Universiteit
Antwerpen

Masterproef scriptie

Recursive Monte Carlo

Auteur: *Isidoor Pinillo Esquivel*

Promotor: *Wim Vanroose*

Academiejaar 2022-2023

Contents

Abstract

We will write this at the end. Also need a dutch abstract

1 Introduction

1.1 Introductory Example

In this subsection we introduce recursive Monte Carlo with our main example for IVPs:

$$y' = y, y(0) = 1. \tag{1}$$

Integrating both sides of (1) obtains:

$$y(t) = 1 + \int_0^t y(s)ds. \tag{2}$$

Equation (2) is a recursive integral equation or to be more specific a linear Volterra integral equation of the second type. By estimating the recursive integral of equation (2) with Monte Carlo one derives following estimator:

$$Y(t) = 1 + ty(Ut).$$

With $U \sim \text{Uniform}(0, 1)$. If y is well behaved then $E[Y(t)] = y(t)$ but we can't calculate $Y(t)$ without accesses to $y(s), s < t$. Notice that we can replace y by a unbiased estimator of it without changing $E[Y(t)] = y(t)$ by the law of total expectance ($E[X] = E[E[X|Z]]$). By replacing y by Y itself we obtain a recursive expression for Y:

$$Y(t) = 1 + tY(Ut). \tag{3}$$

Equation (3) is a recursive random variable equation (RRVE). If you would implement equation (3) with recursion it will run indefinitely. A biased way of around this is by approximating $Y(t) \approx 1$ near $t = 0$. Later we discuss Russian roulette (2.2.1) which can be used as an unbiased stopping mechanism.

Python Code 1.1.1 (implementation of (3))

```
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 realizations of RRVEs, it can be helpful to plot all recursive calls $(t, Y(t))$, as shown in Figure 1 for this implementation.

An issue with (1.1.1) is that the variance increases rapidly when t increases. Later this is issue gets resolved in the section on ODEs. Note that (1.1.1) keeps desirable properties from unbiased Monte Carlo methods such as being embarrassingly parallel and having simple error estimates.

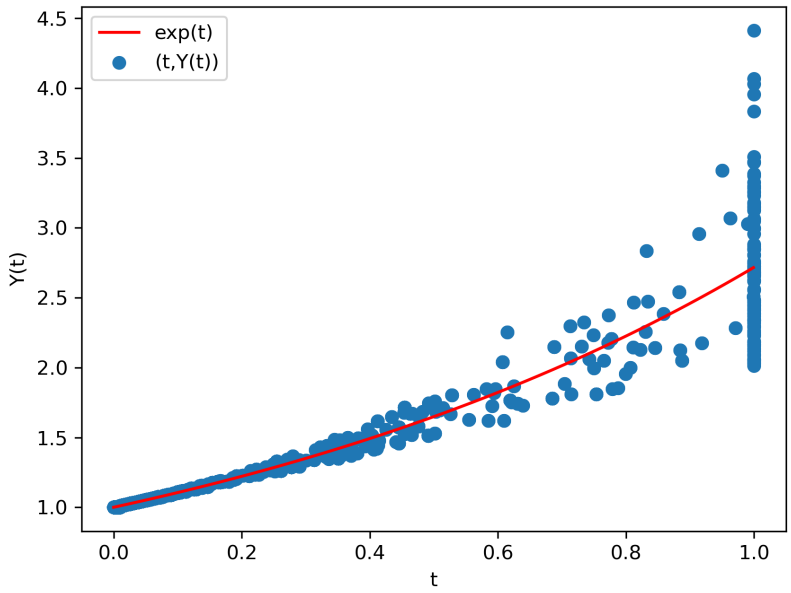

Figure 1: Recursive calls of (1.1.1)

1.2 Contributions

We write this at the end. Probably a lot of conjectures.

1.3 Related Work

work on

- alternative methods for recursive integrals
- MC work on ODEs
- MC work on PDEs
- WoS

This is just to give a general overview we probably reference specific ideas when we first introduce them.

2 Background

2.1 Notation

Notations used in this thesis include:

- $U \sim \text{Uniform}(0, 1)$.
- $B(p) \sim \text{Bernoulli}(p)$.

- RV = random variable.
- RVs will be denoted with capital letters, e.g., X, Y or Z.
- RRVE (recursive RV equation): An equation that defines a family of random variables in terms of its self.
- MC = Monte Carlo.
- RMC = Recursive MC.
- RRMC = Recursion in Recursion MC.

2.2 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.

Russian roulette is a MC technique commonly used in rendering. The main idea behind Russian roulette is to replace a random variable with a less computationally expensive approximation sometimes.

Definition 2.2.1 (Russian roulette)
Define Russian roulette on X with free parameters $Y_1, Y_2 : E[Y_1] = E[Y_2]$, $p \in [0, 1]$ and U independent of Y_1, Y_2, X the following way:

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

Example 2.2.2 (Russian roulette)
Say that we are interested in estimating $E[Z]$ with Z defined in the following way:

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

where $f : \mathbb{R} \to [0, 1]$ expensive to compute. Estimating $E[Z]$ directly would require calling f each simulation. We can modify Z to

$$\tilde{Z} = U + B\left(\frac{1}{100}\right)\frac{f(U)}{10}.$$

Now $\tilde{Z}$ just requires calling f on average once every 100 simulations with the variance only increasing slightly compared to Z.

In this example it is also possible to estimate the expectance of the 2 terms of Z separately.

Example 2.2.3 (Russian roulette on (3))
Russian roulette can fix the indefinite recursion issue of equation (3) by approximating Y near $t = 0$ with 1 sometimes. Concretely we replace the t in front of the recursive term with $B(t)$ when $t < 1$.

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

Python Code 2.2.4 (Russian roulette on (3))

```
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, $Y(t)$ is constrained to take on only integer values. This is visually evident on Figure 2.

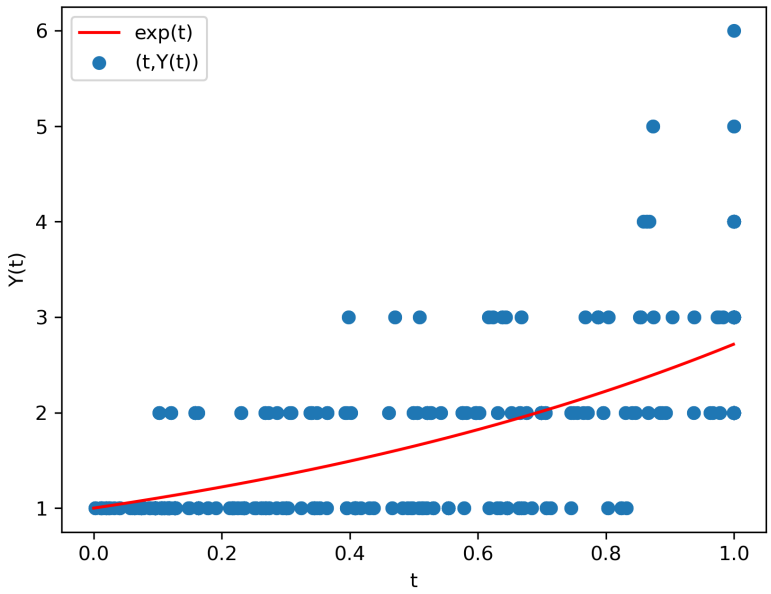

Figure 2: Recursive calls $(t, Y(t))$ of (2.2.4)

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 samples) which reduces the variance.

Definition 2.2.5 (splitting)
Splitting X means using multiple $X_j \sim X$ not independent per se to lower variance by averaging them:

$$\bar{X} = \frac{1}{N}\sum_{j=1}^{N} X_j.$$

Splitting the recursive term in a RRVE can lead to (additive) branching recursion, which requires extra care to ensure that the branches get terminated quickly to avoid an exponential increase in computational complexity. This can be achieved by employing termination strategies that have already been discussed. Later on, we will discuss the use of coupled recursion as a technique for alleviating additive branching recursion in RRVEs.

Example 2.2.6 (splitting on (3))
We can "split" the recursive term of (3) in 2:

$$Y(t) = 1 + \frac{t}{2}(Y_1(Ut) + Y_2(Ut)).$$

with $Y_1(t), Y_2(t)$ i.i.d. $Y(t)$.

Python Code 2.2.7 (splitting on (3))

```
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 2.2.8 (2-level MC)
2-level MC on X with parameters $\tilde{X}, Y : E[\tilde{X}] = E[Y]$:

$$X \to X - \tilde{X} + Y.$$

Definition 2.2.9 (control variates)
Control variate on $f(X)$ is

$$f(X) \to f(X) - \tilde{f}(X) + E[\tilde{f}(X)].$$

Control variates are a special case of 2-level MC. Usually $\tilde{f}$ is an approximation of f to reduce variance.

Example 2.2.10 (control variate on (3))
To make a control variate for (3) that reduces variance we use following approximation of $y(t) \approx 1 + t$:

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

Notice that we can cancel the constant term of the control variate but that would affect the Russian roulette negatively.

Python Code 2.2.11 (control variate on (3))

```
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 2.2.12
Our favorite work that discusses these techniques is [Vea]. More interesting works can be found on MC techniques in rendering. 2-level gets discussed in [Gil13].

2.3 Monte Carlo Trapezoidal Rule

We present in this subsection a MC trapezoidal rule with similar convergence behavior to methods discussed later. The MC trapezoidal rule will just be regular MC control variated with the normal trapezoidal rule.

Definition 2.3.1 (MC trapezoidal rule)
Define the MC trapezoidal rule for f on $[x, x+dx]$ the following way:

$$\int_x^{x+dx} f(s)ds \approx \frac{f(x)+f(x+dx)}{2} + f(S_x) - f(x) - \frac{S_x - x}{dx}\left(f(x+dx) - f(x)\right) \quad (4)$$

with $S_x = \text{Uniform}(x, x+dx)$.

Defining the composite MC trapezoidal rule as the sum of MC trapezoidal rules on equally divided intervals is possible but expensive. Every interval would add a function call compared to the normal composite MC trapezoidal rule. Instead you can aggressively Russian roulette into the normal trapezoidal rule such that the increase in functions calls is arbitrarily small.

Definition 2.3.2 (composite MC trapezoidal rule)
Define the composite MC trapezoidal rule for f on $[a, b]$ with n intervals and a Russian roulette rate l the following way:

$$\int_a^b f(s)ds \approx \quad (5)$$

$$\sum_x \frac{f(x)+f(x+dx)}{2} + lB\left(\frac{1}{l}\right)\left(f(S_x) - f(x) - \frac{S_x - x}{dx}(f(x+dx) - f(x))\right) \quad (6)$$

with $S_x = \text{Uniform}(x, x+dx)$.

Python Code 2.3.3 (implementation of (2.3.2))
We implement (2.3.2) for $\int_0^1 e^s ds$.

```
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(x)+f(x+1/n))/2
    for x in np.arange(0, 1, 1/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:8.794793540941216e-11
```

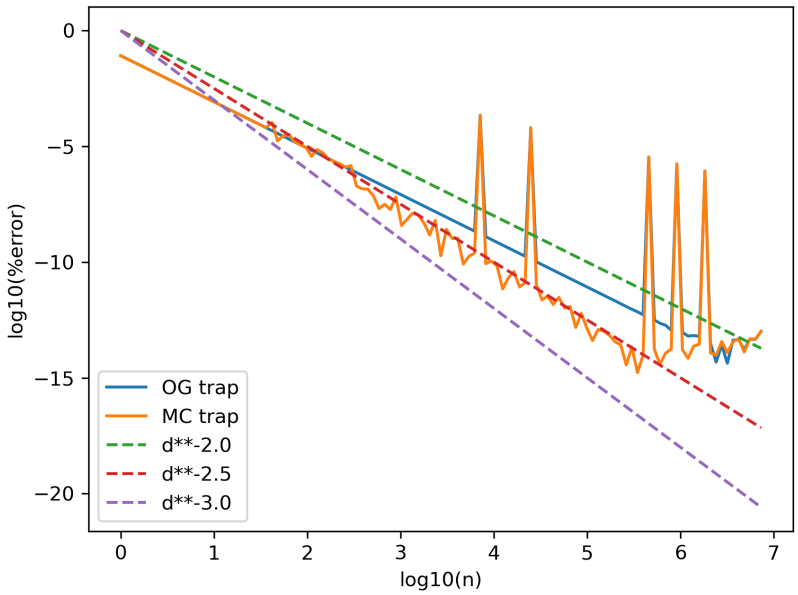

Figure 3: Log-log plot of (2.3.2) for $\int_0^1 e^s ds$ with $l = 100$.

We postulate that this MC composite rule enhances the convergence rate by 0.5 orders compared to the standard composite rule for each dimension, provided that the appropriate conditions of smoothness are met. To substantiate this conjecture, we shall outline our rationale concerning the accumulation of unbiased polynomial errors.

For the sake of simplicity, we reduce the study of the local truncation error to the following expression:

$$\int_0^h s^2 ds = O(h^3).$$

In the standard composite rule, we would drop this term, but in the MC version, we eliminate the bias. As a result, the local truncation error behaves like:

$$\int_0^h s^2 ds - h(hU)^2 = \int_0^h s^2 ds - h^3 U^2 = O(h^3).$$

The main distinction between the standard and MC rule lies in how they accumulate local truncation errors into global truncation error. In the standard case, there is a loss of one order. When measuring the error of randomized algorithms , the root mean square error is typically used, which, in the unbiased case, is equivalent to the standard deviation:

$$\begin{aligned}
\sqrt{\text{Var}\left(\sum_{j=1}^{n}\int_0^h s^2 ds - h^3 U_j^2\right)} &= \sqrt{\text{Var}\left(\sum_{j=1}^{n} h^3 U_j^2\right)} \\
&= h^3\sqrt{\text{Var}\left(\sum_{j=1}^{n} U_j^2\right)} \\
&= h^3\sqrt{\sum_{j=1}^{n}\text{Var}(U_j^2)} \\
&= h^3\sqrt{n\text{Var}(U^2)} \\
&= h^3\sqrt{n}\sqrt{\text{Var}(U^2)} \\
&= O(h^{2.5}).
\end{aligned}$$

Related Work 2.3.4
Optimal theoretical bounds on randomized algorithms can be found in: (see literature randomized trapezoidal rule) [Wu20]. The half order convergence that randomized gains over determinstic isn't the same gain we consider.

2.4 Unbiased Non-Linearity

In this subsection we introduce techniques to deal with non-linearity. At first it may looks only possible to deal with linear problems in an unbiased way but by using independent samples it possible to deal with polynomial non-linearity's (which theoretically extend to any continuos functions by the Weierstrass approximation theorem). It is not always easy to transform non-linearity into polynomials but it is not difficult to come up with biased alternative approaches based on linearization or approximate polynomial non-linearity.

Example 2.4.1 ($y' = y^2$)
Let's do following example:

$$y' = y^2, y(1) = -1. \tag{7}$$

This has solution $-\frac{1}{t}$. Integrate both sides of equation (7) to arrive at following integral equation:

$$y(t) = -1 + \int_1^t y(s)y(s)ds. \tag{8}$$

To estimate the recursive integral in equation (2) we use 2 independent $Y_1, Y_2 \sim Y$:

$$Y(t) = -1 + (t-1)Y_1(S)Y_2(S). \tag{9}$$

With $S \sim \text{Uniform}(1, t)$. This is a branching RRVE this is typical when dealing with non-linearity.

Python Code 2.4.2 ($y' = y^2$)

```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
```

Example 2.4.3 ($e^{E[X]}$)

$e^{\int x(s)ds}$ is 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:

$$e^{E[X]} = \sum_{n=0}^{\infty} \frac{E^n[X]}{n!} \tag{10}$$

$$= 1 + \frac{1}{1}E[X]\left(1 + \frac{1}{2}E[X]\left(1 + \frac{1}{3}E[X]\left(1 + ...\right)\right)\right). \tag{11}$$

Change the fractions of equation (11) to Bernoulli processes and replace all X with independent X_j with $E[X] = E[X_i]$.

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

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

What is inside the expectation is something that we can simulate with simulations of X_j.

Python Code 2.4.4 ($e^{E[X]}$)

The following python code estimates $e^{\int_0^t s^2 ds}$:

```python
from random import random as U
from math import exp
def X(t): return -t**3*U()**2
def num_B(i):  # = depth of Bernoulli's = 1
    return num_B(i+1) if U()*i < 1 else i-1
def res(n, t): return 1 + X(t)*res(n-1, t) if n != 0 else 1
def expE(t): return res(num_B(0), t)

t, nsim = 1, 10**3
sol = sum(expE(t) for _ in range(nsim))/nsim
exact = exp(-t**3/3)
print(f"sol = {sol} %error={(sol- exact)/exact}")
#sol = 0.7075010309320893 %error=-0.01260277046
```

Related Work 2.4.5

A similar approach to non-linearity can be found in [ET19]. We have more papers on how to deal with non-linearity stashed, no idea if they are worth mentioning.

2.5 Recursion

In this subsection we discuss recursion related techniques.

Technique 2.5.1 (coupled recursion)
The idea behind coupled recursion is sharing recursion calls of multiple RRVEs for simulation. This does make them dependent. It is like assuming 2 induction hypotheses at the same time and proving both inductions steps at the same time vs doing separate induction proofs. Which should be easier because you have accesses to more assumptions at the same time.

Example 2.5.2 (coupled recursion)
Lets say you are interested in calculating the sensitivity of the solution of an ODE to a parameter a:

$$y' = ay, y(0) = 1 \Rightarrow \tag{12}$$
$$\partial_a y' = y + a\partial_a y \tag{13}$$

Turn (12) and (13) into RRVEs. To emphasize that they are coupled, that they should recurse together we write them in a matrix equation:

$$\begin{bmatrix} Y(t) \\ \partial_a Y(t) \end{bmatrix} = \begin{bmatrix} 1 \\ 0 \end{bmatrix} + t \begin{bmatrix} a & 0 \\ 1 & a \end{bmatrix} \begin{bmatrix} Y(Ut) \\ \partial_a Y(Ut) \end{bmatrix}. \tag{14}$$

Notice how this gets rid of the additive branching recursion of equation (13).

Python Code 2.5.3 (implementation of (14))

```python
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]
```

Technique 2.5.4 (recursion in recursion)
Recursion in recursion is what is sounds like. It is like proving an induction step of an induction proof with induction.

Related Work 2.5.5 (recursion in recursion)
The next flight variant of WoS is a beautiful example of recursion in recursion described in [Saw+22].

Most programming languages support recursion but this comes with restrictions like maximum recursion depth and performance issues. When possible tail recursion is a way to implement recursion that solves those issues.

Technique 2.5.6 (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 straightforwardly due to the associativity of all operations ($(xy)z = x(yz)$) involved. However, tail recursion may not always be desirable as it discards intermediate values of the recursion calls which may be of interest. To retain some of these intermediate values while still partly optimizing for performance, it is possible to combine tail recursion with normal recursion.

Python Code 2.5.7 (tail recursion on (14))
We implement (14) but this time with tail recursion. We collect addition operations in a vector *sol* and multiplication in a matrix W.

```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]
```

Related Work 2.5.8 (branching recursion)
This blog discusses branching tail recursion: `https://jeroenvanwijgerden.me/post/recursion-1/`. Interesting techniques like tail recursion gets discussed in [VSJ21].

3 ODEs

3.1 Introduction

In this subsection we discuss informally how to turn ODEs into integral equations mainly by example. Our main tool for this are green functions. Before defining green functions we do some examples.

Example 3.1.1 ($y' = y$)
Let's solve

$$y' = y. \tag{15}$$

but this time with the following condition:

$$\int_0^1 y(s)ds = e - 1.$$

This still has solution $y(t) = e^t$. We define the corresponding source green function $G(t, x)$ for y' and this type of condition as follows:

$$G' = \delta(x - t), \int_0^1 G(s, x)ds = 0.$$

Solving this obtains:

$$G(t, x) = H(t - x) + x - 1.$$

with

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

Note that we could have used a different green function corresponding to a different linear differential operator.

It shouldn't be clear from this point but with this green function we form following integral equation for (15):

$$y(t) = e - 1 + \int_0^1 G(t, s)y(s)ds. \tag{16}$$

Turning equation (16) into a RRVE with recursive MC gives:

$$Y(t) = e - 1 + 2B\left(\frac{1}{2}\right) Y(S)(H(t - S) + S - 1). \tag{17}$$

With $S \sim U$. We will be skipping over the python implementation of equation (17) because it adds nothing new. Instead we plot realizations of equation (17) in Figure 4.

Example 3.1.2 ($y'' = y$)
Lets solve the following boundary problem:

$$y'' = y, y(0) = 1, y'(1) = e. \tag{18}$$

This has solution $y(t) = e^t$. We define the source green function $G(t, x)$ for y'' and Dirichlet/Neumann boundary conditions in the following way:

$$G'' = \delta(t - x), G(0) = 0, G'(1) = 0.$$

Solving this obtains:

$$G(t, x) = \begin{cases} -t & \text{if } t < x \\ -x & \text{if } t \geq x \end{cases}.$$

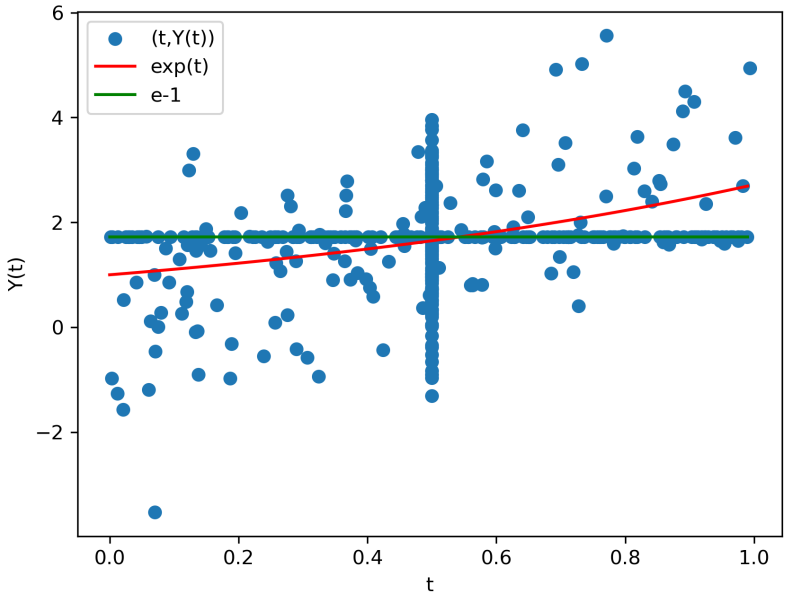

Figure 4: Recursive calls of (17) when calling $Y(0.5)$ 300 times. Points accumulate on the green line due to the Russian roulette, and at $t = 0.5$ because it is the starting value of the simulation.

The boundary green function $P(t, x)$ (for $x \in \{0, 1\}$) for y'' and Dirichlet/Neumann boundary conditions is defined the following way:

$$P'' = 0, (P(0, x), P'(1, x)) = \begin{cases} (1, 0) & \text{if } x = 0 \\ (0, 1) & \text{if } x = 1 \end{cases}.$$

Which is just a basis for the homogenous solutions for now. Solving this gives:

$$P(t, x) = \begin{cases} 1 & \text{if } x = 0 \\ t & \text{if } x = 1 \end{cases}.$$

Again it shouldn't be clear from this point but with these set of green functions we form following integral equation for (18):

$$y(t) = P(t, 0)y(0) + P(t, 1)y(1) + \int_0^1 G(t, s)y(s)ds. \tag{19}$$

Equation (19) looks like:

$$Y(t) = 1 + te + lB\left(\frac{1}{l}\right) G(t, S)Y(S). \tag{20}$$

With $S \sim U$ and $l > 1 \in \mathbb{R}$. We visualize equation (20) on Figure 5.

Definition 3.1.3 (green function)
Vaguely speaking we define the green function as a type of kernel function that we use to solve linear problems with linear conditions. The Green's function is the

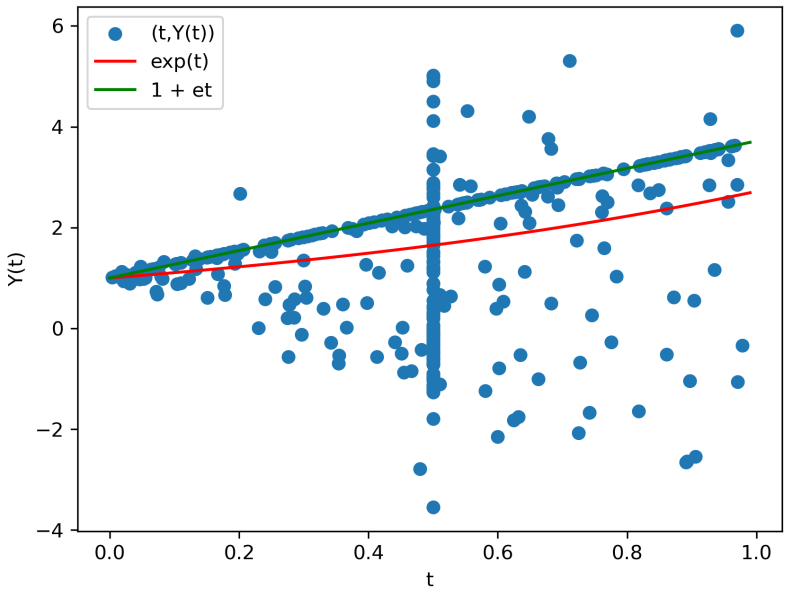

Figure 5: Recursive calls of (20), $l = 2$ when calling $Y(0.5)$ 300 times.

kernel that we need to put in front of the linear conditions or the source term that we integrate over and to obtain the solution and the green function has the property that it satisfies either null linear conditions and a Dirac delta source term, or vice versa.

3.2 Convergence

The ODE examples we have seen so far exhibit rapidly increasing variance as the domain increases, some cases this leads to non-convergence. However, we have not yet discussed convergence criteria for recursive Monte Carlo. Rather than delving deeply into variance or convergence analysis, we will suggest potential solutions in this subsection.

Example 3.2.1 (Dirichlet $y'' = y$)

The following problem will be the main testing example for boundary value problems:

$$y'' = y, y(b_0), y(b_1). \tag{21}$$

The green functions corresponding to y'' and Dirichlet conditions are:

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

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

Straight from these green functions you get following integral equation and RRVE:

$$y(t) = P(t, b_0)y(b_0) + P(t, b_1)y(b_1) + \int_{b_0}^{b_1} G(t, s)y(s)ds \tag{22}$$

$$Y(t) = P(t, b_0)y(b_0) + P(t, b_1)y(b_1) + lB\left(\frac{1}{l}\right)(b_1 - b_0)G(t, S)y(S). \tag{23}$$

With $l \in \mathbb{R}$ the Russian roulette rate and $S \sim \text{Uniform}(b_1, b_0)$.

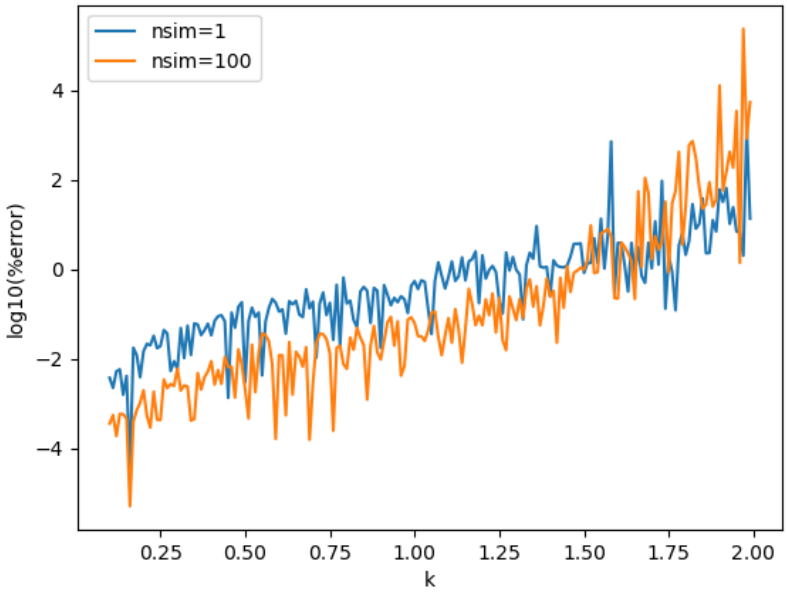

Figure 6: The logarithmic percentage error of $Y(0)$ for (23), with $l = 1.2$ and initial conditions $y(-k) = e^{-k}$ and $y(k) = e^{k}$, displays an exponential increase until approximately $k = 1.5$, beyond which additional simulations fail to reduce the error, indicating that the variance doesn't exist.

Definition 3.2.2 (Fredholm equation of the second kind)
A Fredholm equation of the second kind for φ is of the following form:

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

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

If both K and f are nice, then for sufficiently small λ, it is straightforward to establish the existence and uniqueness of solutions for Fredholm equations of the second kind using a fix point argument.

We would like to have MC algorithm that converges in that case. Our best guess is a combination of splitting and coupling.

Example 3.2.3 (coupled splitting on (3.2.1))

Next to normal splitting (2.2.5) we can also split the domain in equation (22):

$$y(t) = P(t,b_0)y(b_0) + P(t,b_1)y(b_1) + \frac{1}{2}\int_{b_0}^{b_1} G(t,s)y(s)ds + \frac{1}{2}\int_{b_0}^{b_1} G(t,s)y(s)ds \tag{24}$$

$$y(t) = P(t,b_0)y(b_0) + P(t,b_1)y(b_1) + \int_{b_0}^{\frac{b_1+b_0}{2}} G(t,s)y(s)ds + \int_{\frac{b_1+b_0}{2}}^{b_1} G(t,s)y(s)ds \tag{25}$$

Coupling can get rid of the additive branching recursion in the RRVEs corresponding to (24) and (25). Resulting in following RRVE:

$$X(t_1,t_2) = \begin{bmatrix} P(t_1,b_0) & P(t_1,b_1) \\ P(t_2,b_0) & P(t_2,b_1) \end{bmatrix} \begin{bmatrix} y(b_0) \\ y(b_1) \end{bmatrix} + W \begin{bmatrix} G(t_1,S_1) & G(t_1,S_2) \\ G(t_2,S_1) & G(t_2,S_2) \end{bmatrix} X(S_1,S_2). \tag{26}$$

With W the right weighting matrix (see code (3.2.4) for an example) and S_1, S_2 can be chosen in various ways.

Python Code 3.2.4 (implementation of (26))
We implemented equation (26) in example (3.2.3) with recursion but in this case it is actually possible to implement it forwardly because the time proces is nice.

```
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 = 1.04 # 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
```

It appears that Figure 7 resemble a fixed point iterations, leading us to hypothesize that coupled splitting can achieve convergence in most cases where a fixed point argument holds true.

We also conjecture that the convergence speed is very similar to fix points methods until the accuracy of the stochastic approximation of the operator is reached. A way around this is by increasing coupled splitting amount when reaching this bottleneck which can be done very smoothly, creating resembling a multi-grid method. Alternately when reaching the bottleneck it is possible to rely on MC convergence.

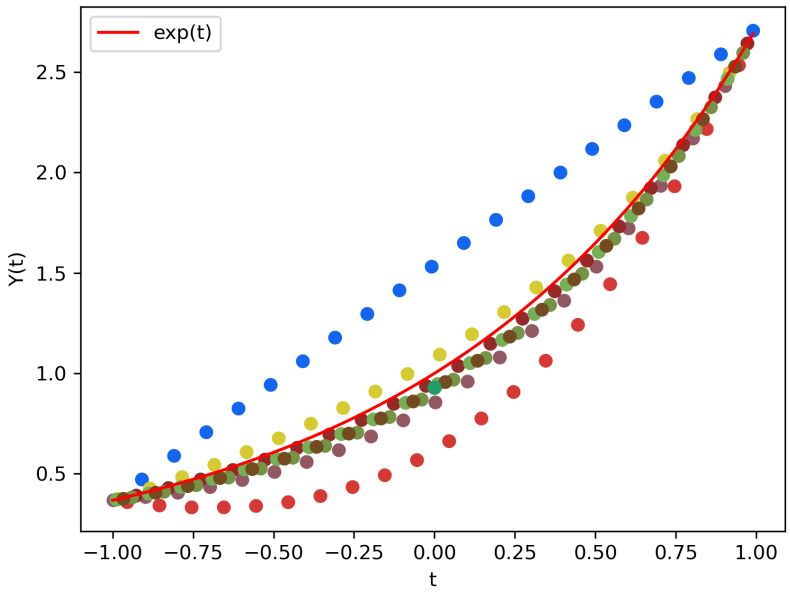

Figure 7: Recursive calls of equation (26) when calling $X(0)$ once, with a split size of 20, S_j coupled such they're equally spaced (they don't have to be independent) and coupling is colored. The initial conditions for this call are $y(-1) = e^{-1}$ and $y(1) = e^1$, with Russian roulette rate $l = 1.2$.

Coupled splitting was tested on the example shown in Figure 6, but it did not contribute to the convergence of the method. This suggests that a fix point argument would not be effective for this particular example with this Russian roulette.

Related Work 3.2.5 (coupled splitting)
Coupled splitting is very similar to the ideas discussed in [SM09]. (the paper on using more rows for MC in linear systems)

Example (3.2.3) is not the best example to demonstrate coupled splitting. First the problem is local and second everything is smooth. Later we discuss algorithms that abuse locality via recursion in recursion and smoothness with control variates. A better use case for this algorithm is for Fredholm equations of the second kind with for example a non-smooth kernel. Similar use cases for MC in differential equations get discussed in [JN09].

explaining that coupled split is cool for global problems but for ODEs and PDEs we are interested in exploiting locality.

3.3 IVPs ODEs

In this subsection we do some IVPs examples.

Right now we don't have a general RMC algorithm that can guarantee a reasonable

variance when increasing the time domain. Classing IVP solvers rely on shrinking the time steps for convergence. Recursion in Recursion MC (RRMC) for IVPs tries to emulate this behavior.

Example 3.3.1 (RRMC $y' = y$)
() Let's us explain RRMC for IVPs with our main example. Imagine we have a time stepping sheme (t_n) $(t_n > t_{n-1})$ then following integral equations hold:

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

Turn these in following class of RRVEs:

$$Y_n(t) = y(t_n) + (t - t_n)Y_n((t - t_n)U + t_n), t > t_n.$$

A problem with these RRVEs is that we don't know $y(t_n)$. Instead we can replace it with an unbiased estimate y_n which we keep frozen:

$$Y_n(t) = y_n + (t - t_n)Y_n((t - t_n)U + t_n), t > t_n \tag{27}$$

$$y_n = \begin{cases} Y_{n-1}(t_n) & \text{if } n \neq 0 \\ y(t_0) & \text{if } n = 0 \end{cases}. \tag{28}$$

We refer to equation (27) as the inner recursion and equation (28) as the outer recursion of the recursion in recursion.

Python Code 3.3.2 (implementation of (3.3.1))

talk about how to get higher order convergence with control variates

Example 3.3.3 (control RRMC $y' = y$)
example of control variated RRMC

talk about non-linearity and bias, Newton (linearized in time) and maybe how other methods deal with nonlinearity and a fix by special CVs strategies.

Example 3.3.4 (nonlinear RRMC)
We made an example to show that RRMC works for non linear problems but it is biased. The inner recursions are correlated because they use the same info from the outer recursions, this doesn't mean that reducing variance by splitting doesn't work , you just have to be careful with the bias. (see statistical learning they have a whole set of tools to estimate performance when considering bias vs variance)
Consider:

$$y' = y^2 - t^4 + 2t, y(0) = 0.$$

with solution: $y(t) = t^2$. With integral equation:

$$y(t) = y(T) + \int_{T}^{t} y^2(s)ds - \frac{t^5 - T^5}{5} + (t^2 - T^2).$$

control variating $y^2(s)$ up to second order:

$$\begin{aligned} y^2(t) &\approx y^2(T) + 2(t - T)y(T)y'(T) + ((t - T)y'(T))^2 + O((t - T)^2) \\ &\approx y^2(T) + 2(t - T)y(T)y'(T) + O((t - T)^2) \end{aligned}$$

we could have kept the third term we think it would be more accurate but we're lazy. Then we have to integrate the control variate:

$$\int_T^t y^2(T) + 2(s-T)y(T)y'(T)ds$$
$$= (t-T)y^2(T) + 2\left(\frac{t^2-T^2}{2} - T(t-T)\right)y(T)y'(T)$$

.

talk about stability and then conjecture a solution DRRMC

Example 3.3.5 (DRRMC)

diagonal RRMC

talk about performance stability and maybe comparison to parareal (critical path and how to measure performance)

3.4 BVPs ODEs

Example 3.4.1 (local BVP)

a RRMC algo for dirichlet see period 5

discuss getting higher order

Example 3.4.2 (higher order local BVP)

a higher order RRMC algo + convergence

4 Recursive Brownian Motion

4.1 Brownian Motion

introduces Brownian motion

4.2 Boundary Green Function Of The Heat Equation

introduce with 1D case, numerically calculating it

4.3 Recursive Brownian Motion

WoS like way to simulate Brownian motion which is related to the green function of the heat equation, this

Related Work 4.3.1 (recursive Brownian motion)

see Brownian motion tag in zotero

4.4 Heat Equation

a geometric robust way to solve the heat equation and maybe a higher order method to solve the heat equation

References

[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).

[Gil13] Michael B. Giles. *Multilevel Monte Carlo methods*. en. arXiv:1304.5472 [math]. Apr. 2013. URL: http://arxiv.org/abs/1304.5472 (visited on 12/04/2022).

[JN09] A. Jentzen and A. Neuenkirch. "A random Euler scheme for Carathéodory differential equations". en. In: *Journal of Computational and Applied Mathematics* 224.1 (Feb. 2009), pp. 346–359. ISSN: 03770427. DOI: 10.1016/j.cam.2008.05.060. URL: https://linkinghub.elsevier.com/retrieve/pii/S0377042708002136 (visited on 03/06/2023).

[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).

[Vea] Eric Veach. "ROBUST MONTE CARLO METHODS FOR LIGHT TRANSPORT SIMULATION". en. In: ().

[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).

[Wu20] Yue Wu. *A randomised trapezoidal quadrature*. en. arXiv:2011.15086 [cs, math]. Dec. 2020. URL: http://arxiv.org/abs/2011.15086 (visited on 03/06/2023).

5 Appendix

Derivation of the green functions and some expressions.