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]. They study an unbiased method for a Cauchy problem for large systems of linear ODEs. 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,
- 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 for linear IVPs see Example 3.2.3 by using recursion in recursion and variance reduction techniques.

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 method that involves 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 expected value of $f(S)$, with $S \sim \text{Uniform}(\Omega)$. Combined with the Best Linear Unbiased Estimators (BLUEs), MC Integration in that case, can be summarized in the following formula:

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

```
uniform_MC_int(f, n) = sum(f(rand()) for _ in 1:n) / n
```

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 also 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 a GPU implementation. These assumptions are useful for establishing a baseline and when they are close to being optimal, they become highly practical. In this case, 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 2.4.5.
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.

Notation 2.2.1 (U, U_j)
We will frequently use the uniform distribution, so we will abbreviate it

$$U_j \text{ i.i.d Uniform}(0, 1). \tag{6}$$

The subscripts are used to clarify independence between uniforms. However, when there is no risk of confusion, we simply use U.

By estimating the recursive integral in (5) using MC, we derive the following estimator:

$$Y(t) = 1 + ty(Ut). \tag{7}$$

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:

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

```
Y(t, eps) = t > eps ? 1 + t * Y(rand() * t, eps) : 1
```

(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 (every Y needs to sample another Y). 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.

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

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 2.3.4, 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 $Y(t)$ for $t < 1$ becomes:

$$y(t) \cong Y(t) = 1 + B(t)Y(Ut) \tag{13}$$

```
Y(t) = rand() < t ? 1 + Y(rand() * t) : 1 # correct for t<1
```

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 to sample $Y(t)$ equals e^t.

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

Definition 2.3.7 (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 an 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 2.3.8 (splitting on (13))
We can "split" the recursive term of (13) into two parts as follows:

$$y(t) \cong Y(t) = 1 + \frac{B(t)}{2}(Y_1(U_1 t) + Y_2(U_1 t)) \tag{15}$$

```
function Y(t) # correct for t<1
    u = rand()
    rand() < t ? 1 + (Y(u * t) + Y(u * t)) / 2 : 1
end
```

$$y(t) \cong Y(t) = 1 + \frac{B(t)}{2}(Y_1(U_1 t) + Y_2(U_2 t)) \tag{16}$$

```
Y(t) = rand() < t ? 1 + (Y(rand() * t) + Y(rand() * t)) / 2 : 1
```

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

Definition 2.3.9 (control variates)
Define control variating $f(U)$ with $\tilde{f}$ an approximation of f as:

$$f(U) \cong f(U_1) - \tilde{f}(U_1) + E[\tilde{f}(U)] \tag{17}$$
$$= (f - \tilde{f})(U_1) + E[\tilde{f}(U)]. \tag{18}$$

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

Example 2.3.10 (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:

$$y(t) \cong Y(t) = 1 + t(Y(U_1 t) - \tilde{y}(U_1 t) + E[\tilde{y}(Ut)]) \tag{19}$$
$$= 1 + t\,(E[1 + Ut]) + t(Y(U_1 t) - 1 - U_1 t) \tag{20}$$
$$= 1 + t + \frac{t^2}{2} + t(Y(U_1 t) - 1 - U_1 t). \tag{21}$$

```
function Y(t) # correct for t<1
    u = rand()
    1 + t + t^2 / 2 + (rand() < t ? Y(u * t) - 1 - u * t : 0)
end
```

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.

Related Work 2.3.11 (MC modification)
For further reference on Russian roulette, splitting and control variates see [Vea97].

2.4 Monte Carlo Trapezoidal Rule

In this subsection, we introduce an 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 $\tilde{f}$ represents the linear approximation of f corresponding to the trapezoidal rule as follows:

$$\int_x^{x+\Delta x} f(s)ds \tag{22}$$

$$= \int_x^{x+\Delta x} \tilde{f}(s)ds + \int_x^{x+\Delta x} f(s) - \tilde{f}(s)ds \tag{23}$$

$$= \Delta x \frac{f(x)+f(x+\Delta x)}{2} + E\left[f(S) - \tilde{f}(S)\right] \tag{24}$$

$$\cong \Delta x \frac{f(x)+f(x+\Delta x)}{2} + \Delta x l B\left(\frac{1}{l}\right)\left(f(S) - f(x) - \frac{S-x}{\Delta x}\left(f(x+\Delta x) - f(x)\right)\right), \tag{25}$$

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

$$\text{RMSE} = O\left(\Delta x^3\right). \tag{26}$$

Proof. Start from (25). The MSE is the variance so we can ignore addition by constants.

$$\text{MSE} = \text{Var}\left(\Delta x l B\left(\frac{1}{l}\right)\left(f(S) - f(x) - \frac{S-x}{\Delta x}\left(f(x+\Delta x) - f(x)\right)\right)\right) \tag{27}$$

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

$$\text{MSE} = \text{Var}\left(\Delta x l B\left(\frac{1}{l}\right)\left(f(\Delta x U + x) - f(x) - U\left(f(x+\Delta x) - f(x)\right)\right)\right) \tag{28}$$

$$= \text{Var}\left(\Delta x l B\left(\frac{1}{l}\right)\left(U\Delta x f'(x) + \frac{U^2\Delta x^2}{2} f''(Z_1) - U\left(\Delta x f'(x) + \Delta x^2 f''(z_2)\right)\right)\right) \tag{29}$$

$$= \text{Var}\left(\Delta x l B\left(\frac{1}{l}\right)\left(\frac{U^2\Delta x^2}{2} f''(Z_1) - \frac{U\Delta x^2}{2} f''(z_2)\right)\right) \tag{30}$$

$$= \Delta x^6 \text{Var}\left(l B\left(\frac{1}{l}\right)\left(\frac{U^2}{2} f''(Z_1) - \frac{U}{2} f''(z_2)\right)\right), \tag{31}$$

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 of Lemma 2.4.2)

A more generalizable proof for other types of control variates can be constructed by applying the 'separation of the main part' technique, as shown in Lemma 4 of [HM93].

Definition 2.4.4 (composite (MC) trapezoidal rule)

Define the MC trapezoidal rule for approximating the integral of function f over the interval $[0, 1]$ with a uniform grid with n intervals as follows:

$$\int_0^1 f(s)ds \approx \Delta x \sum_{j=0}^{n-1} \frac{f(x_j) + f(x_j + \Delta x)}{2}. \tag{32}$$

```
trapezium(f, n) = sum((f(j / n) + f((j + 1) / n)) / 2 for j in 0:n-1) / n
```

Define the corresponding composite MC trapezoidal with a Russian roulette rate l as follows:

$$\int_0^1 f(s)ds \cong \Delta x \sum_{j=0}^{n-1} \frac{f(x_j) + f(x_j + \Delta x)}{2} + lB\left(\frac{1}{l}\right)\left(f(S_j) - f(x_j) - \frac{S_j - x_j}{\Delta x}(f(x_j + \Delta x) - f(x_j))\right), \tag{33}$$

```
function MCtrapezium(f, n, l=100)
    sol = 0
    for j in 0:n-1
        if rand() * l < 1
            x, xx = j / n, (j + 1) / n
            S = x + rand() * (xx - x) # \sim Uniform(x, xx)
            sol += l * (f(S) - f(x) - n * (S - x) * (f(xx) - f(x))) / n
        end
    end
    return trapezium(f, n) + sol
end
```

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

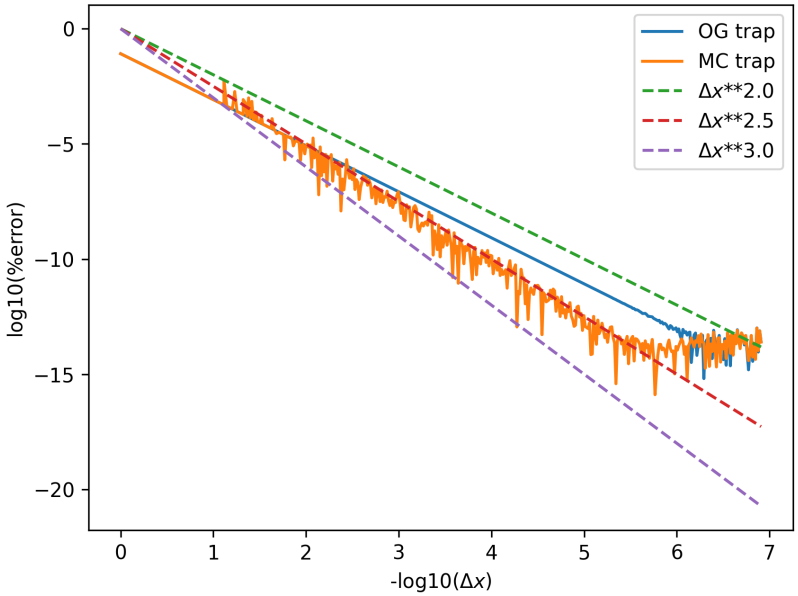

Figure 1: Log-log plot of the error of (33) for $\int_0^1 e^s ds$ *with* $l = 100$*. At floating point accuracy, the convergence ceases.*

Figure 1 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).

Theorem 2.4.5 (RMSE Composite Trapezoidal MC Rule)
The composite trapezoidal MC rule with n intervals for a twice differentiable function has

$$\text{RMSE} = O\left(\frac{1}{n^{2.5}}\right). \tag{34}$$

The proof uses Lemma 2.4.2, and is similar to the proof of the normal trapezoidal rule. The main difference is 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.

$$\sqrt{\text{Var}\left(\sum_{j=1}^{n} \Delta x^3 U_j^2\right)} = \Delta x^3 \sqrt{\sum_{j=1}^{n} \text{Var}(U_j^2)} \tag{35}$$

$$= \Delta x^3 \sqrt{n \text{Var}(U^2)} \tag{36}$$

$$= O(\Delta x^{2.5}). \tag{37}$$

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

Related Work 2.4.6 (Monte Carlo Trapezoidal Rule)
Due to an argument like 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 cases are known for some smoothness classes; see [HN01] for details.

2.5 Unbiased Non-Linearity

In this subsection, we present techniques for handling polynomial non-linearity. The main idea behind 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:

$$y_t = y^2, \quad y(1) = -1. \tag{38}$$

The solution to this equation is given by $y(t) = -\frac{1}{t}$. By integrating both sides of (38), we obtain the following integral equation:

$$y(t) = -1 + \int_1^t y(s)y(s)ds. \tag{39}$$

To estimate the recursive integral in (38), we use Y_1, Y_2 i.i.d. Y in following RRVE:

$$y(t) \cong Y(t) = -1 + (t-1)Y_1(S_1)Y_2(S_1), \tag{40}$$

```
function Y(t) # Y(u)^2 != Y(u) * Y(u) !!!
    t > 2 && error("doesn't support t > 2")
    S1 = rand() * (t - 1) + 1
    rand() < t - 1 ? -1 + Y(S1) * Y(S1) : -1
end # Y(t) = 0 or Y(t) = -1
```

where $S_1 \sim \text{Uniform}(1, t)$.

Example 2.5.2 ($e^{E[X]}$)
$e^{\int x(s)ds}$ 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:

$$e^{E[X]} = \sum_{n=0}^{\infty} \frac{E^n[X]}{n!} \tag{41}$$

$$= 1 + \frac{1}{1}E[X]\left(1 + \frac{1}{2}E[X]\left(1 + \frac{1}{3}E[X]\,(1 + ...)\right)\right). \tag{42}$$

Change the fractions of (42) to Bernoulli processes and replace all X with independent $X_j \cong X$.

$$e^{E[X]} = E\left[1 + B\left(\frac{1}{1}\right) E[X_1]\left(1 + B\left(\frac{1}{2}\right) E[X_2]\left(1 + B\left(\frac{1}{3}\right) E[X_3]\,(1 + ...)\right)\right)\right] \tag{43}$$

$$\cong 1 + B\left(\frac{1}{1}\right) X_1\left(1 + B\left(\frac{1}{2}\right) X_2\left(1 + B\left(\frac{1}{3}\right) X_3\,(1 + ...)\right)\right). \tag{44}$$

```
num_B(i) = rand() * i < 1 ? num_B(i + 1) : i - 1
res(X, n) = (n != 0) ? 1 + X() * res(X, n - 1) : 1
expE(X) = res(X, num_B(0))
```

Sampling (44) requires a finite amount of samples from X_j's with probability 1.

Related Work 2.5.3 (Example 2.5.2)
NVIDIA has a great paper on optimizing Example 2.5.2 [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:

$$y_t = ay, y(0) = 1 \Rightarrow \tag{45}$$

$$\partial_a y_t = y + a\partial_a y \tag{46}$$

Turn (45) and (46) into RRVEs. To emphasize that they are coupled and should recurse together we write them in a matrix equation:

$$\begin{bmatrix} y(t) \\ \partial_a y(t) \end{bmatrix} \cong \begin{bmatrix} Y(t) \\ \partial_a Y(t) \end{bmatrix} = X(t) = \begin{bmatrix} 1 \\ 0 \end{bmatrix} + t \begin{bmatrix} a & 0 \\ 1 & a \end{bmatrix} X(Ut). \tag{47}$$

```
(q = [1, 0]; A(a) = [a 0; 1 a])
X(t, a) = (rand() < t) ? q + A(a) * X(rand() * t, a) : q
```

Observe how this eliminates the additive branching recursion present in (46).

Technique 2.6.3 (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 2.6.4 (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 2.6.5 (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 using tail recursion thanks to the associativity of all operations ($(xy)z = x(yz)$) involved.

Julia Code 2.6.6 (tail recursion on (47))

We implemented (47) using tail recursion this time. We collect addition operations in a vector called *sol*, and multiplications in a matrix named W. W may also be referred to as accumulated weight or throughput.

```
function X(t, a)
    (q = [1, 0]; A = [a 0; 1 a])
    (sol = q; W = [1.0 0.0; 0.0 1.0])
    while rand() < t
        W = W * A
        sol += W * q
        t *= rand()
    end
    sol
end
```

Tail recursion avoids intermediate values of the recursive calls which may be useful and can increase computational costs because of the ordering of operations. In the example shown in Code 2.6.6, it would be more efficient to avoid matrix-matrix multiplication. This efficiency concern becomes worse with larger matrix multiplications. An alternative to tail recursion is implementing the recursion with a stack.

Related Work 2.6.7 (non-branching tail recursion)

Example 2.6.2 is inspired by [VSJ21]. [VSJ21] proposes an efficient unbiased backpropagation algorithm for rendering exploiting properties of non-branching tail recursion. To access intermediate values in tail recursion they use path replay.

Julia Code 2.6.8 (stack recursion on (47))

We implement (47) but this time with a stack. We want to avoid matrix multiplication and only use matrix-vector multiplications. To do this on (47) 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.

```
function sample_path(t)
    res = [float(t)]
    while rand() < t
        t *= rand()
        push!(res, t)
    end
```

```
    res
end

function X(t, a)
    (q = [1.0, 0.0]; A = [a 0.0; 1.0 a])
    (sol = zero(q); path = sample_path(t))
    while !isempty(path)
        t = path[end]
        sol = q + A * sol
        pop!(path)
    end
    sol
end
```

In stack recursion we need to store the sample path because we generate the sample path in reverse order then we need it. Reverse number generators exists but because of Russian roulette there is no starting point without going through a forward (it is still confusing what is forward and backwards ...) pass of the path. Poisson time processes don't have this problem because forward paths and backward paths have the same distribution.

Definition 2.6.9 (main Poisson)

$$y' = Ay + f \Leftrightarrow \tag{48}$$
$$y' + \sigma y = (A + \sigma I)y + f \Leftrightarrow \tag{49}$$
$$e^{-\sigma t}(e^{\sigma t}y)' = (A + \sigma I)y + f \Leftrightarrow \tag{50}$$
$$y(t) = e^{-\sigma t}y(0) + \int_0^t e^{(s-t)\sigma}\left((A + \sigma I)y(s) + f(s)\right) ds. \tag{51}$$

With $\sigma > 0$. Do following substitution: $e^{(s-t)\sigma} = \tau$ equivalent to importance sampling the exponential term:

$$y(t) = \int_0^{e^{-\sigma t}} y(0)d\tau + \int_{e^{-\sigma t}}^1 \left(\frac{A}{\sigma} + I\right) y(s) + \frac{f(s)}{\sigma} d\tau. \tag{52}$$

```
function Y(t, sig, A::Function, f::Function, y0)
    (s = -log(rand()) / sig; sol = y0)
    while s < t
        sol += (A(s) * sol .+ f(s)) ./ sig
        s -= log(rand()) / sig
    end
    sol
end
```

(52) can be turned into a RRVE by sampling $\tau \sim U$, no Russian roulette is needed for termination because in the recursion the first integral get sampled with probability 1. No recursion is used for the implementation as we can sample the Poisson proces in the reverse order.

Related Work 2.6.10 (main Poisson)

A very similar estimator is used to solve ODEs derived from the telegrapher equations in [AR16].

3 Ordinary Differential Equations

3.1 Green's Functions

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

A Green's function is a type of kernel function used for solving linear problems with linear conditions. In this context, Green's functions are analogous to homogeneous and particular solutions to a specific problem, which are combined through integration to solve more general problems.

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

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

$$H(x) = \begin{cases} 0 & \text{if } x < 0 \\ 1 & \text{else} \end{cases}. \tag{53}$$

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

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:

$$y_t = y, \tag{54}$$

but this time with a different condition:

$$\int_0^1 y(s)ds = e - 1. \tag{55}$$

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:

$$G_t = \delta(x - t), \quad \int_0^1 G(s, x)ds = 0. \tag{56}$$

Solving this equation yields:

$$G(t, x) = H(t - x) + x - 1. \tag{57}$$

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

$$P_t = 0, \quad \int_0^1 P(s)ds = e - 1. \tag{58}$$

Solving this equation yields:

$$P(t) = e - 1. \tag{59}$$

The Green's functions are constructed such that we can form the following integral equation for (54):

$$y(t) = P(t) + \int_0^1 G(t,s)y(s)ds. \tag{60}$$

Converting (60) into an RRVE using RMC, we obtain:

$$Y(t) = e - 1 + 2B\left(\frac{1}{2}\right) Y(S)(H(t-S) + S - 1), \tag{61}$$

where $S \sim U$. We plot realizations of (61) in Figure 2.

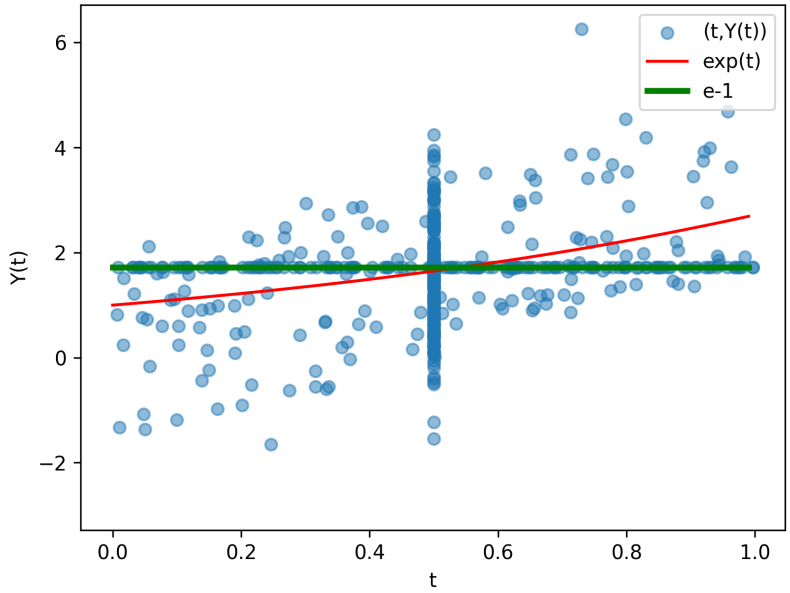

Figure 2: Recursive calls $(t, Y(t))$ of (61) when calling $Y(0.5)$ 300 times. Points accumulate on the Green's line due to the Russian roulette, and at $t = 0.5$ because it is the starting value of the simulation.

(60) is a Fredholm integral equation 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:

$$\varphi(t) = f(t) + \lambda \int_a^b K(t,s)\varphi(s)ds. \tag{62}$$

Here, $K(t,s)$ represents a kernel, and $f(t)$ is a given function.

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 transform the following ODE into Fredholm integral equation of the second kind for testing:

$$y_{tt} = y, \quad y(b_0), y(b_1). \tag{63}$$

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

$$P(t|x) = \begin{cases} \frac{b_1 - t}{b_1 - b_0} & \text{if } x = b_0 \\ \frac{t - b_0}{b_1 - b_0} & \text{if } x = b_1 \end{cases}, \tag{64}$$

$$G(t,s) = \begin{cases} -\frac{(b_1 - t)(s - b_0)}{b_1 - b_0} & \text{if } s < t \\ -\frac{(b_1 - s)(t - b_0)}{b_1 - b_0} & \text{if } t < s \end{cases}. \tag{65}$$

Directly from these Green's functions, we obtain the following integral equation and RRVE:

$$y(t) = P(t|b_0)y(b_0) + P(t|b_1)y(b_1) + \int_{b_0}^{b_1} G(t,s)y(s)ds, \tag{66}$$

$$Y(t) = P(t|b_0)y(b_0) + P(t|b_1)y(b_1) + lB\left(\frac{1}{l}\right)(b_1 - b_0)G(t,S)y(S), \tag{67}$$

with the Russian roulette rate $l \in \mathbb{R}$ and $S \sim \text{Uniform}(b_1, b_0)$. In Figure 3 we test convergence of (67).

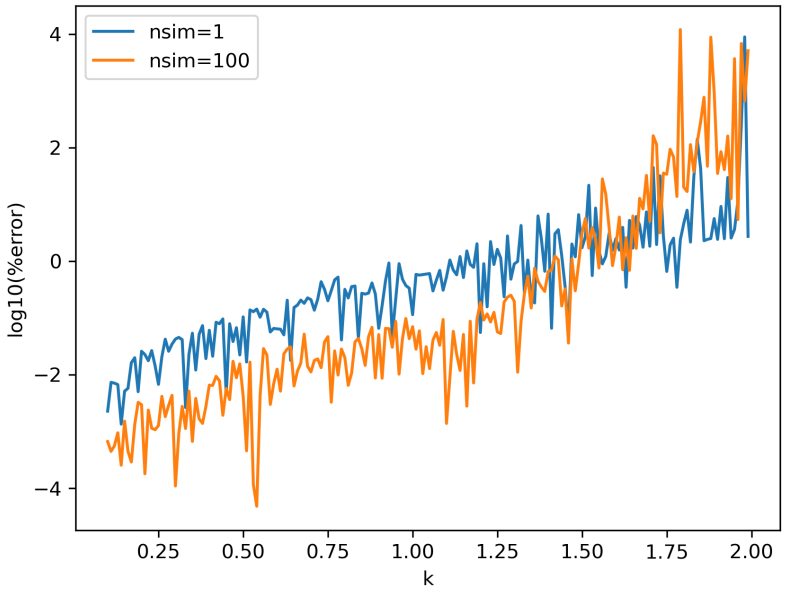

Figure 3: The logarithmic percentage error of $Y(0)$ for (67), with $l = 1.2$ and initial conditions $y(-k) = e^{-k}$ and $y(k) = e^{k}$, displays an exponential increase until approximately $k = 1.5$, beyond which additional simulations fail to reduce the error, indicating that the variance doesn't exist.

Coupled splitting is one of the ideas we tested on Example 3.1.6. Coupled splitting removes the 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 2.3.7), we can also split the domain in (66) as follows:

$$y(t) = P(t|b_0)y(b_0) + P(t|b_1)y(b_1) + \frac{1}{2}\int_{b_0}^{b_1} G(t,s)y(s)ds + \frac{1}{2}\int_{b_0}^{b_1} G(t,s)y(s)ds, \tag{68}$$

$$y(t) = P(t|b_0)y(b_0) + P(t|b_1)y(b_1) + \int_{b_0}^{\frac{b_1+b_0}{2}} G(t,s)y(s)ds + \int_{\frac{b_1+b_0}{2}}^{b_1} G(t,s)y(s)ds. \tag{69}$$

By coupling, we can eliminate the additive branching recursion in the RRVEs corresponding to (68) and (69). This results in the following RRVE:

$$X(t_1,t_2) = \begin{bmatrix} P(t_1|b_0) & P(t_1|b_1) \\ P(t_2|b_0) & P(t_2|b_1) \end{bmatrix} \begin{bmatrix} y(b_0) \\ y(b_1) \end{bmatrix} + W \begin{bmatrix} G(t_1,S_1) & G(t_1,S_2) \\ G(t_2,S_1) & G(t_2,S_2) \end{bmatrix} X(S_1,S_2), \tag{70}$$

```
Pb0(t, b0, b1) = (b1 - t) / (b1 - b0)
Pb1(t, b0, b1) = (t - b0) / (b1 - b0)
G(t, s, b0, b1) = (t < s) ?
                  -(b1 - s) * (t - b0) / (b1 - b0) :
                  -(b1 - t) * (s - b0) / (b1 - b0)
function X(T::Array, y0, y1, b0, b1, l)
    PP = hcat([Pb0(t, b0, b1) for t in T], [Pb1(t, b0, b1) for t in T])
    sol = PP * [y0, y1]
    if rand() * l < 1
        (u = rand(); n = length(T))
        SS = [b0 + (u + j) * (b1 - b0) / n for j in 0:n-1]
        GG = reshape([G(t, S, b0, b1) for t in T, S in SS], n, n)
        sol += l * GG * X(SS, y0, y1, b0, b1) .* (b1 - b0) ./ n
    end
    sol
end
```

with W some weighting matrix, S_1 and S_2 can be chosen in various ways. (70) is unbiased in the following way: $X(t_1,t_2) \cong [y(t_1) \quad y(t_2)]^T$.

Related Work 3.1.8 (coupled splitting)
Coupled splitting is partly inspired by the approach of [SM09], which reduces variance by using larger submatrices in unbiased sparsification of matrices. The idea of reusing samples for WoS is discussed in both [Mil+23] and [BP23].

Figure 4 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 splitting to convergence.

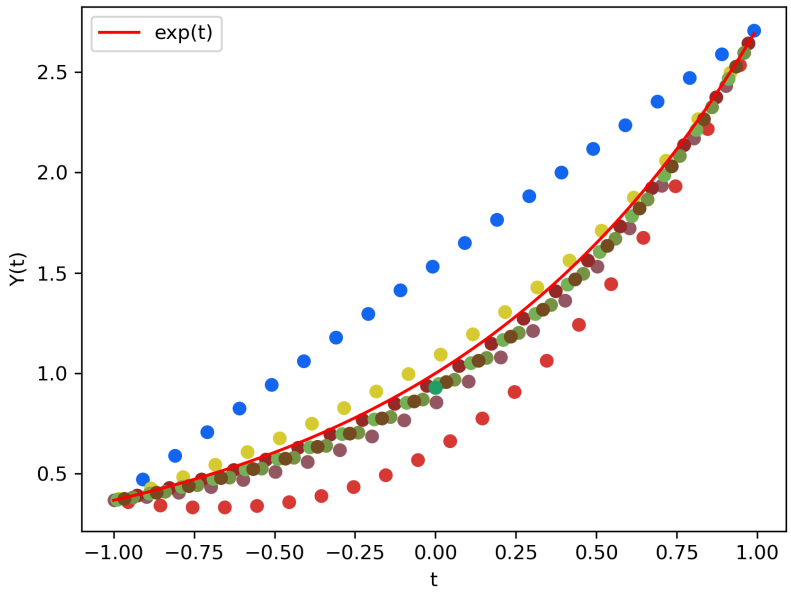

Figure 4: Recursive calls of (70) when calling $X(0)$ once. We chose X to have 20 points and colored each call. The S_j are coupled such that they are equally spaced. The initial conditions for this call are $y(-1) = e^{-1}$ and $y(1) = e^1$, with Russian roulette rate $l = 1.2$.

Related Work 3.1.9 (convergence coupled splitting)
See [GH21] for a discussion on the convergence of recursive stochastic algorithms.

Related Work 3.1.10 (IBC integral equations)
Optimal IBC is known for integral equations see [Hei98] for the solution at 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 which tries to emulate this behavior.

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

$$y_t = y, \quad y(0) = 1. \tag{71}$$

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

$$y(t) = y(t_n) + \int_{t_n}^{t} y(s)ds, \quad t > t_n. \tag{72}$$

Turn these in the following class of RRVEs:

$$y(t) \cong Y_j(t) = y(t_j) + (t - t_j)Y_j((t - t_j)U + t_j), \quad t > t_j. \tag{73}$$

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 fixed in the inner recursion:

$$y(t) \cong Y_j(t) = y_j + (t - t_j)Y_j((t - t_j)U + t_j), \quad t > t_j \tag{74}$$

$$y(t_j) \cong y_j = \begin{cases} Y_{j-1}(t_j) & \text{if } j \neq 0 \\ y(t_0) & \text{if } j = 0 \end{cases}. \tag{75}$$

```
function Y_in(t, tn, yn, h)
    S = tn + rand() * (t - tn)  # \sim Uniform(T,t)
    return rand() < (t - tn) / h ? yn + h * Y_in(S, tn, yn, h) : yn
end
function Y_out(tn, h) # h is out step size
    TT = tn > h ? tn - h : 0
    return tn > 0 ? Y_in(tn, TT, Y_out(TT, h), h) : 1
end
```

We refer to (74) as the inner recursion and (75) as the outer recursion of the recursion in recursion. The measured RMSE for estimating $y(t)$ is of the order $O\left(h^{1.5}\right)$, where h represents the step size. For the implementation we used a scaled version of the time process from Example 2.3.6 for the inner recursion, such that the average number of total inner recursion calls is en, where n represents the total number of outer recursion calls.

Conjecture 3.2.2 (local RMSE RMC)
Consider a general linear IVP:

$$y_t(t) = A(t)y(t) + g(t), y(t_0), \tag{76}$$

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

$$y(t) \cong Y(t) = y(t_0) + hB\left(\frac{t - t_0}{h}\right)A(S)Y(S) + g(S), \tag{77}$$

where $S \sim \text{Uniform}(t_0, t)$. Then, the following relation holds:

$$E[||y(t_1) - Y(t_1)||^2] = O(h^2), \tag{78}$$

with $t_1 = t_0 + h$.

To achieve a higher order of convergence RRMC can be combined with control variates.

Example 3.2.3 (CV RRMC $y_t = y$)
Let us control variate Example 3.2.1.

$$y(t) = y(t_n) + \int_{t_n}^{t} y(s)ds, \quad t > t_n. \tag{79}$$

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

$$y(s) = y(t_n) + (s - t_n)y_t(t_n) + O((s - t_n)^2) \tag{80}$$
$$\approx y(t_n) + (s - t_n)f(y(t_n), t_n) \tag{81}$$
$$\approx y(t_n) + (s - t_n)\left(\frac{y(t_n) - y(t_{n-1})}{t_n - t_{n-1}}\right) \tag{82}$$
$$\approx y(t_n)(1 + s - t_n). \tag{83}$$

Using (83) as a control variate for the integral:

$$y(t) = y(t_n) + \int_{t_n}^{t} y(s)ds \tag{84}$$
$$= y(t_n) + \int_{t_n}^{t} y(s) - y(t_n)(1 + s - t_n) + y(t_n)(1 + s - t_n)ds \tag{85}$$
$$= y(t_n)\left(1 + (1 - t_n)(t - t_n) + \frac{t^2 - t_n^2}{2}\right) + \int_{t_n}^{t} y(s) - y(t_n)(1 + s - t_n)ds. \tag{86}$$

Figure 5 displays the error of realizations of an RRVE constructed from (86) for various step sizes.

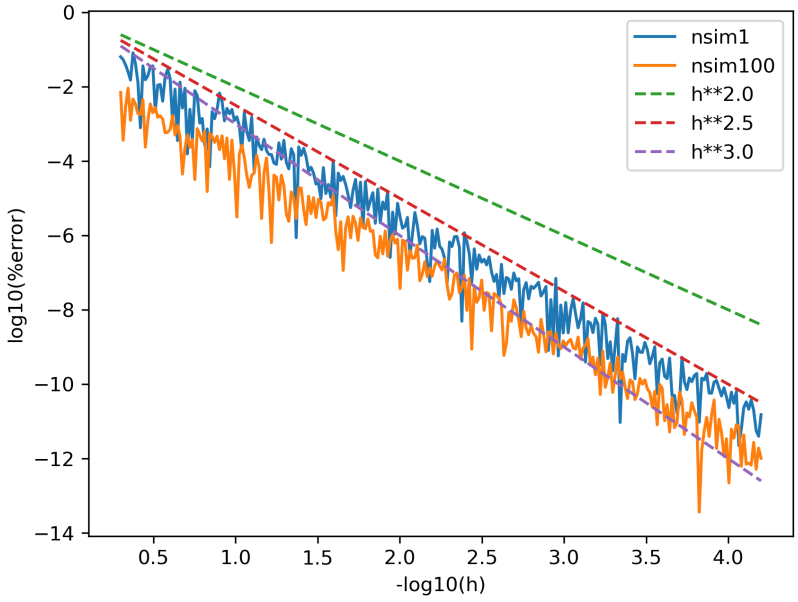

Figure 5: Log-log plot of the error for Example 3.2.3 at $Y(10)$.

Related Work 3.2.4 (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 believe that an approach more like exponential integrators is more viable.

4 Brownian Motion

Current RMC algorithms for PDEs are related to Brownian motion. In this section, we explore the relationship between the heat equation and Brownian motion, and discuss how recursive first-passage sampling fits into the picture.

Lemma 4.0.1 (self-affinity Brownian motion)
Brownian motion is a self-affine random process, which implies that any subpath can be translated and scaled in such a way that its distribution matches that of the entire path.

$$\forall c \in \mathbb{R}_0^+ : \frac{W_{ct}}{\sqrt{c}} \sim W_t. \tag{87}$$

4.1 Heat Equation

In this subsection, we introduce the relation between the heat equation and Brownian motion.

Definition 4.1.1 (1D heat equation Dirichlet)
We define the 1D heat equation for $u(x,t)$ with $(x,t) \in \Omega = [0,1]^2$ with Dirichlet boundary conditions the following way:

$$\frac{\partial u}{\partial t} = \frac{\partial^2 u}{\partial x^2} \Leftrightarrow \tag{88}$$

$$u_t(x,t) = u_{xx}(x,t), \quad \forall (x,t) \in \Omega = [0,1]^2. \tag{89}$$

Given $\{u(x,0), u(1,t), u(0,t) | \forall x,t \in [0,1]\}$.

Definition 4.1.2 (semi-discretization 1D heat equation Dirichlet)
We define the semi-discretization 1D heat equation for $\tilde{u}$ as space discretized heat equation using the central difference scheme:

$$\tilde{u}_t(j\Delta x, t) = \frac{\tilde{u}((j+1)\Delta x, t) - 2\tilde{u}(j\Delta x, t) + \tilde{u}((j-1)\Delta x, t)}{\Delta x^2} \tag{90}$$

$\forall (j,t) \in \{0,1,\ldots,N\} \times [0,1]$.
Given $\{\tilde{u}(j\Delta x, 0), \tilde{u}(1,t), \tilde{u}(0,t) | \forall j \in \{0,1,\ldots,N\}, \forall t \in [0,1]\}$.
It well-known that the solution to the semi-discretization 1D heat equation converges to the solution of the 1D heat equation with corresponding boundary conditions as $\Delta x \to 0$.
We want a point estimator for the solution to the semi-discretization 1D heat equation. We obtain this by using (52) with $\sigma = \frac{2}{\Delta x^2}$ and looking at 1 point.

$$\tilde{u}(j\Delta x, t) = \int_0^{e^{-\frac{2t}{\Delta x^2}}} \tilde{u}(j\Delta x, 0) d\tau + \int_{e^{-\frac{2t}{\Delta x^2}}}^1 \frac{\tilde{u}((j+1)\Delta x, s) + \tilde{u}((j-1)\Delta x, s)}{2} d\tau. \tag{91}$$

Julia Code 4.1.3 (implementation of estimator of (91))
(91) can be turned into a estimator $Y(j\Delta x, t, \Delta x) \cong \tilde{u}(j\Delta x, t)$ by sampling $\tau \sim U$

(and transforming everything back), treating the 2 integrals as 1, sampling 1 of the $\tilde{u}$ of the second integral to avoid branching recursion and recursing when $\tilde{u}$ isn't known. Note that we can't reverse the order of the Poisson process because of termination on the boundary but tail recursion is efficient because sampling 1 term avoids matrix multiplications completely.

```
function Ytail(x, t, dx)
    while true
        xold = x
        t += dx^2 * log(rand()) / 2
        x += rand(Bool) ? dx : -dx
        return t <= 0 ? u(xold, 0) :
               x <= 0 ? u(0, t) :
               x >= 1 ? u(1, t) : continue
    end
end
```

From Code 4.1.3 we can see that the point estimator is just the boundary condition for the first passage point from the time space domain for a Poisson time proces and a random walk in space. In this implementation as $\Delta x \to 0$ the cost of calculating the first passage time grows $\approx O(\Delta x^{-2})$ and the process converges to Brownian motion. Note that the estimator needs only 1 function call from the boundary conditions.

A similar derivation can be done for

$$u_t = u_{xx} + a(x,t)u + f(x,t). \tag{92}$$

Again using (52) with $\sigma = \frac{2}{\Delta x^2} - a_0$ and looking at 1 point we obtain:

$$u = \int_0^{e^{-t\sigma}} u_0 d\tau + \int_{e^{-t\sigma}}^1 \sigma^{-1} \left(\frac{u_+ + u_-}{\Delta x^2} + (a(x,s) - a_0)u + f \right) d\tau. \tag{93}$$

A typical choice for a_0 is $\min(a(x,s))$. As $\Delta x \to 0$ the $\frac{u_+ + u_-}{\Delta x^2}$ term is the main contribution to the second integral therefore we sample it with $(1 - a_0 \frac{\Delta x^2}{2})$ probability and the rest with $a_0 \frac{\Delta x^2}{2}$ probability. In total for 1 sample, the $f(x,s)$ and the $a(x,s)u$ term only has to be sampled some times and this does not scale with Δx.

Julia Code 4.1.4

Because of the high chance of sampling the $\frac{u_+ + u_-}{\Delta x^2}$ it efficient to sample until we don't sample it. This is geometrically distributed.

```
using Distributions
function Yvar(x, t, dx, a0)
    siginv = 1 / (2 / dx^2 - a0)
    geom = Geometric(a0 * dx^2 / 2)
    expon = Exponential(siginv)
    (sol = 0; w = 1)
    sourcejump = rand(geom)
    while true
        t -= rand(expon)
        t <= 0 && return sol + w * u(x, 0)
```

```
        if sourcejump != 0
            x += rand(Bool) ? dx : -dx
            w *= 2 * (siginv / (1 - a0 * dx^2 / 2)) / dx^2
            sourcejump -= 1
            return x <= 0 ? w * u(0, t) + sol :
                   x >= 1 ? w * u(1, t) + sol : continue
        else # this is only done O(tstart) times in an estimator
            sourcejump = rand(geom)
            sol += w * f(x, t) * siginv / (a0 * dx^2 / 2)
            w *= (a(x, t) - a0) * siginv / (a0 * dx^2 / 2)
        end
    end
end
```

As $\Delta x \to 0$ the bottleneck is sampling the proces that moves through space and time similar to first passage sampling. This proces doesn't depend directly on $a(x,t)$ and $f(x,t)$, therefore we can optimize it independent of it and even partly precompute it.

Related Work 4.1.5

The Feynman-Kac formula is related to this. See [Øks03] for a reference.

4.2 First Passage Sampling

In this subsection we discuss techniques to accelerate first passage sampling.

Definition 4.2.1 (first passage time)

Define the first passage time for a process X_t for a set of valid states V as

$$\mathrm{FPt}(X_t, V) = \inf\{t > 0 | (X_t, t) \notin V\}. \tag{94}$$

Note that the first passage time is a RV itself.

Definition 4.2.2 (first passage)

Define the first passage for a process X_t for a set of valid states V as

$$\mathrm{FP}(X_t, V) = (X_\tau, \tau), \tau = \mathrm{FPt}(X_t, V). \tag{95}$$

Lemma 4.2.3

When a process has more valid states the first passage time gets larger i.e.

$$V_1 \subset V_2 \Rightarrow \mathrm{FPt}(X_t(\omega), V_1) \leq \mathrm{FPt}(X_t(\omega), V_2). \tag{96}$$

The ω is to indicate we mean the same realization of X_t.

Theorem 4.2.4 (Green's functions and first passage distribution)

The the density of first passages of Code 4.1.3 is the corresponding Dirichlet boundary Green's function for the semi-discretized heat equation.

Proof. Follows directly from the definition. Let $P(x, t|x_0, t_0)$ denote the boundary Green's function for the semi-discretized heat equation and X_t the corresponding

space time proces.

$$P(x,t|x_0,t_0) = E[Y(x_0,t_0)] \tag{97}$$
$$= E[\tilde{u}(X_\tau,\tau)|X_{t_0} = x_0] \tag{98}$$
$$= E[\delta_{((X_\tau,\tau)=(x,t))}|X_{t_0} = x_0] \tag{99}$$
$$= P((X_\tau,\tau) = (x,t)|X_{t_0} = x_0). \tag{100}$$

□

Another way to obtain an unbiased estimator for the semi-discretized heat equation is using unbiased estimates of the boundary Green's boundary function. Because of:

$$\tilde{u}(x_0,t_0) = \int_{\partial\Omega} \tilde{u}(x,t)P(x,t|x_0,t_0)d(x,t) \tag{101}$$
$$= \int_{\partial\Omega} \tilde{u}(x,t)dP((X_\tau,\tau) = (x,t)|X_{t_0} = x_0). \tag{102}$$

Obtaining unbiased estimator for the boundary Green's function is not trivial.

Lemma 4.2.5 (law of total probability (not sure))
Let P_1 be the boundary Green's function for a contained domain Ω_1 within the semi-discretized heat problem then:

$$P(x,t|x_0,t_0) = \int_{\partial\Omega_1} P_1(x,t|x_1,t_1)P(x_1,t_1|x_0,t_0)d(x_1,t_1). \tag{103}$$

Given a family of "elementary" boundary Green's functions for the semi-discretized heat problem we can construct an unbiased estimator for the boundary Green's function by recursively sampling the law of total probability.

Related Work 4.2.6
For a reference on Green's functions in this context see [Qi+22].

Similar to RMC for boundary Green's function we can recursively sample first passages.

Technique 4.2.7 (recursive first passage sampling)
Recursive first passage sampling involves sampling an initial, simpler first passage, the base that includes fewer valid states. Using this sampled first passage as a starting point, we then perform the same sampling process until the sampled first passage is almost invalid.

Example 4.2.8 (Euler first passage sampling)
In this example, we approximately sample the first passage of Brownian motion for a parabolic barrier by simulating Brownian motion with the Euler scheme. We plotted this in Figure 6. The accuracy of the sampled first passage are $O(\Delta t)$ and the cost to sample is $O(\Delta t^{-1})$.

Example 4.2.9 (recursive first passage sampling)
In this example, we sample the first passage of Brownian motion from a parabolic

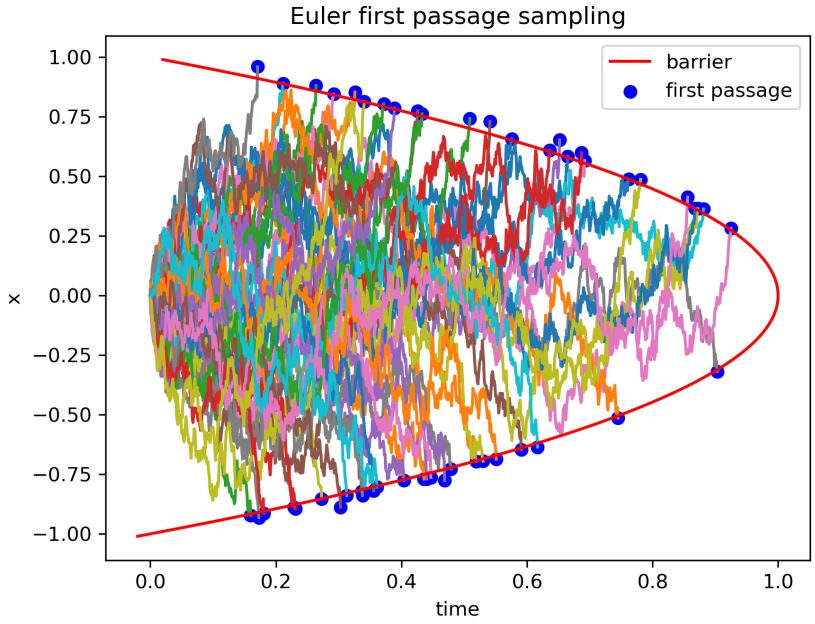

Figure 6: 50 *realizations of Euler first passage sampling with step size* 0.001.

barrier with recursive first passage sampling. For the simpler first passage sampler, we scale and translate samples from a triangular barrier so its valid states are contained in the parabola generated by the Euler scheme. The precomputed samples are created by (4.2.10) and used to produce first passages in (4.2.11). Assuming that the error of the base sampler is insignificant. The accuracy of the sampled first passage are $O(\varepsilon)$ and the cost to sample is $O(\ln(\varepsilon^{-1}))$.

Julia Code 4.2.10 (Euler first passage sampling)

```
in_triangle(time, pos) = (1 - time > abs(pos))
function sample_euler_triangle(dt=0.001)
    (pos = 0; time = 0)
    while in_triangle(time, pos)
        pos += sqrt(dt) * randn()
        time += dt
    end
    return (time, pos)
end
```

Julia Code 4.2.11 (recursive first passage sampling)

The maximum scaling of the triangular barrier that fits in the parabola is derived through (4.0.1) and using the fact that a parabola domain is convex. To dampen barrier overstepping of (4.2.10) we use a smaller scaling than the maximum.

```
function triangle_scale_in_para(time, pos)
    xx = sqrt(1 - time) - abs(pos)
    tt = abs(1 - abs(pos)^2 - time)
    return sqrt(tt) < xx ? sqrt(tt) : xx
end
```

```
function sample_recursive_para(accuracy=0.01, scale_mul=0.9)
    (time = 0.0; pos = 0.0)
    scale = triangle_scale_in_para(time, pos)
    while scale > accuracy
        scale *= scale_mul
        dtime, dpos = triangle_sample[rand(1:length(triangle_sample))]
        dtime, dpos = scale^2 * dtime, scale * dpos
        (time += dtime; pos += dpos)
        scale = triangle_scale_in_para(time, pos)
    end
    return (time, pos)
end
```

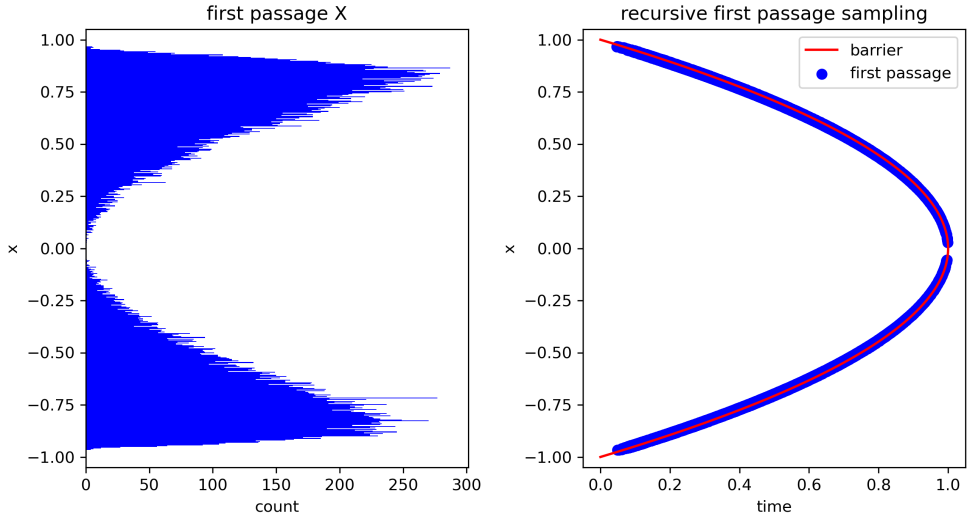

Figure 7: 50000 *of realizations of recursive first passage sampling produced by (4.2.11). The precomputed sample of* 5000 *first passages of a triangular barrier uses the Euler scheme with step size* 0.001.

Related Work 4.2.12 (recursive first passage sampling)
The original walk on spheres is a recursive first passage algorithm. Recursive first passage sampling for Brownian motion is discussed in [HT16] and by transformation also first passage problems for the Ornstein-Uhlenbeck process. An alternative to resampling from an Euler scheme is to use tabulated inverse cumulative probability functions, as demonstrated in [HGM01].

Technique 4.2.13 (Brownian motion path stitching)
Instead of sampling the first passages you can also sample whole paths to the first passage. Similar to before we need to be able to generate paths for a simple first passage problem and "stitch" these paths together. An advantage over normally generating paths is that a path can be represented by its subpaths and their scalings requiring less memory than a fully stored path. This can be useful in case you need to look back to a part of the path. The only downsides are that the time steps are inhomogeneous and it requires storing paths for the simpler first passage problem.

Example 4.2.14 (path stitching parabola)

This is the same as example (4.2.9) but now we have to keep track of the whole resampled Euler scheme generated paths.

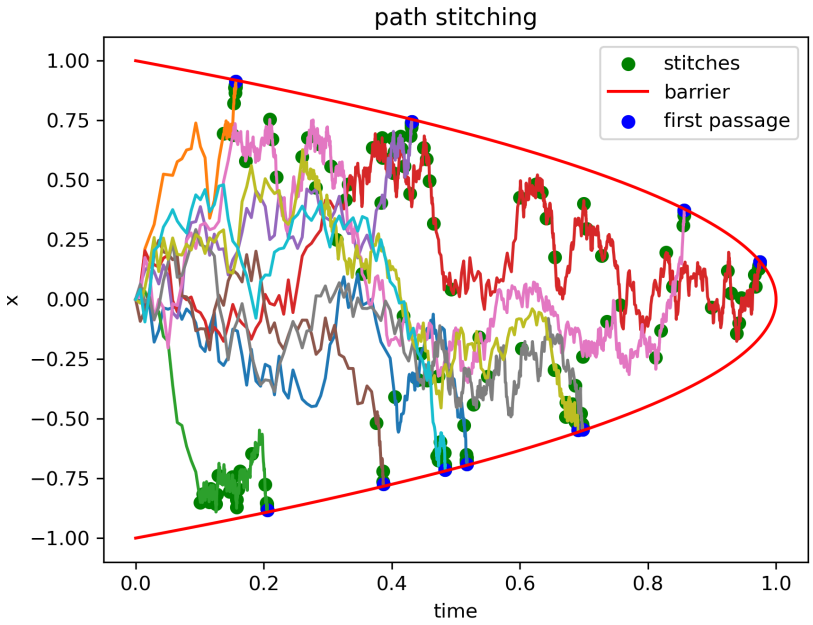

Figure 8: 10 *paths build with path stitching build out of precomputed Euler scheme generated paths with step size* 0.01.

Related Work 4.2.15 (path stitching)
Path stitching appears frequently in rendering and also in [Das+15] and [JL12].

4.3 Limitations and Future Work

It would be interesting to extend proof technique of (**??**) to the full Feynman-Kac formula. Just adding a source term f in the Feynman-Kac formula adds an integral of the source term over the recursion path. In equation (**??**) it would add $\frac{\Delta t \Delta x^2}{2\Delta t + \Delta x^2} f(x,t)$ term. This term can either be kept or Russian rouletted to prevent infinite source evaluations in the limit. This roughly corresponds to estimating the integral by MC integration also requires knowing a finite amount of intermediate points of the recursion path sampled randomly. These can possibly be obtained from the intermediate first passages of recursive first passage sampling or with a smart application of path stitching.

Recursive first passage sampling for Brownian motion can be extended to geometric Brownian motion by transforming the space. However, this transformation breaks the symmetries required for the average used in the example (**??**). An unsatisfying way to fix this is to approximate the base sampler at different points in space and at different scales. Just accelerating a classic Euler scheme at critical places with a base sampler can reduce precomputation requirements.

5 Limitations and Future Work

We believe 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 where the linear trade-off between cost and variance is close to optimal.

Besides that, some problems require access to low-bias solutions of ODEs. For example, when integrating a high-dimensional parametric ODE problem. The following example is a toy problem to showcase this.

Example 5.0.1
Consider the following parametric IVP:

$$y_t = ay, \quad y(0) = 1, \tag{104}$$

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:

$$\begin{aligned} E[f(y(t, U))] &= E[f(y(t, u)) \mid U = u] && (105)\\ &= E[f(E[Y(t, u)]) \mid U = u]. && (106) \end{aligned}$$

To estimate $E[f(E[Y(t, u)]) \mid U = u]$, we use the approach outlined in Example 2.5.2. The first two moments of $y(t, U)$ are:

$$E[y(t, U)] = \frac{e^t}{t} - \frac{1}{t}, \tag{107}$$

$$E[y^2(t, U)] = \frac{e^{2t}}{2t} - \frac{1}{2t}. \tag{108}$$

Python Code 5.0.2 (implementation of Example 5.0.1)

```
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
```

The time process we used based on Example 2.3.6 has little control over the distribution of recursion calls $(t, Y(t))$ in time see Figure **??**.

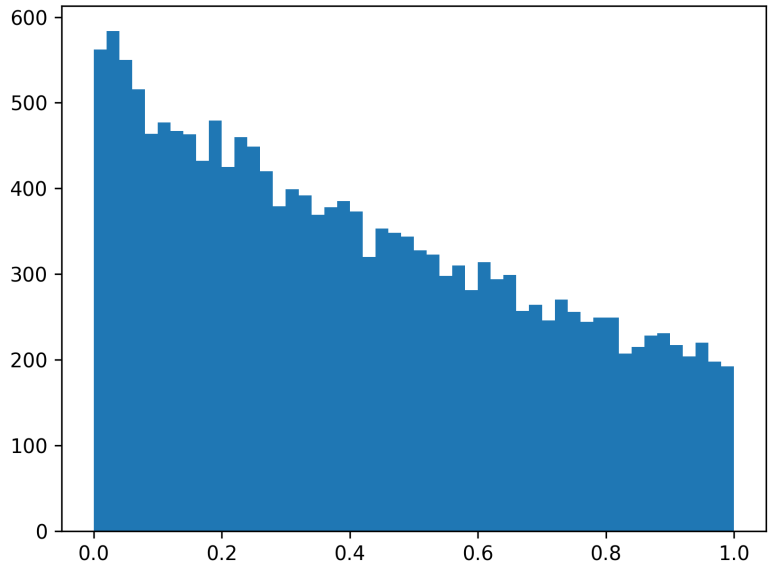

Figure 9: This histogram depicts how the points on Figure **??** *are distributed over time excluding* $t = 1$.

Instead of that, a Poisson point process can be used. It samples backward with the corresponding exponential distribution and employs Russian roulette when out of range. The distribution of the recursion calls over time can be controlled by its intensity. In the case of constant intensity, the number of recursive calls follows a Poisson distribution.

Besides the time process, other areas of development could include the cheaper construction of control variates, different types of control variates, adaptive schemes, freezing less important terms in the inner recursion or Russian rouletting them into reasonable approximations, error estimates based on variance in the inner recursion, etc.

To handle stiff problems we weigh towards exponential integrators type of methods. The biggest obstacle to implementing them similar to diagonal RRMC is getting unbiased estimates of $e^{A(t-s)}y(s)$ type of expressions. In the case of a big matrix A unbiased sparsifaction will probably be useful see [SM09]. Initially, we came up with Example 2.5.2, 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 utilize the smoothness of $A(s)$, which is necessary for optimal IBC.
One of the elements lacking in our findings is rigor. We believe that RMC is an informal approach to an unbiased estimate of the Von Neumann series to the corresponding integral equation. [ES21] presents Theorems 1 and 2 to show that their estimates have finite variance and provide an expression for it. Before becoming aware of [ES21], we previously derived a similar expression (with errors) by employing the law of total variance, similar to (16) in [Rat+22].
We believe that proving the optimality of IBC in Example 3.2.3 is feasible but tedious. [Dau11] presented a proof for optimal IBC for their algorithm. The proof we have in mind is using a lower bound on IBC from integration and proving it is attained.

Optimal IBC isn't everything. Being optimal in IBC doesn't necessarily mean it's optimized. An algorithm that uses 1000 times more function calls may still have the same IBC. Additionally, the computational goal might not align well with the IBC framework. We admire [Bec+22], which employs deep learning to extend beyond the IBC framework. The emphasis here is on performing multiple rapid solves (inferences) while allowing for an expensive precomputation (training). IBC also doesn't take into account the parallel nature of computations. Given infinite parallel resources, it would be reasonable to cease reducing variance by decreasing the step size and instead opt for splitting the final estimator. The only necessary communication in splitting is averaging the final estimator. We hypothesize that for RMC, the wall time at risk increases logarithmically with the size of the splitting.

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

[AR16] Juan A. Acebrón and Marco A. Ribeiro. "A Monte Carlo method for solving the one-dimensional telegraph equations with boundary conditions". en. In: *Journal of Computational Physics* 305 (Jan. 2016), pp. 29–43. ISSN: 00219991. DOI: 10.1016/j.jcp.2015.10.027. URL: https://linkinghub.elsevier.com/retrieve/pii/S0021999115006919 (visited on 09/09/2023).

[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).

[Das+15] Atish Das Sarma et al. "Fast distributed PageRank computation". en. In: *Theoretical Computer Science* 561 (Jan. 2015), pp. 113–121. ISSN: 03043975. DOI: 10.1016/j.tcs.2014.04.003. URL: https://linkinghub.elsevier.com/retrieve/pii/S0304397514002709 (visited on 01/05/2023).

[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).

[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).

[Hei98] S. Heinrich. "Monte Carlo Complexity of Global Solution of Integral Equations". In: *Journal of Complexity* 14.2 (1998), pp. 151–175. ISSN: 0885-064X. DOI: https://doi.org/10.1006/jcom.1998.0471. URL: https://www.sciencedirect.com/science/article/pii/S0885064X9890471X.

[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).

[HM93] Stefan Heinrich and Peter Mathé. "The Monte Carlo complexity of Fredholm integral equations". In: *mathematics of computation* 60.201 (1993). ISBN: 0025-5718, pp. 257–278.

[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).

[HT16] Samuel Herrmann and Etienne Tanré. "The first-passage time of the Brownian motion to a curved boundary: an algorithmic approach". en. In: *SIAM Journal on Scientific Computing* 38.1 (Jan. 2016). arXiv:1501.07060 [math], A196–A215. ISSN: 1064-8275, 1095-7197. DOI: 10.1137/151006172. URL: http://arxiv.org/abs/1501.07060 (visited on 12/20/2022).

[JL12] Hao Ji and Yaohang Li. "Reusing Random Walks in Monte Carlo Methods for Linear Systems". en. In: *Procedia Computer Science* 9 (2012), pp. 383–392. ISSN: 18770509. DOI: 10.1016/j.procs.2012.04.041. URL: https://linkinghub.elsevier.com/retrieve/pii/S1877050912001627 (visited on 09/17/2022).

[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).

[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).

[Øks03] Bernt Øksendal. *Stochastic Differential Equations*. en. Universitext. Berlin, Heidelberg: Springer Berlin Heidelberg, 2003. ISBN: 978-3-540-04758-2 978-3-642-14394-6. DOI: 10.1007/978-3-642-14394-6. URL: http://link.springer.com/10.1007/978-3-642-14394-6 (visited on 05/08/2023).

[Qi+22] Yang Qi et al. "A bidirectional formulation for Walk on Spheres". en. In: *Computer Graphics Forum* 41.4 (July 2022), pp. 51–62. ISSN: 0167-7055, 1467-8659. DOI: 10.1111/cgf.14586. URL: https://onlinelibrary.wiley.com/doi/10.1111/cgf.14586 (visited on 12/22/2022).

[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).