2)/s2}")
# E(YU(3))   is approx 6.5683, %error = 0.0324
# E(Y2U(3)) is approx 64.5843, %error = -0.0370
```

While writing this thesis, we came up with a better way to sample the time process. The time process we used is based on example 2.3.6. There are a few ways to generalize that example for example using a linear function instead of an exponential. An issue is the lack of control over the stationary distribution of the time process see Figure 2.

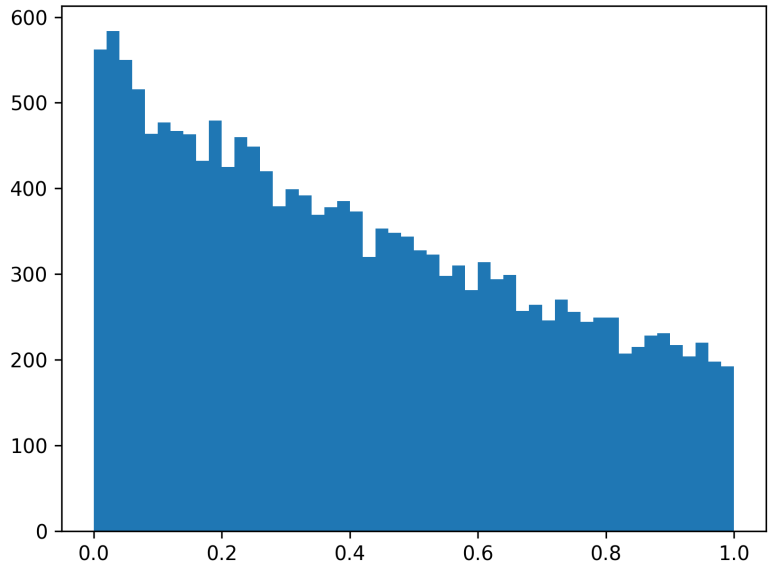

Figure 10: This histogram depicts how the points on Figure 2 are distributed over time excluding $t = 1$.

What we have in mind is using a Poisson point process that samples forwardly with the corresponding exponential distribution and Russian roulettes when out of range. This way the stationary distribution is controlled by the intensity of the Poisson process and the amount of recursion calls (for constant intensity) are Poisson distributed.

Besides the time process, the following could be developed: the way control variates are constructed, the type of control variates, adaptive schemes, freezing less important terms in the recursion in recursion or Russian rouletting them into reasonable

approximations, error estimates based on variance in the inner recursion, etc.

To deal with stiff problems in an unbiased way we would bet on exponential integrators type of methods. The biggest obstacle to implementing them like diagonal RRMC is getting unbiased estimates of $e^{A(t-s)}y(s)$ type of expressions. In the case of a big matrix A we think it would be important use unbiased sparsifaction see [SM09]. Initially, we came up with example 2.5.3 but some time later we found the paper from NVIDIA [Ket+21] that optimizes that example. Closely related to this is directly estimating the Magnus expansion where expressions like $e^{\int_0^{\Delta t} A(s)ds}y(0)$ are needed. In this case [Ket+21] doesn't address using the smoothness of $A(s)$ which is needed for optimal IBC.

One of the elements that is lacking in our findings is rigor. We think that RMC for IVPs is an informal way to think of an unbiased estimate of the Von Neumann series to the corresponding Volterra integral equation. [ET19] has theorems (Theorem 1 and 2) that these estimates have finite variance and even an expression for it. Before being aware of [ET19] we also derived a similar expression (probably with mistakes) by using the law of total variance similar to (16) in [Rat+22].
We think that proving optimal IBC for control variates + RRMC is doable but tedious. [Dau11] has a proof for optimal IBC for their algorithm in the biased non-linear case. The proof we have in mind is using a lower-bound on IBC from integration and proving it gets achieved.

Optimal IBC isn't everything. Optimal IBC doesn't mean optimized. An algorithm that uses 1000 times more function calls has the same IBC. Besides that, the computational goal may not fit well in the framework of IBC. We really like [Bec+22] which uses deep learning to go outside the IBC framework, the focus is more on multiple fast inferences where a big precomputation is allowed. IBC also doesn't take into account how parallel computations are. Given infinite parallel resources it would make sense to stop decreasing variance by reducing the step size and splitting the final estimator. All the communication that is needed then is averaging the final estimator. We conjecture that in such an estimator the wall time at risk would increase logarithmically with splitting size.

Another topic that could be investigated is only estimating 1 component of the solution. It is not obvious if this is always possible but due to the Feynman-Kac formula, it seems possible for a space discretization of the heat equation. We made the following informal partial derivation for the Feynman-Kac formula in the case heat equation to understand why and what makes point estimators possible.
Discretize the heat equation ($u_t = u_{xx}$) with a regular rectangular mesh that includes (x, t) with equally spaced intervals over space and time $(\Delta x, \Delta t)$ with the corresponding difference equation:

$$\frac{u(x,t) - u(x,t-\Delta t)}{\Delta t} = \frac{u(x+\Delta x,t) - 2u(x,t) + u(x-\Delta x,t)}{\Delta x^2}. \tag{91}$$

Isolate $u(x, t)$:

$$u(x,t) = \frac{\Delta t}{2\Delta t + \Delta x^2}\left(u(x+\Delta x,t) + u(x-\Delta x,t)\right) + \frac{\Delta x^2}{2\Delta t + \Delta x^2}\left(u(x,t-\Delta t)\right). \tag{92}$$

Now comes the essential step in the derivation. Because $u(x+\Delta x,t) \approx u(x-\Delta x,t) \approx u(x,t-\Delta t) \approx$ right-hand side of (92) we may Russian roulette to remove branching recursion and generate a recursion path instead of a tree.

$$Z(x,t) = \begin{cases} Z(x+\Delta x,t) & \text{with chance } \frac{\Delta t}{2\Delta t+\Delta x^2} \\ Z(x-\Delta x,t) & \text{with chance } \frac{\Delta t}{2\Delta t+\Delta x^2} \\ Z(x,t-\Delta t) & \text{with chance } \frac{\Delta x^2}{2\Delta t+\Delta x^2} \end{cases} \tag{93}$$

Taking the limit makes the discrete solution converge to the real solution. For (93) the limit makes the recursion path go to Brownian motion. We won't treat how to stop the recursion with boundary conditions.

Abstract

Deze scriptie onderzoekt recursieve Monte Carlo voor het oplossen van lineaire gewone differentiaalvergelijkingen met het oog op partiële differentiaalvergelijkingen. De voorgestelde algoritmes maken gebruik van de geschikte combinatie van Monte Carlo technieken. Deze Monte Carlo technieken worden geïntroduceerd met voorbeelden en code.

References

[Bec+22] Sebastian Becker et al. *Learning the random variables in Monte Carlo simulations with stochastic gradient descent: Machine learning for parametric PDEs and financial derivative pricing*. en. arXiv:2202.02717 [cs, math]. Feb. 2022. URL: http://arxiv.org/abs/2202.02717 (visited on 01/24/2023).

[BP23] Ghada Bakbouk and Pieter Peers. "Mean Value Caching for Walk on Spheres". en. In: (2023). Artwork Size: 10 pages Publisher: The Eurographics Association, 10 pages. DOI: 10.2312/SR.20231120. URL: https://diglib.eg.org/handle/10.2312/sr20231120 (visited on 08/01/2023).

[Dau11] Thomas Daun. "On the randomized solution of initial value problems". en. In: *Journal of Complexity* 27.3-4 (June 2011), pp. 300–311. ISSN: 0885064X. DOI: 10.1016/j.jco.2010.07.002. URL: https://linkinghub.elsevier.com/retrieve/pii/S0885064X1000066X (visited on 03/06/2023).

[ES21] S. M. Ermakov and M. G. Smilovitskiy. "The Monte Carlo Method for Solving Large Systems of Linear Ordinary Differential Equations". en. In: *Vestnik St. Petersburg University, Mathematics* 54.1 (Jan. 2021), pp. 28–38. ISSN: 1063-4541, 1934-7855. DOI: 10.1134/S1063454121010064. URL: https://link.springer.com/10.1134/S1063454121010064 (visited on 01/20/2023).

[ET19] S. M. Ermakov and T. M. Tovstik. "Monte Carlo Method for Solving ODE Systems". en. In: *Vestnik St. Petersburg University, Mathematics* 52.3 (July 2019), pp. 272–280. ISSN: 1063-4541, 1934-7855. DOI: 10.1134/S1063454119030087. URL: https://link.springer.com/10.1134/S1063454119030087 (visited on 01/20/2023).

[GH21] Abhishek Gupta and William B. Haskell. *Convergence of Recursive Stochastic Algorithms using Wasserstein Divergence*. en. arXiv:2003.11403 [cs, eess, math, stat]. Jan. 2021. URL: http://arxiv.org/abs/2003.11403 (visited on 01/11/2023).

[Hei] Stefan Heinrich. "Monte Carlo Complexity of Global Solution of Integral Equations". en. In: ().

[HGM01] Chi-Ok Hwang, James A. Given, and Michael Mascagni. "The Simulation–Tabulation Method for Classical Diffusion Monte Carlo". en. In: *Journal of Computational Physics* 174.2 (Dec. 2001), pp. 925–946. ISSN: 00219991. DOI: 10.1006/jcph.2001.6947. URL: https://linkinghub.elsevier.com/retrieve/pii/S0021999101969475 (visited on 12/18/2022).

[HN01] S. Heinrich and E. Novak. *Optimal Summation and Integration by Deterministic, Randomized, and Quantum Algorithms*. en. arXiv:quant-ph/0105114. May 2001. URL: http://arxiv.org/abs/quant-ph/0105114 (visited on 03/19/2023).

[Hua+17] Yipeng Huang et al. "Hybrid analog-digital solution of nonlinear partial differential equations". en. In: *Proceedings of the 50th Annual IEEE/ACM International Symposium on Microarchitecture*. Cambridge Massachusetts: ACM, Oct. 2017, pp. 665–678. ISBN: 978-1-4503-4952-9. DOI: 10.1145/3123939.3124550. URL: https://dl.acm.org/doi/10.1145/3123939.3124550 (visited on 03/01/2023).

[Ket+21] Markus Kettunen et al. “An unbiased ray-marching transmittance estimator”. en. In: *ACM Transactions on Graphics* 40.4 (Aug. 2021). arXiv:2102.10294 [cs], pp. 1–20. ISSN: 0730-0301, 1557-7368. DOI: 10.1145/3450626.3459937. URL: http://arxiv.org/abs/2102.10294 (visited on 06/18/2023).

[MH] Peter Mathé and Stefan Heinrich. “THE MONTE CARLO COMPLEXITY OF FREDHOLM INTEGRAL EQUATIONS”. en. In: ().

[Mil+23] Bailey Miller et al. *Boundary Value Caching for Walk on Spheres*. en. arXiv:2302.11825 [cs]. Feb. 2023. URL: http://arxiv.org/abs/2302.11825 (visited on 05/02/2023).

[Rat+22] Alexander Rath et al. “EARS: efficiency-aware russian roulette and splitting”. en. In: *ACM Transactions on Graphics* 41.4 (July 2022), pp. 1–14. ISSN: 0730-0301, 1557-7368. DOI: 10.1145/3528223.3530168. URL: https://dl.acm.org/doi/10.1145/3528223.3530168 (visited on 02/10/2023).

[Saw+22] Rohan Sawhney et al. “Grid-free Monte Carlo for PDEs with spatially varying coefficients”. en. In: *ACM Transactions on Graphics* 41.4 (July 2022), pp. 1–17. ISSN: 0730-0301, 1557-7368. DOI: 10.1145/3528223.3530134. URL: https://dl.acm.org/doi/10.1145/3528223.3530134 (visited on 09/17/2022).

[SM09] K. Sabelfeld and N. Mozartova. “Sparsified Randomization Algorithms for large systems of linear equations and a new version of the Random Walk on Boundary method”. en. In: *Monte Carlo Methods and Applications* 15.3 (Jan. 2009). ISSN: 0929-9629, 1569-3961. DOI: 10.1515/MCMA.2009.015. URL: https://www.degruyter.com/document/doi/10.1515/MCMA.2009.015/html (visited on 09/17/2022).

[Vea97] Eric Veach. “Robust Monte Carlo Methods for Light Transport Simulation. Ph.D. Dissertation. Stanford University.” en. In: *Robust Monte Carlo Methods for Light Transport Simulation*. (1997).

[VSJ21] Delio Vicini, Sébastien Speierer, and Wenzel Jakob. “Path replay backpropagation: differentiating light paths using constant memory and linear time”. en. In: *ACM Transactions on Graphics* 40.4 (Aug. 2021), pp. 1–14. ISSN: 0730-0301, 1557-7368. DOI: 10.1145/3450626.3459804. URL: https://dl.acm.org/doi/10.1145/3450626.3459804 (visited on 02/17/2023).