Astin Bulletin (2026), **56**, pp. 127–153
doi:10.1017/asb.2025.10078

International Actuarial Association
Association Actuarielle Internationale

RESEARCH ARTICLE

Collaborative and Parametric Insurance on the Ethereum Blockchain

Pierre-Olivier Goffard[1] and Stéphane Loisel[2]

[1]Institut de Recherche Mathématique Avancée, Université de Strasbourg, Strasbourg, France
[2]CNAM, Efab, Lirsa, Paris, France
Corresponding author: Pierre-Olivier Goffard; Email: goffard@unistra.fr

Received: 3 December 2024; **Revised:** 30 August 2025; **Accepted:** 10 October 2025; **First published online:** 7 November 2025

Keywords: Blockchain; parametric insurance; actuarial science

MSC 2020: 91B30

Abstract

This article introduces a blockchain-based insurance scheme that integrates parametric and collaborative elements. A pool of investors, referred to as surplus providers, locks funds in a smart contract, enabling blockchain users to underwrite parametric insurance contracts. These contracts automatically trigger compensation when predefined conditions are met. The collaborative aspect is embodied in the generation of tokens, which are distributed to surplus providers. These tokens represent each participant's share of the surplus and grant voting rights for management decisions. The smart contract is developed in Solidity, a high-level programming language for the Ethereum blockchain, and deployed on the Sepolia testnet, with data processing and analysis conducted using Python. In addition, open-source code is provided and main research challenges are identified, so that further research can be carried out to overcome limitations of this first proof of concept.

1. Introduction

A blockchain is a decentralized ledger maintained by consensus within a peer-to-peer network. The nodes in this network agree on a shared history of data by executing an algorithm known as a consensus protocol. The Bitcoin blockchain, as described by Nakamoto (2008), simply records transactions between bitcoin users. In contrast, the Ethereum blockchain, introduced by Buterin (2014), is also equipped with a compiler to execute pieces of code called smart contract. Ethereum is designed to operate as a "world computer," enabling developers to deploy decentralized applications (dApps) that users can interact with directly.

Through these decentralized applications, Ethereum extends the capabilities of blockchains beyond simple fund transfers, enabling more complex financial operations and fostering the emergence of a new financial ecosystem known as decentralized finance (DeFi), as described by Schär (2020). Traditional finance relies on trusted third parties and centralized databases, which involve intermediary costs, lack transparency, operate only during limited hours, and are susceptible to cyberattacks. In contrast, distributed ledgers are publicly accessible, replicated in multiple locations, and function continuously, 24/7. Furthermore, the code that governs decentralized applications is auditable, improving transparency. For further discussion, we refer the reader to Lipton and Treccani (2021), Chapter 2.

Some criticisms of the financial system can also be extended to the insurance sector. Insurance policies are often complex and difficult for the general public to understand. Premium calculations are typically based on historical data that are privately held by insurance companies, which limits transparency. In addition, the claim settlement process is lengthy and costly as it requires onsite assessments by one or more experts.

© The Author(s), 2025. Published by Cambridge University Press on behalf of The International Actuarial Association.

https://doi.org/10.1017/asb.2025.10078 Published online by Cambridge University Press

One response to these challenges is parametric insurance, also known as index-based insurance. Parametric insurance provides compensation based on predefined triggers, such as specific weather conditions, rather than relying on traditional, often time-consuming, claims assessments. The main advantages include fast payouts and reduced management costs. When the verification of the triggering condition is done using a reliable and independent data source, it increases policyholders' confidence in the system.

Parametric insurance is not a new concept. It emerged in Asia during the 1990s, allowing farmers to buy protection against income loss due to poor harvests following adverse weather conditions. Since then, parametric insurance has been used to cover perils like natural catastrophes. Notable examples include the Caribbean Catastroph Risk Insurance Facility (CCRIF), launched in 2007, and the African Risk Capacity (ARC) founded in 2014. Fast payouts in the context of natural disaster relief are crucial, as they enable the rapid deployment of resources by government agencies and nongovernmental organizations providing immediate assistance.

In recent years, parametric insurance schemes have emerged in mass-market products, such as flight delay insurance offered by China Southern Airlines, which provides payouts of up to 300 yuan (approximately US$40) for delays of three hours or more. Note that death benefits in traditional insurance often consist of a fixed amount that varies according to limited number of parameters, such as the number of children or marital status. This forms a prominent example of a parametric insurance contract that is widely accepted and commercialized.

The main downside of parametric insurance is basis risk, where the insured's losses may not match the coverage amount, or the parameter may not trigger. Effective structuring and pricing require understanding the policyholder's exact risks and choosing the right parameter. The availability of finer granularity data and advances in predictive modeling can mitigate basis risk. Hybrid insurance solutions that combine parametric and standard indemnity mechanisms can also be effective, see for instance Lopez and Nkameni (2025). An initial payment following the parametric trigger can provide immediate relief, and this amount can later be deducted from the compensation resulting from the traditional claims assessment process. The policyholder has the option to either maintain parametric insurance or purchase an additional traditional insurance contract to cover basis risk.

For a historical overview of parametric insurance, we refer the reader to the press article by Clere (2022) in InsurTech Digital magazine. Interesting discussions on the use of parametric insurance to address climate risk issues can be found on the websites of the World Economic Forum (2025) and the National Association of Insurance Commissioners (2023). The application of parametric insurance to crop insurance has been discussed in the literature (see Abdi *et al*., 2022 and Conradt *et al*., 2015). The simplicity of the insurance mechanism makes it suitable for encoding in a smart contract. A blockchain can store data immutably and transparently, making the information associated with the triggering event and compensation available to all interested parties. Connecting the blockchain to users' wallets allows for fast and automated processing of indemnity payments.

We present a smart contract on the Ethereum blockchain to manage a portfolio of parametric insurance policies. We have developed an actuarial framework suited for weather-linked triggering events, specifically monitoring daily rainfall. This framework establishes ground rules for underwriting, rate-making, and managing risk. Our smart contract is designed to operate with minimal human interference, maximizing automation. Every interaction with the smart contract incurs a cost, as any transaction submitted to the blockchain requires a fee to be processed by the network of nodes. Therefore, optimizing the number of transactions is crucial for reducing operational costs. Actuarial methods for calculating risk capital are often too complex to be directly implemented in a high-level language like Solidity, which, for instance, only supports integer-based arithmetic. Additionally, deploying a smart contract on the blockchain entails costs that grow with its complexity, emphasizing the need for simplicity in design. One of our key contributions is the simplification of these computations through justified approximations. In particular, we propose a straightforward recursive formula to determine the solvency capital required to adequately cover the risks of a parametric insurance portfolio.

https://doi.org/10.1017/asb.2025.10078 Published online by Cambridge University Press

The smart contract structure is inspired by the discussion in Cousaert *et al*. (2022), from which we adopt the role distribution and the "tokenization" mechanism of risk. The smart contract owner writes the code and sets the initial values for parameters such as the premium loading and the risk level required for risk capital allocation. Investors participate by locking funds in the smart contract, in exchange for which they receive protocol tokens. These funds are denominated in ETH, the native cryptocurrency of the Ethereum blockchain. The contribution of each investor is reflected in their token holdings. The value of the token, or equivalently its exchange rate against ETH, fluctuates over time depending on whether compensation are being paid to policyholders. Conversely, the token value increases if no compensation is paid at settlement. Token ownership enables governance of the insurance protocol. The contract owner may update the smart contract parameters based on recommendations from a board composed of token holders. This governance model transforms the smart contract into a Decentralized Autonomous Organization (DAO). The tokenization of risk on the blockchain can be viewed as a novel way to design insurance-linked securities, as outlined in Barrieu and Albertini (2012). By enabling policyholders to become investors, our smart contract creates a collaborative insurance scheme, akin to those described in Feng (2023). Investors, referred to as surplus providers (SPs), are analogous to liquidity providers in decentralized exchanges, which serve as blockchain-based trading venues for crypto assets; see, for instance, Mohan (2022).

Future policyholders interact with the protocol to seek protection against adverse events. An insurance agreement can only be concluded if the funds provided by the investors are sufficient to cover the risk. The smart contract funds must remain above a specified threshold at all times, as prescribed by standard insurance regulations such as the European directive Solvency II.

We showcase the concrete implementation of our framework using the Solidity programming language. This includes instructions on deploying the contract on the Ethereum testnet, interacting with it, and retrieving data via the Etherscan Application Programming Interface (API). For a comprehensive overview of how the Ethereum blockchain operates and a tutorial on coding in Solidity, we refer the reader to the book of Antonopoulos and Gavin Wood Ph (2018).

This paper aims to show the feasibility of deploying a beta version of such a smart contract, where certain complexities are intentionally simplified. We also identify the main limitations of this simplified framework and provide the research and actuarial communities with an open-source beta version of the smart contract. This prototype serves as a foundation for developing generalizations and improved versions that address these limitations.

The rest of the paper is organized as follows. Section 2 provides a brief overview on blockchain technology. Section 3 presents an actuarial framework for managing parametric insurance contract portfolios. Section 4 focuses on the design of the smart contract. We introduce the participants, including the contract owner, SPs, and policyholders, in Section 4.1. A smart contract is characterized by state variables, which we describe in Section 4.2. Examples of state variables include the smart contract's balance and the tokens held by each participant. Users interact with the smart contract by invoking functions, and the possible actions and their impacts on the system's state are detailed in Section 4.3. Section 5 demonstrates how the smart contract operates through an event-driven scenario, showcasing all its functions and culminating in the liquidation and reset of the smart contract. The main limitations of this beta version of the smart contract are discussed in Section 6, along with potential ideas for addressing them.

2. Blockchain technology and smart contracts

The concept of blockchain technology was first introduced with Bitcoin, a decentralized digital currency launched in 2008 with Nakamoto (2008). Bitcoin's blockchain is a public ledger that records all transactions in a secure, immutable, and transparent manner. Its core innovation lies in achieving consensus among distributed nodes without the need for a central authority, using a Proof-of-Work (PoW) mechanism.

https://doi.org/10.1017/asb.2025.10078 Published online by Cambridge University Press

Table 1. *Examples of decentralized applications across sectors.*

Category	Project	Description	URL
Finance (DeFi)	Uniswap	A decentralized exchange (DEX) that allows users to swap ERC-20 tokens using liquidity pools and an automated market maker model	https://uniswap.org
Insurance	Etherisc	Provides decentralized parametric insurance, e.g., crop insurance for farmers, using smart contracts and weather data oracles	https://etherisc.com
Governance	Aragon	A platform for building and managing decentralized autonomous organizations (DAOs) with voting and fund management capabilities	https://aragon.org
Digital Ownership	Decentraland	A virtual reality platform on Ethereum where users can buy, develop, and monetize virtual plot of land using NFTs (Non-Fungible Tokens)	https://decentraland.org

However, the Bitcoin blockchain was designed with a narrow scope: it enables peer-to-peer transfer of value but offers very limited programmability. As a result, more complex applications, such as decentralized financial instruments or autonomous organizations, are not feasible on the Bitcoin network. To overcome these limitations, the concept of *programmable blockchains* emerged, most notably with the introduction of Ethereum in 2015, see Buterin (2014). Ethereum extends the blockchain model by embedding a *Turing-complete virtual machine*, the Ethereum Virtual Machine (EVM), which enables the execution of code on-chain. This innovation allows developers to write *smart contracts* – autonomous programs that run exactly as programmed – opening the door to a wide range of decentralized applications (dApps) across finance, insurance, governance or digital ownership, some concrete projects are mentionned in Table 1. "ERC-20 tokens" refer to fungible tokens on the Ethereum blockchain that are interchangeable and used for various transactions and financial operations, while "NFTs" represent unique digital assets, such as collectibles or in-game items, that are not interchangeable and verify ownership on the blockchain. Token generation is a central feature of the Ethereum blockchain as it enables the creation of diverse digital assets that power decentralized applications and facilitate transactions between users.

Ethereum is currently the most widely adopted programmable blockchain, and as such was selected for the implementation of our parametric insurance framework. Ethereum benefits from the largest and most active developer community among blockchain platforms. This has led to a wealth of tooling, documentation, libraries (e.g., Hardhat, Truffle, `web3.js`), and community support, which greatly facilitate the development and deployment of decentralized applications. Many formal verification tools and auditing services are tailored for Ethereum-based contracts, increasing trust and reducing the likelihood of critical vulnerabilities. The Ethereum blockchains successfully transited from PoW to Proof-of-Stake inducing a significant reduction of its energy consumption and improved network scalability, helping to ensure long-term sustainability and compliance with environmental standards. Lastly, Ethereum hosts the majority of DeFi protocols, NFT platforms, and DAO frameworks. Its large user base and vibrant application ecosystem make it the natural choice for projects that require integration with or exposure to these decentralized systems. We acknowledge the existence of other programmable blockchains that offer valuable features. Binance Smart Chain (BSC) offers compatibility with Ethereum's EVM but is more centralized, with fewer validators, potentially limiting trust and decentralization. Solana provides high throughput and low latency but suffers from frequent network outages and a more complex

https://doi.org/10.1017/asb.2025.10078 Published online by Cambridge University Press

programming environment. Avalanche supports custom blockchains and fast finality, but its developer ecosystem and tooling are less mature compared to Ethereum.

3. Parametric insurance portfolio management framework

This section presents an actuarial framework for parametric insurance tailored to an implementation on the Ethereum blockchain. While this framework supports our analysis, we emphasize that it is not the primary contribution of our work. We further acknowledge that it could be extended or refined to accommodate a broader class of insurance solutions.

3.1. Parametric insurance contract over time

A parametric insurance contract pays a predefined compensation l if some observable quantity $(Q_t)_{t\geq 0}$ reaches a threshold $\overline{Q}$ at some time T. The quantity $Q := (Q_t)_{t\geq 0}$ is a stochastic process defined on a filtered probability space $(\Omega, \mathcal{F}, (\mathcal{F}_t)_{t\geq 0}, \mathbb{P})$. A compensation is paid if the $\mathcal{F}_T$-measurable event: $\{Q_T > \overline{Q}\}$ occurs. A parametric insurance contract is represented as a 5-tuple $(S, T, Q, \overline{Q}, l)$, where S denotes the time at which the contract is underwritten. We outline in Remarks 3.1 and 3.2 two extensions associated to the time of the triggering event and the amount compensated, respectively.

Remark 3.1. When the trigger is based on weather condition, the insurance coverage usually concerns a time interval. The triggering event definition must be suited to the meteorological quantity we monitor. We consider daily rainfalls as an example later in the article and so a typical event would be

"The cumulative precipitation on the 1st of May, 2025 in Strasbourg, France, exceeded 10 mm".

We therefore consider Q as a discrete time stochastic process or a time series corresponding to the amount rain that fell on a specific day. A continuous time model would be better suited for other situations like flood risk for example. The trigger would be based on the cumulated rain over a rolling window of several days. If we consider windspeed, then time T should be replaced by a time interval (T_-, T_+) and the triggering event then becomes

$$\left\{\max_{t\in(T_-,T_+)} Q_t > \overline{Q}\right\}, \text{ or } \left\{\min_{t\in(T_-,T_+)} Q_t > \overline{Q}\right\},$$

where Q is a continuous time stochastic process. Our approach could be extended to accomodate such refinements.

Remark 3.2. The compensation can be a function $l : Q_T \mapsto l(Q_T)$ of the quantity Q_T. Here, we only consider a pre-specified lump sum payment.

Define the status of the contract over time as

$$\xi_t = \begin{cases} 0, & \text{if } t < T, \\ 1, & \text{if } t \geq T \text{ and } Q_T \leq \overline{Q}, \\ 2, & \text{if } t \geq T \text{ and } Q_T > \overline{Q}. \end{cases} \tag{1}$$

The state space of $(\xi_t)_{t\geq 0}$ is $\mathcal{S} = \{0, 1, 2\}$. Each state corresponds to a specific condition of the contract:

$0 =$ "open," $1 =$ "closed without compensation," and $2 =$ "closed with compensation."

The smart contract may be forced to file for bankruptcy if the surplus falls below the minimum capital requirement (MCR) at any point between S and T. If this occurs, the remaining active policies are

https://doi.org/10.1017/asb.2025.10078 Published online by Cambridge University Press

cancelled, and the premiums are refunded to the policyholders. This possibility implies to add a fourth state

$$3 = \text{“cancelled,”}$$

which is covered in Section 4.3.4.

The pure premium associated with a contract $(S, T, Q, \overline{Q}, l)$ is given by

$$\mathbb{P}(Q_T > \overline{Q}|\mathcal{F}_S) \cdot l.$$

The probability of the event $\{Q_T > \overline{Q}\}$ estimated at some time $t \in [0, T)$ is a stochastic process defined by

$$\theta_t = \mathbb{P}(Q_T > \overline{Q}|\mathcal{F}_t) \text{ for } t \in [0, T). \tag{2}$$

The filtration $(\mathcal{F}_t)_{t\geq 0}$ gathers all the available information at time $t \leq T$ relevant to the occurrence of event $\{Q_T > \overline{Q}\}$ or equivalently the study of the quantity $(Q_t)_{t\geq 0}$. For weather-linked events, the quantity Q_T may be forecasted based on the available information at time $t < T$ using weather forecast models. Concretely, the filtration $(\mathcal{F}_t)_{t\geq 0}$ contains both observations and predictions available at time $t \geq 0$, and this must be accounted for in our actuarial pricing framework. Constraints on S can be imposed to mitigate the risk of adverse selection as outlined in Remark 3.3.

Remark 3.3. From a practical standpoint, we can assume the existence of a time $S^* = T - \Delta < T$ before which the triggering event cannot be reliably anticipated. Therefore, the estimation of the probability of $\{Q_T > \overline{Q}\}$ relies solely on historical, observed data. If policyholders have access to a more accurate predictive model than that of the insurers, it could lead to adverse selection. A straightforward approach to mitigate this risk is to stipulate that a policy $(S, T, Q, \overline{Q}, l)$ cannot be underwritten if $S > S^*$. We can assume that

$$\mathbb{P}(Q_T > \overline{Q}|\mathcal{F}_s) = \theta, \text{ for any } s \leq S^*.$$

We consider as a use case a parametric insurance coverage against rain episodes. An illustration is provided in Example 1 with a concrete estimation of the probability θ.

Example 1. *Consider the insurance policy $(S, T, Q, \overline{Q}, l)$, where $Q = (Q_t)_{t\geq 0}$ is the daily precipitation height measured at the meteo station of Strasbourg airport (Entzheim). We assume that $S \leq S^*$. We consider a parametric model for the daily precipitation height where $Q_T|\mathcal{F}_S$ is distributed as a compound Poisson-gamma random variable. We have*

$$Q_T|\mathcal{F}_S \overset{\mathcal{D}}{=} \sum_{k=1}^{N_T} U_{k,T},$$

where N_T is a Poisson random variable $\mathsf{Poisson}(\lambda_T)$ *and the U_k are* IID *gamma random variables* $\mathsf{Gamma}(\alpha_T, \beta_T)$ *independent from N_T. This model was proposed by Dunn (2004) with N_T corresponding to the number of rain episodes during a day and the U_k's characterize their intensity. We assume that the parameters of the model λ_T, α_T, and β_T simply depend on the month associated with the date T. We calibrate this model using daily data from the years 2000 to 2023 provided by Météo France and retrieved from the data.gouv.fr. The time series is shown in Figure 1.*

The model is fitted using a simple partial method of moments so that $\widehat{\lambda} = -\log(\widehat{p}_0)$, where $\widehat{p}_0$ is the proportion of days without rain episodes over a month. We further have

$$\widehat{\alpha} = \frac{\mu^2}{\widehat{\lambda}\sigma^2 - \mu^2} \text{ and } \widehat{\beta} = \frac{\mu}{\widehat{\lambda}\widehat{\alpha}},$$

where μ and σ denote the empirical mean and variance of the daily precipitations for rainy days. This simple method is described for instance in Goffard et al. (2022). We have a compound Poisson-Gamma model for the STRASBOURG-ENTZHEIM station for each month; the estimations of the parameters allow us to estimate the number of rain episodes and their intensity for each month as shown in Figure 2.

https://doi.org/10.1017/asb.2025.10078 Published online by Cambridge University Press

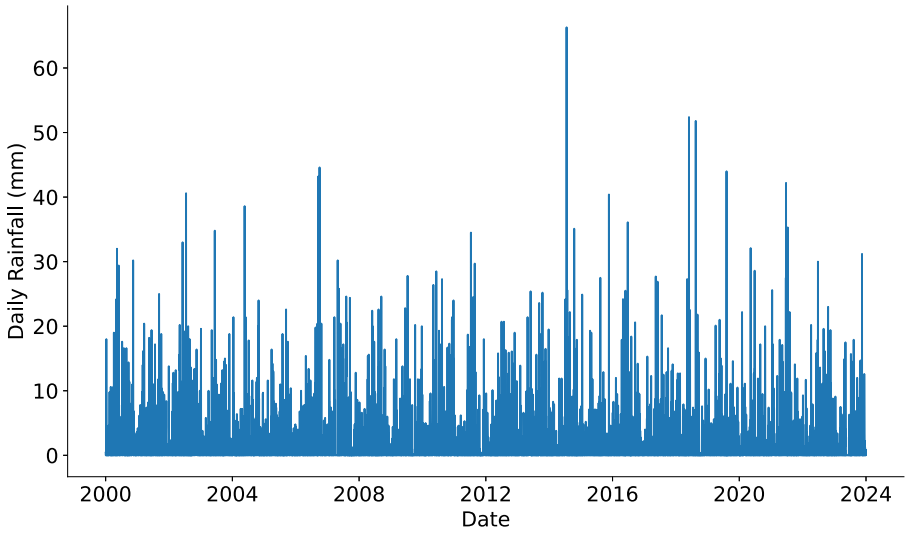


Figure 1. *Daily rainfall measurements recorded at the STRASBOURG-ENTZHEIM station, spanning from January 1, 2000, to December 31, 2023.*

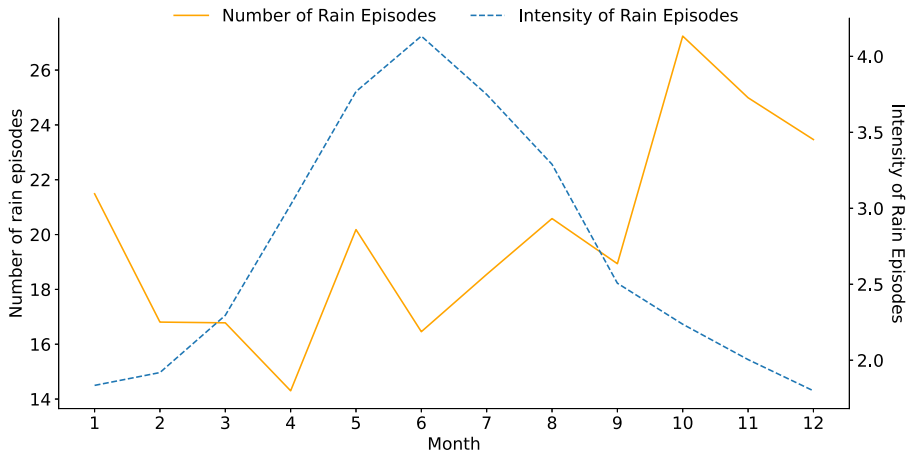


Figure 2. *Number of rain episodes per month and intensity of the rain episodes depending on the month in Strasbourg.*

The climate in Strasbourg is "semi-continental". In terms of precipitation, it is characterized by low precipitation levels during the winter months (replaced by snow) and few but intense rain episodes during the summer months. The model then provides us with the probability θ associated with the threshold $\overline{Q}$ for each month, see Figure 3.

The estimation of θ yields the pure premium of the contract as $\theta \cdot l$.

Remark 3.4. The pricing of our parametric insurance contract relies on a piecewise constant function $T \mapsto \theta(T)$. From a practical perspective, dealing with dates in Solidity (the programming language of the Ethereum blockchain) is not straightforward; furthermore, having to handle 12 clusters of dates is tedious. Our solution consists of writing T as an integer number between 1 and 365 and approximating the function $T \mapsto \theta(T)$ through a polynomial. In our smart contract, we only consider one threshold with $\overline{Q} = 5$. The polynomial approximation of degree 4 is provided in Figure 4.

https://doi.org/10.1017/asb.2025.10078 Published online by Cambridge University Press

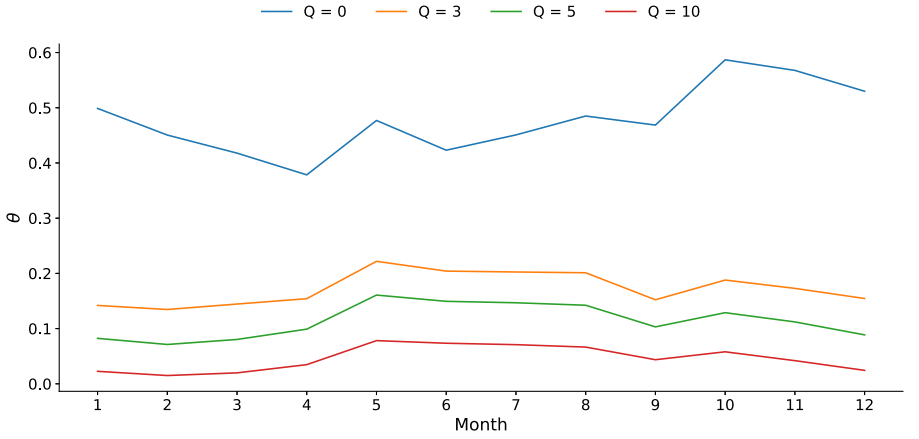


Figure 3. *Estimation of the probability* $\mathbb{P}(Q_T > \overline{Q})$ *depending on the month and the threshold* $\overline{Q} \in \{0, 3, 5, 10\}$.

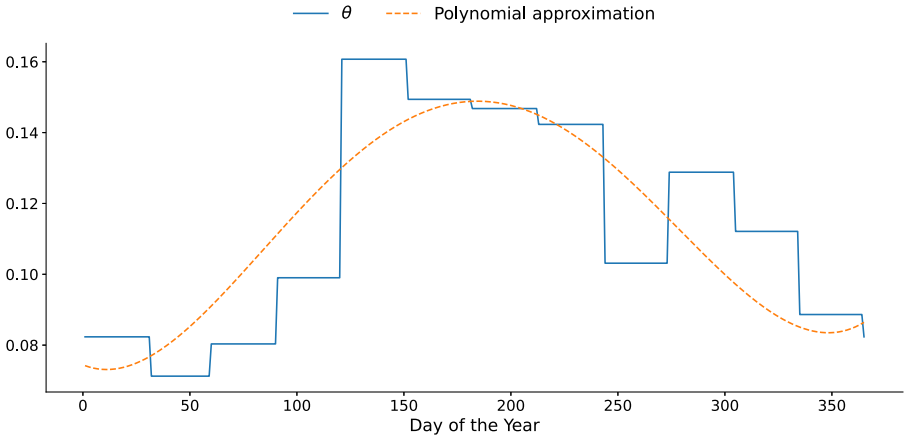


Figure 4. *Estimation of the probability* $\mathbb{P}(Q_T > 5)$ *depending on the day of the year.*

Concretely, we use Python to find the coefficients of the polynomial approximation:

$$\theta(T) \approx a_0 + a_1 \cdot T + a_2 \cdot T^2 + a_3 \cdot T^3 + a_4 \cdot T^4,$$

before hardcoding the formula in Solidity. We note in passing that Solidity could allow us to encode a pricing formula that could result from the fitting of a logistic regression model, which is suitable for modeling the probability of occurrence of an event, provided that the model does not include too many covariates. The use of logistic regression for ratemaking purposes of parametric insurance contracts has been explored in Figueiredo *et al*. (2018).

Suppose that $S \leq S^*$, where S^* has been defined in Remark 3.3. The commercial premium is then given by

$$\pi_S = (1 + \eta_S) \cdot \theta \cdot l,$$

using the expectation premium principle, where $\eta_S > 0$ is the loading factor at time S to ensure profitability on average and cover management costs. The safety loading evolves over time according to

https://doi.org/10.1017/asb.2025.10078 Published online by Cambridge University Press

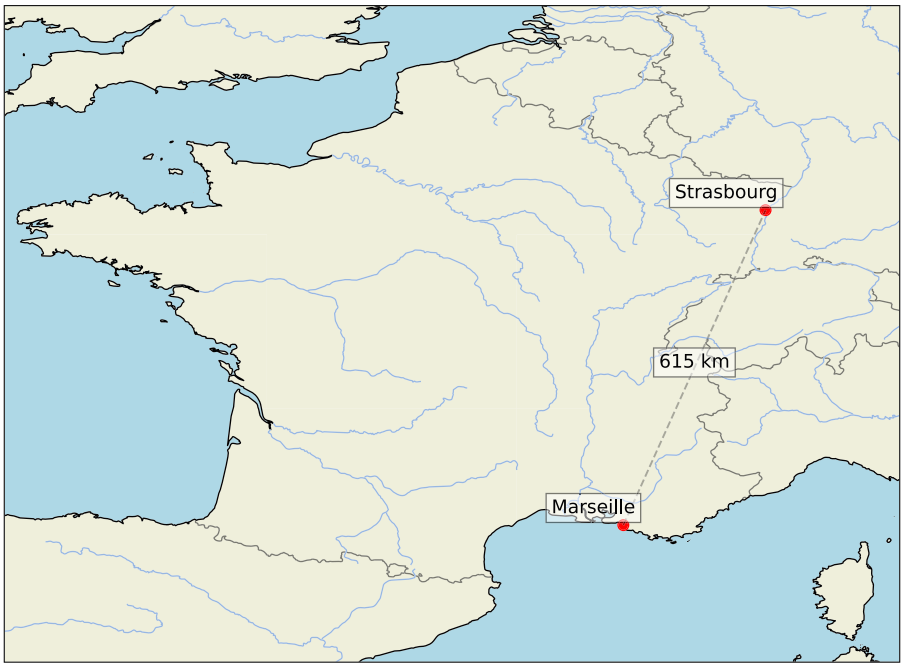


Figure 5. *Location of Marseille and Strasbourg in France.*

managerial decisions based on the available information $\mathcal{F}_t$. It constitutes an $\mathcal{F}_t$-measurable stochastic process denoted by $(\eta_t)_{t\geq 0}$.

Remark 3.5. An insurance coverage $(S, T, Q, \overline{Q}, l)$ cannot be purchased after time S^*. To further mitigate the risk of adverse selection, the safety loading could also be adjusted to incentivize early purchases – specifically, by offering a premium discount to customers who buy the coverage well before S^*.

3.2. Parametric insurance portfolio over time

A parametric insurance portfolio is a collection of parametric insurance contracts $\{(S_i, T_i, Q_i, \overline{Q}_i, l_i)\}_{i\geq 1}$. These contracts may be associated with their own observable quantities $Q_i := \left(Q_t^i\right)_{t\geq 0}$ defined on a common probability space $(\Omega, \mathcal{F}, (\mathcal{F}_t)_{t\geq 0}, \mathbb{P})$. Let us take a concrete situation in Example 2 by following up on Example 1.

Example 2. *In addition to Strasbourg, we wish to offer coverage against rain episodes in the city of Marseille. We assume that Strasbourg and Marseille are sufficiently far apart (see the map on Figure 5) so that the precipitation levels of these cities are not correlated. Each contract i is then associated with a location so that* $Q^i \in \{Q^{Marseille}, Q^{Strasbourg}\}$ *and a date* T_i *for all* $i \geq 1$. *We assume that* $Q^{Marseille}$ *and* $Q^{Strasbourg}$ *are mutually independent. We assume that* $S_i < S_i^*$ *for each contract so as to model the precipitation height via a compound Poisson-gamma distribution at each location. The models are fitted to the same data as in Example 1 and result in two different risk profiles as illustrated on Figure 6, where we plot the probability* $\mathbb{P}(Q_T^i > 5)$ *as a function of the month of T for each location i.*

The number of contracts underwritten up to time $t \geq 0$ is modeled as an $\mathcal{F}_t$-adapted counting process, defined as

$$N_0 = 0, \quad N_t = \sum_{i\geq 1} \mathbb{I}_{S_i \leq t}.$$

https://doi.org/10.1017/asb.2025.10078 Published online by Cambridge University Press

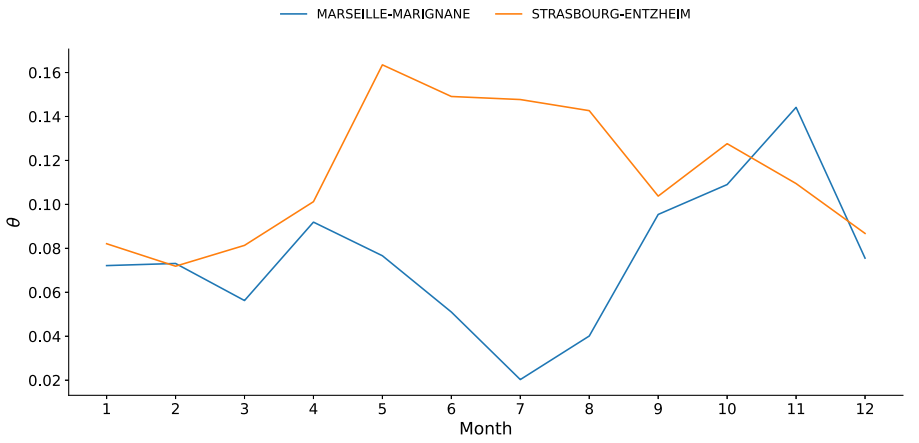


Figure 6. *Estimation of the probability $\mathbb{P}(Q_T > 5)$ depending on the month and the location.*

The statuses of the contracts are stored in a vector $\Xi_t = \left(\xi_t^{(1)}, \ldots, \xi_t^{(N_t)}\right)$. Denote by $Y_i = \mathbb{I}_{Q^i_{T_i} > \bar{Q}_i} \cdot l_i$ the pay-out of insurance contract i, where $\left(\mathbb{I}_{Q^i_{T_i} > \bar{Q}_i}\right)_{i=1,\ldots,N_t}$ is a sequence of Bernoulli variables with parameters

$$\theta_i = \mathbb{P}(Q^i_{T_i} > \bar{Q}_i | \mathcal{F}_{S_i}) \text{ for } i = 1, \ldots, N_t.$$

The liability at time $t > 0$ is a stochastic process defined as

$$L_0 = 0, \quad L_t = \sum_{i=1}^{N_t} Y_i \mathbb{I}_{S_i \leq t, T_i > t}, \quad \text{for } t \geq 0. \tag{3}$$

It corresponds to the future losses given the active policies in our portfolio at time t.

Remark 3.6. Note that the assumption of fixed-parameter Bernoulli variables Y_i with parameters is an approximation. In reality, the event probabilities follow stochastic processes, recall the definition (2). A more precise approach would involve dynamically updating these probabilities when analyzing the liability distribution at time t. However, we adopt this simplification to avoid the prohibitive transaction costs that would arise from real-time updates in a blockchain environment, which would require frequent oracle queries and solvency capital reevaluations.

The premiums collected to compensate for this liability are given by

$$\Pi_t = \sum_{i=1}^{N_t} \pi_i \mathbb{I}_{S_i \leq t, T_i > t}, \quad \text{for } t \geq 0,$$

where

$$\pi_i = (1 + \eta_{S_i}) \cdot \theta_i \cdot l_i, \quad \text{for } i = 1, \ldots, N_t.$$

We refer to Π_t as the collected but yet to be earned premiums. We define two solvency capitals as

$$SCR_t = \text{Quantile}(L_t - \Pi_t\,; \alpha_{\text{SCR}}), \quad \text{and} \quad MCR_t = \text{Quantile}(L_t - \Pi_t\,; \alpha_{\text{MCR}}). \tag{4}$$

Here, *SCR* stands for Solvency Capital Requirement, and *MCR* stands for Minimum Capital Requirement, by analogy with the European directive Solvency II, under which $\alpha_{\text{SCR}} = 0.995$ and $\alpha_{\text{MCR}} = 0.85$.

https://doi.org/10.1017/asb.2025.10078 Published online by Cambridge University Press

3.3. Solvency capital calculation

3.3.1. Model points creation to handle dependency

The main challenge associated with calculating the risk capitals (4) lies in the study of the distribution of the liability $(L_t)_{t\geq 0}$, more specifically the likely dependency among the Bernoulli variables. We show in Example 3 how to gather two correlated contracts into one within the framework of Examples 1 and 2.

Example 3. *Consider two contracts* $\left(S_{i_1}, T_{i_1}, Q_{i_1}, \overline{Q}_{i_1}, l_{i_1}\right)$ *and* $\left(S_{i_2}, T_{i_2}, Q_{i_2}, \overline{Q}_{i_2}, l_{i_2}\right)$ *associated with the same location* $Q = Q_{i_1} = Q_{i_2}$ *and the same event date* $T = T_{i_1} = T_{i_2}$. *The two contracts do not have the same threshold nor the same underwriting time. We assume without loss of generality that* $\overline{Q}_{i_1} \leq \overline{Q}_{i_2}$ *and* $S_{i_1} < S_{i_2}$. *We assume that* S_{i_1} *and* S_{i_2} *are both sufficiently far from the event date T. Denote by*

$$MP_t = \begin{cases} \{i_1\}, & \text{if } S_{i_1} \leq t < S_{i_2}, \\ \{i_1, i_2\}, & \text{if } S_{i_2} \leq t < T, \\ \emptyset, & \text{otherwise,} \end{cases}$$

the time-dependent set of correlated contracts. The notation "MP" stands for model point and corresponds to a group of contracts. These contracts are aggregated through the definition of a payout as

$$Z_t = \begin{cases} Y_{i_1}, & \text{if } S_{i_1} \leq t < S_{i_2}, \\ Y_{i_1} + Y_{i_2}, & \text{if } S_{i_2} \leq t < T, \\ 0, & \text{otherwise.} \end{cases}$$

The probability distribution of Z_t *for* $t \geq S_{i_2}$ *is given by*

$$\mathbb{P}(Z_t = 0) = 1 - \theta_{i_1}, \quad \mathbb{P}(Z_t = l_{i_1}) = \theta_{i_1} - \theta_{i_2}, \quad \text{and} \quad \mathbb{P}(Z_t = l_{i_1} + l_{i_2}) = \theta_{i_2}.$$

Following up on Example 3 for $t \geq 0$, we partition our insurance portfolio $\left\{\left(S_i, T_i, Q_i, \overline{Q}_i, l_i\right)\right\}_{i=1,\ldots,N_t}$ into M_t groups $\mathrm{MP}_{1,t}, \ldots, \mathrm{MP}_{M_t,t}$ of correlated contracts. The classes are associated with independent payouts $Z_{j,t}$, $j = 1, \ldots, M_t$. The process $(M_t)_{t\geq 0}$ counts the number of model points defined up to time $t \geq 0$. The liability $(L_t)_{t\geq 0}$ is rewritten as

$$L_t = \sum_{j=1}^{M_t} Z_{j,t} \mathbb{I}_{T_j > t},$$

where T_j coincides with the common event date of the contracts in $\mathrm{MP}_{j,t}$ and

$$Z_{j,t} = \sum_{i \in \mathrm{MP}_{j,t}} \mathbb{I}_{Q^i_{T_i} > \overline{Q}_i} \cdot l_i.$$

The liability is now a sum of independent, discrete, and positive random variables. The concept of model points has been introduced in the European directive Solvency II to speed up the calculations of best estimate liabilities associated with life insurance portfolios when using cash-flow projection models. Grouping and aggregating insurance contracts has become a common practice among actuaries, and various methods have been documented in the actuarial science literature, see for instance the works of Goffard and Guerrault (2015), Blanchet-Scalliet *et al*. (2017), Kiermayer and Weiß (2020), Gweon *et al*. (2020), Gweon and Li (2023). Here, the creation of model points is a workaround for the dependency of the Bernoulli variables, which allows us to study the distribution of the liability in a tractable way as shown in Section 3.3.2.

https://doi.org/10.1017/asb.2025.10078 Published online by Cambridge University Press

3.3.2. Recursive calculation of the solvency capitals

The liability at time $t > 0$ is given by

$$L_0 = 0, \quad L_t = \sum_{j=1}^{M_t} Z_{j,t} \mathbb{I}_{T_j > t}, \quad \text{for } t \geq 0, \tag{5}$$

where T_j denotes the common event dates of the contracts in $\text{MP}_{j,t}$. The number of active model points is denoted by

$$M_t^{\text{active}} = \sum_{j=1}^{M_t} \mathbb{I}_{T_j > t} \mathbb{I}_{\min_{i \in \text{MP}_{j,t}} S_i \geq t}.$$

Provided that $M_t^{\text{active}} = n$, the payouts of $\text{MP}_1, \ldots, \text{MP}_n$, at time $t \geq 0$ are denoted by $Z_1, \ldots, Z_n$. Consequently, the distribution of the liability $L_t = \sum_{k=1}^{n} Z_k$ is represented as the sum of independent random variables. While it is possible to compute the exact distribution using combinatorial analysis, this method becomes impractical as n increases. We use instead a fast Fourier transform (FFT) procedure. After obtaining the probability distribution of L_t, we can compute its cumulative distribution function and inverse it using a bisection algorithm. However, despite its effectiveness, FFT cannot be implemented on the blockchain due to the limitations of smart contract functions, which are unable to encode complex operations. This constraint necessitates performing solvency capital calculations off-chain, requiring intervention from the smart contract manager. Furthermore, any changes in the portfolio at a later time $s \geq t$ would require recomputing the probability distribution of the liability L_s to determine the Solvency Capital Requirement (SCR_s) or Minimum Capital Requirement (MCR_s).

To address this, we use a normal approximation, justified by the generalized central limit theorem, which is applicable to independent random variables with finite variance. Let $\mathbb{E}(Z_k) = \mu_k$ and $\mathbb{V}(Z_k) = \sigma_k^2$ for $k = 1, \ldots, n$ be the mean and variance of the random variables Z_k. Define $s_n^2 = \sum_{k=1}^{n} \sigma_k^2$ as the sum of these variances.

Since the random variables $(Z_k)_{k=1,\ldots,n}$ are bounded, they satisfy Lyapunov's condition that reads as

$$\lim_{n \to \infty} \frac{1}{s_n^{2+\delta}} \sum_{k=1}^{n} \mathbb{E}\left[|Z_k - \mu_k|^{2+\delta}\right] = 0.$$

Thus, the following convergence result holds:

$$\frac{1}{s_n} \sum_{k=1}^{n} (Z_k - \mu_k) \xrightarrow{\mathcal{D}} \mathsf{Normal}(0, 1), \quad \text{as } n \to \infty,$$

where $\mathsf{Normal}(0, 1)$ represents the standard normal distribution and $\xrightarrow{\mathcal{D}}$ denotes convergence in distribution. We note that

$$L_t - \Pi_t = \sum_{k=1}^{n} \left(Z_k - \sum_{i \in \text{MP}_k} (1 + \eta_{S_i}) \cdot \theta_i \cdot l_i \right) = s_n \left(\frac{1}{s_n} \sum_{i=1}^{n} (Z_i - \mu_i) \right) - \sum_{k=1}^{n} \sum_{i \in \text{MP}_k} \eta_{S_i} \cdot \theta_i \cdot l_i.$$

We deduce the following approximations for the SCR and MCR:

$$\text{SCR}_t \approx s_n \cdot q_{\alpha_{\text{SCR}}} - \sum_{k=1}^{n} \sum_{i \in \text{MP}_k} \eta_{S_i} \cdot \theta_i \cdot l_i, \quad \text{and} \quad \text{MCR}_t \approx s_n \cdot q_{\alpha_{\text{MCR}}} - \sum_{k=1}^{n} \sum_{i \in \text{MP}_k} \eta_{S_i} \cdot \theta_i \cdot l_i, \tag{6}$$

where q_α is the quantile of order α of the standard normal distribution. The accuracy of the approximation in (6) depends on the convergence of the Central Limit Theorem (CLT) as n grows large. A common way to improve this approximation is through the Cornish–Fisher expansion, as seen in Cornish and Fisher (1938), Fisher and Cornish (1960). We limit ourselves to the third- and fourth-order expansions, which involve replacing the quantile q_α by

$$q_\alpha + \gamma_1 \frac{q_\alpha^2 - 1}{6}, \quad \text{and} \quad q_\alpha + \gamma_1 \frac{q_\alpha^2 - 1}{6} + \gamma_2 \frac{q_\alpha^3 - 3q_\alpha}{24} - \gamma_1^2 \frac{2q_\alpha^3 - 5q_\alpha}{36}, \tag{7}$$

https://doi.org/10.1017/asb.2025.10078 Published online by Cambridge University Press

Table 2. *Characteristics of the first five parametric insurance contracts.*

S	T	Q	l	station
2024-11-23	2025-01-08	5	5	MARSEILLE-MARIGNANE
2024-12-23	2025-01-08	5	5	STRASBOURG-ENTZHEIM
2024-07-06	2025-01-15	5	5	STRASBOURG-ENTZHEIM
2024-01-26	2025-01-10	5	20	STRASBOURG-ENTZHEIM
2024-01-06	2025-01-09	5	10	MARSEILLE-MARIGNANE

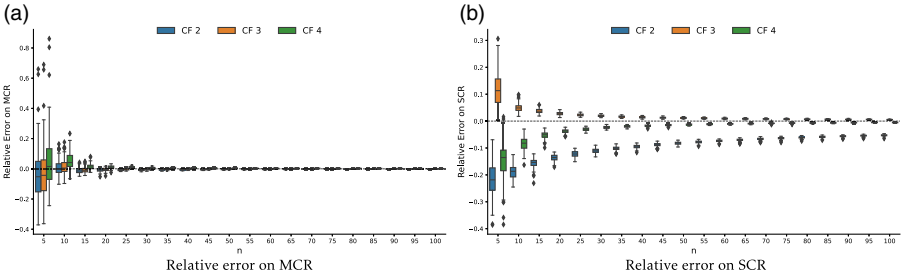


Figure 7. *Relative error on MCR and SCR depending on the number of models points in the porttfolio.*

respectively, where γ_1 and γ_2 denote the skewness and excess kurtosis of L_t. Note that the approximation in (6) is referred to as the second-order Cornish–Fisher approximation. We then decide on a threshold $\bar{n}$ such that if $n > \bar{n}$, we compute the solvency capital using the approximations in (6) or (7). Otherwise, we fall back to:

$$\mathrm{SCR}_t = \mathrm{MCR}_t = \sum_{k=1}^{N_t} l_k.$$

The determination of such a threshold can be based on a brief simulation study, as illustrated in Example 4.

Example 4. *We sample a synthetic parametric insurance portfolio* $\{(S_i, T_i, Q_i, \overline{Q}_i, l_i)\}_{i\geq 1}$ *in order to yield exactly* $n = 100$ *model points. To achieve this, we select 100 combinations of location and date. For each combination, we draw uniformly at random from* $\{1, 2, 3, \ldots, 10\}$ *a number of parametric insurance contracts to be aggregated in a model point. The threshold is fixed to* $\overline{Q}_i = 5$ *and the compensations are drawn at random as:*

$$l_i \sim \mathit{Unif}(\{5, 10, 15, 20\}), \quad \text{for } i \geq 1.$$

The first five rows of such a synthetic portfolio are provided in in Table 2.

We simulate 100 synthetic portfolios and calculate the relative errors between the exact and approximated solvency capitals for an increasing number of model points, n. The order of the quantiles are set to $q_{\alpha_{MCR}} = 0.85$ *and* $q_{\alpha_{SCR}} = 0.995$. *The results of these simulations are displayed in Figure 7.*

The normal approximation becomes acceptable for the MCR relatively quickly, but the relative error for the SCR converges more slowly. Overall, adjusting for skewness is beneficial for estimating both the MCR and SCR. The level of error falls below 5% when $n \geq 15$ *and drops below 1% for* $n \geq 30$. *If we consider a 5% error level to be acceptable, then the third-order Cornish–Fisher approximation could be used as soon as the number of model points reaches 15.*

https://doi.org/10.1017/asb.2025.10078 Published online by Cambridge University Press

Remark 3.7. It is common practice among insurers to limit the underwriting of highly correlated risks. This can be easily implemented by preventing underwriting if the contract is aggregated within a model point that already contains too many contracts. Note that we implicitly do this in Example 4, as the number of correlated risks is limited to 10. Additionally, the condition could be based on the maximum possible loss associated with the model point.

The conclusions and resulting recommendations of Example 4, are valid for the specific case under consideration. A similar study could be conducted to adapt the threshold for the number of model points and guide the selection of an appropriate order for the Cornish–Fisher approximation for another situation.

4. Smart contract description

A smart contract is a system whose state evolves according to the actions of blockchain users. The roles of the users of our smart contract are outlined in Section 4.1. The stochastic processes used to monitor the state of the system are defined in Section 4.2. We introduce the methods that allow users to interact with the smart contract in Section 4.3.

4.1. Role distribution

The participants in our blockchain insurance scheme include the smart contract owner, the investors (also referred to as surplus providers), and the policyholders. The processes of underwriting and claim assessment are fully automated within the smart contract protocol. The role of our investors is analogous to that of the reinsurers in Cousaert *et al*. (2022).

4.1.1. Smart contract owner

The smart contract owner is responsible for designing the insurance protocol by writing the Solidity code for the smart contract. While deploying the smart contract on the Ethereum blockchain, the owner initializes three parameters: the premium loading η, the risk levels α_{SCR} and α_{MCR} associated with the solvency capitals, as defined in Section 3, and the lower bound m for the number of active model points to use the normal approximation of Section 3.3 to calculate the solvency capitals. Additionally, the contract owner acts as the claim assessor or data oracle by inputting the observed quantity at the time of the event–in this case, the precipitation height. In the blockchain ecosystem, an oracle refers to an off-chain data provider that a smart contract can query. This feature is not implemented in our current solution, it would be beneficial to do so in order to further automate our workflow.

4.1.2. Surplus providers

SP, also known as investors, contribute financially to the smart contract to constitute the solvency capital. These contributions are denominated in ETH, the native cryptocurrency of the Ethereum blockchain. In return for their investment, investors receive tokens that represent their proportional share of the surplus. SPs have the option to withdraw a portion of their funds by burning these tokens in exchange for an equivalent amount of ETH, with the exchange rate defined in the Section 4.2. Withdrawals are permitted provided that the funds remaining in the smart contract do not fall below a specified threshold, known as the SCR.

4.1.3. Policyholders

Policyholders seek financial protection against adverse events through our smart contract. Upon the occurrence of such events, the smart contract pays a lump-sum compensation in ETH. To obtain this coverage, policyholders must pay a premium, also in ETH, to the smart contract.

https://doi.org/10.1017/asb.2025.10078 Published online by Cambridge University Press

An insurance agreement is only concluded if the SCR, after accounting for the new risk, does not exceed the funds currently locked in the smart contract. Should the settlement of a claim lead to the smart contract's funds falling below the MCR, the system enters a state of bankruptcy. In such instances, priority is given to reimbursing policyholders with the remaining funds. Any funds that remain after all policyholder reimbursements are distributed to investors, proportional to their token holdings.

4.2. State variables

The state variables of the smart contract include the stochastic processes defined in Section 3, supplemented by additional processes defined on the filtered probability space $(\Omega, \mathcal{F}, (\mathcal{F})_{t\geq 0}, \mathbb{P})$.

Upon deployment on the Ethereum blockchain, a smart contract is assigned an Ethereum address, enabling it to send and receive ETH. The balance $(B_t)_{t\geq 0}$ increases when investors contribute funds or when premiums are paid for underwritten policies. Conversely, the balance decreases upon investor withdrawals or compensation payments to policyholders. Let $(\tau_k^+)_{k\geq 1}$ and $(\tau_k^-)_{k\geq 1}$ denote the times of investor deposits and withdrawals, respectively, with associated amounts $(x_k^+)_{k\geq 1}$ for deposits and $(x_k^-)_{k\geq 1}$ for withdrawals. The balance can be expressed as

$$B_0 = 0, \quad \text{and} \quad B_t = \sum_{k\geq 1} \mathbb{I}_{\tau_k^+ \leq t} x_k^+ - \sum_{k\geq 1} \mathbb{I}_{\tau_k^- \leq t} x_k^- + \sum_{i\geq 1} \mathbb{I}_{S_i \leq t} \pi_i - \sum_{i\geq 1} \mathbb{I}_{T_i \leq t} \mathbb{I}_{Q_{T_i} > \bar{Q}_i} l_i, \quad \text{for } t \geq 0.$$

We define $(X_t)_{t\geq 0}$ as the surplus of the smart contract, representing the funds available to underwrite new insurance policies. Unlike the balance, the surplus does not increase immediately upon underwriting a new contract. Instead, the premium is added at the policy's resolution, when it is considered "earned." Thus, the surplus is given by

$$X_0 = 0, \quad \text{and} \quad X_t = \sum_{k\geq 1} \mathbb{I}_{\tau_k^+ \leq t} x_k^+ - \sum_{k\geq 1} \mathbb{I}_{\tau_k^- \leq t} x_k^- + \sum_{i\geq 1} \mathbb{I}_{T_i \leq t} \pi_i - \sum_{i\geq 1} \mathbb{I}_{T_i \leq t} \mathbb{I}_{Q_{T_i} > \bar{Q}_i} l_i, \quad \text{for } t \geq 0.$$

Note that $B_t \geq X_t$ for every $t \geq 0$. The surplus $(X_t)_{t\geq 0}$ is compared against SCR_t and MCR_t. If $X_t \leq \text{SCR}_t$, no new policies may be underwritten, and investors are prohibited from withdrawing funds. If $X_t \leq \text{MCR}_t$, the smart contract ceases operations, and the remaining balance is redistributed–first to policyholders and then to investors, as detailed in Section 4.3.4. Active insurance policies at the time of termination are cancelled.

Let $(A_t)_{t\geq 0}$ denote the total number of distinct Ethereum addresses that have transferred funds to the smart contract and received protocol tokens. Let

$$Y_t = \left(Y_t^{(1)} \ \ldots \ Y_t^{(A_t)}\right),$$

be a vector representing the token holdings of the smart contract participants. The total supply of tokens is given by $\overline{Y}_t = \sum_{i=1}^{A_t} Y_t^{(i)}$. The exchange rate of tokens against ETH is modeled by the stochastic process $(r_t)_{t\geq 0}$, defined as

$$r_0 = 1, \quad r_t = \frac{X_t}{\overline{Y}_t}, \quad \text{for } t \geq 0.$$

The exchange rate must be updated whenever X_t or $\overline{Y}_t$ changes. Lastly, denote by

$$\Lambda_0 = 0, \quad \Lambda_t = \sum_{i=1}^{N_t} l_i \mathbb{I}_{S_i \leq t, T_i > t}, \quad \text{for } t \geq 0,$$

the sum of all the potential compensations to be paid.

4.3. Method of the smart contract

The state of a smart contract changes according to the actions of blockchain users. In our parametric insurance example, these users include the contract owner, SP, and policyholders (see Section 4.1).

https://doi.org/10.1017/asb.2025.10078 Published online by Cambridge University Press

Participants interact with the smart contract through specific functions, known as methods, which modify the system's state. The remainder of this section describes each possible action and its impact on the stochastic processes introduced thus far.

4.3.1. Funding the contract

Assume that an investor transfers x ETH to the contract at time $t > 0$. As a result, both the balance and the surplus of the contract increase as follows:

$$B_{t+h} = B_t + x, \quad X_{t+h} = X_t + x,$$

where $h > 0$ represents an infinitesimal time lapse.

If the investor is new, the count of distinct addresses increases, such that $A_{t+h} = A_t + 1$; otherwise, $A_{t+h} = A_t$. The j-th component of the vector $(Y_t)_{t\geq 0}$, where $j \in \{1, \dots, A_{t+h}\}$, is updated as

$$Y^{(j)}_{t+h} = Y^{(j)}_t + y,$$

where $y = x/r_t$.

4.3.2. Underwriting a parametric insurance policy

A customer wishes to purchase an insurance contract $(S, T, Q, \overline{Q}, l)$ at time $t = S$ such that $S < S^*$. The premium for such a parametric insurance contract is given by

$$\pi = (1 + \eta_t) \cdot l \cdot \theta,$$

where $\theta = \mathbb{P}(Q_T > \overline{Q} | \mathcal{F}_S)$. The contract can only be underwritten if the surplus of the smart contract is sufficient to cover the resulting increase in the SCR.

The liability, the cumulative premium collected, and the number of active model points would then be updated to

$$\widetilde{\Lambda}_t = \Lambda_t + l, \quad \widetilde{L}_t = L_t + l \cdot \mathbb{I}_{Q_T > \bar{Q}}, \quad \widetilde{\Pi}_{t+h} = \Pi_t + \pi,$$

and

$$\widetilde{M}^{\text{active}}_t = \begin{cases} M^{\text{active}}_t, & \text{if the new contract is grouped into an active model point,} \\ M^{\text{active}}_t + 1, & \text{otherwise.} \end{cases}$$

We calculate the updated solvency capital requirement (SCR):

$$\widetilde{\text{SCR}}_t = \begin{cases} \text{Quantile}(\widetilde{L}_t - \widetilde{\Pi}_t \,; \alpha_{\text{SCR}}), & \text{if } \widetilde{M}^{\text{active}}_t \geq m, \\ \Lambda_t, & \text{otherwise.} \end{cases}$$

and underwrite the contract only if:

$$X_t \geq \widetilde{\text{SCR}}_t. \tag{8}$$

If condition (8) holds, the number of contracts, liability, premium collected, and number of active model points are updated as follows:

$$N_{t+h} = N_t + 1, \quad \Lambda_{t+h} = \Lambda_t + l, \quad L_{t+h} = L_t + l \cdot \mathbb{I}_{Q_T > \bar{Q}}, \quad \Pi_{t+h} = \Pi_t + (1 + \eta_t) \cdot l \cdot \theta, \text{ and } M^{\text{active}}_{t+h} = \widetilde{M}^{\text{active}}_t$$

The SCR and MCR are also updated as

$$\text{SCR}_{t+h} = \begin{cases} \text{Quantile}(L_{t+h} - \Pi_{t+h} \,; \alpha_{\text{SCR}}), & \text{if } M^{\text{active}}_{t+h} \geq m, \\ \Lambda_t, & \text{otherwise,} \end{cases}$$

https://doi.org/10.1017/asb.2025.10078 Published online by Cambridge University Press

and

$$\mathrm{MCR}_{t+h} = \begin{cases} \mathrm{Quantile}(L_{t+h} - \Pi_{t+h}\,; \alpha_{\mathrm{MCR}}), & \text{if } M_{t+h}^{\text{active}} \geq m, \\ \Lambda_t, & \text{otherwise.} \end{cases}$$

We add a component:

$$\xi_{t+h}^{(N_{t+h})} = 0,$$

to the vector Ξ_t that gathers the statuses of the insurance policies. The balance of the contract is updated as

$$B_{t+h} = B_t + (1 + \eta_t) \cdot \theta \cdot l.$$

4.3.3. Withdrawing from the smart contract and burning tokens

An investor $j \in \{1, \ldots, A_t\}$ may withdraw some of her funds by burning protocol tokens at time $t > 0$. This action is subject to the conditions that the participant holds enough protocol tokens $Y_t^{(j)}$, the smart contract has a sufficient balance B_t, and the surplus X_t remains adequate.

Assume that participant j wishes to burn y tokens at time $t > 0$. Upon burning, the participant should receive $x = y \cdot r_t$ ETH. The transaction is allowed if the following conditions are met:

$$y \leq Y_t^{(j)}, \quad x \leq B_t, \quad \text{and} \quad x < X_t - \mathrm{SCR}_t. \tag{9}$$

Provided that the conditions in (9) are satisfied, we update the contract balance and the surplus as follows:

$$B_{t+h} = B_t - x, \quad \text{and} \quad X_{t+h} = X_t - x.$$

We further update the token balance of investor j with:

$$Y_{t+h}^{(j)} = Y_t^{(j)} - y.$$

Remark 4.1. Since the events that trigger compensation are predictable in the short term, it is essential to impose timing constraints on investors' fund withdrawals. One practical solution would be to require investors to submit withdrawal orders, which the smart contract owner would validate before processing. Under this approach, investors would indicate their intention to withdraw funds at a future date and specify the desired amount in advance. Note that this feature has not been implemented in our current project.

4.3.4. Contract resolution and claim settlement

At time $t = T_i$, the contract $(S_i, T_i, Q_i, \overline{Q}_i, l_i)$, for some $i = 1, \ldots, N_t$, is settled as it becomes known whether the event $Q_{T_i} > \overline{Q}_i$ has occurred. The liabilities and the premium collected are updated as follows:

$$\widetilde{\Lambda}_t = \Lambda_t - l_i, \widetilde{L}_t = L_t - l_i, \quad \widetilde{\Pi}_t = \Pi_t - (1 + \eta_{S_i}) \cdot l_i \cdot \theta_i.$$

The balance of the contract remains unchanged, $\widetilde{B}_t = B_t$, while the surplus is updated as

$$\widetilde{X}_t = \begin{cases} X_t + (1 + \eta_{S_i}) \cdot \theta_i \cdot l_i, & \text{if } Q_{T_i} \leq \overline{Q}_i, \\ X_t + (1 + \eta_{S_i}) \cdot \theta_i \cdot l_i - l_i, & \text{otherwise.} \end{cases}$$

https://doi.org/10.1017/asb.2025.10078 Published online by Cambridge University Press

Recall that S_i is the time at which contract i was underwritten. The number of active model points is updated as follows:

$$\widetilde{M}_t^{\text{active}} = \begin{cases} M_t^{\text{active}} - 1, & \text{if the contract was the last contract of a given model point,} \\ M_t^{\text{active}}, & \text{otherwise.} \end{cases}$$

The SCR and MCR are also updated as

$$\widetilde{\text{SCR}}_t = \begin{cases} \text{Quantile}(\widetilde{L}_t - \widetilde{\Pi}_t\,; \alpha_{\text{SCR}}), & \text{if } \widetilde{M}_t^{\text{active}} \geq m, \\ \widetilde{\Lambda}_t, & \text{otherwise,} \end{cases}$$

and

$$\widetilde{\text{MCR}}_t = \begin{cases} \text{Quantile}(\widetilde{L}_t - \widetilde{\Pi}_t\,; \alpha_{\text{MCR}}), & \text{if } \widetilde{M}_t^{\text{active}} \geq m, \\ \widetilde{\Lambda}_t, & \text{otherwise.} \end{cases}$$

At this stage, we distinguish two cases. First assume that $\widetilde{X}_t > \widetilde{\text{MCR}}_t$ then all the processes are updated for good with

$$B_{t+h} = \widetilde{B}_t,\ X_{t+x} = \widetilde{X}_t,\ \Lambda_{t+h} = \widetilde{\Lambda}_t,\ L_{t+h} = \widetilde{L}_t,\ \Pi_{t+x} = \widetilde{\Pi}_t,\ \text{SCR}_{t+h} = \widetilde{SCR}_t,\ \text{MCR}_{t+x} = \widetilde{\text{MCR}}_t, \text{ and}$$

$$M_{t+h}^{active} = \widetilde{M}_t^{\text{active}}.$$

The premium for contract i is now considered "earned." The exchange rate is updated as

$$r_{t+h} = \frac{X_{t+h}}{\overline{Y}_{t+h}},$$

where it is noted that the total supply of protocol tokens remains unchanged between t and $t+h$.

Now suppose that the surplus downcrosses the MCR as $\widetilde{X}_t \leq \widetilde{\text{MCR}}_t$. If this occurs, the smart contract is reset. The outstanding balance is first used to reimburse the policyholders of active contracts whose status become "cancelled". The total amount is therefore given by

$$x = \sum_{i=1}^{N_t} \mathbb{I}_{\xi_t^{(i)}=0} \left(1 + \eta_{S_i}\right) \cdot \theta_i \cdot l_i.$$

If $\widetilde{X}_t > x$, the premiums are fully refunded, and the remaining wealth, $\widetilde{X}_t - x$, is distributed to the investors according to their token holdings, using the updated exchange rate:

$$\widetilde{r}_t = \frac{\widetilde{X}_t - x}{\bar{Y}_t}.$$

The smart contract transfers:

$$x_j = \widetilde{r}_t \cdot Y_t^{(j)},$$

to all investors $j = 1, \ldots, A_t$.

If $\widetilde{X}_t \leq x$, the policyholders are reimbursed pro rata based on the premiums they paid. Specifically, we define weights for the policyholders with active contracts as

$$w_i = \frac{\mathbb{I}_{\xi_t^{(i)}=0} \left(1 + \eta_{S_i}\right) \cdot \theta_i \cdot l_i}{\sum_{i=1}^{N_t} \mathbb{I}_{\xi_t^{(i)}=0} \left(1 + \eta_{S_i}\right) \cdot \theta_i \cdot l_i},$$

and transfer:

$$x_i = w_i \cdot \widetilde{X}_t,$$

to the policyholder associated to policy i for $i = 1, \ldots, N_t$.

https://doi.org/10.1017/asb.2025.10078 Published online by Cambridge University Press

After completing these transfers, the smart contract variables are reset to their initial state with:

$$B_{t+h}=0,\quad X_{t+h}=0,\quad \Lambda_{t+h}=0,\quad L_{t+h}=0,\quad \Pi_{t+h}=0,\quad \mathrm{SCR}_{t+h}=0,\quad \mathrm{MCR}_{t+h}=0,\quad r_{t+h}=1,$$

and $M_{t+h}^{active}=0$.

The number of contracts and the number of token holders remain unchanged:

$$N_{t+h}=N_t,\quad A_{t+h}=A_t.$$

However, all token holdings are reset to zero:

$$Y_{t+h}^{(j)}=0,\quad \text{for all } j=1,\ldots,A_{t+h}.$$

The practical implementation of the smart contract using the programming language Solidity is documented in the online accompaniment of this paper.[1]

5. Interaction with the smart contract

During the development phase, a smart contract is not immediately deployed on the Ethereum mainnet. Instead, developers use testnets to simulate the behavior of the real blockchain, ensuring that the smart contract operates as intended. For our project, we deployed the smart contract on the Ethereum testnet known as Sepolia. Sepolia includes a block explorer called Etherscan,[2] which provides an API for data retrieval. Our contract, named `InsuranceLogic`, is located at the address:

0xbe035cf1367c45A0C9517969F5ABDd3abF743ae7.

It can be viewed on Etherscan.[3] The complexity of the insurance mechanism necessitated the deployment of two additional smart contracts. `PricingLogic` located at the address

0xb5a208E3d8c74464d7a70C9B7219880097c6DE81,

calculates the premium and `ModelPointsLogic` located at

0xA1Ad49F8a7e8781A0488029eF7AC471901131Fce,

computes the solvency capitals. The Solidity code for these smart contracts is thoroughly discussed in the online supplementary materials that accompany this paper.[4]

The deployment of the contract was carried out using Remix IDE,[5] which is an open-source, web-based integrated development environment specifically designed for writing, testing, and deploying smart contracts on the Ethereum blockchain. We first compile the Solidity code before deploying it to the blockchain. Once deployed, the contract is associated with an Ethereum address, enabling interaction through Remix IDE by calling functions and checking the values of the state variables.

Each action on the testnet incurs a cost denominated in Sepolia testnet tokens, which act as a substitute for ETH. Payments are made through wallets. We use MetaMask,[6] a popular cryptocurrency wallet and gateway to blockchain applications. MetaMask serves as a browser extension or mobile app that enables users to interact with Ethereum-based decentralized applications (dApps) directly from their web browser or smartphone.

To test the smart contract, three Ethereum accounts were created to represent the roles of the smart contract owner, surplus provider, and policyholder. The addresses of these accounts (plus that of the smart contract) are provided in Table 3.

[1] https://github.com/LaGauffre/smart_parametric_insurance/blob/main/latex/sup_material_blockchain_parametric_insurance.pdf.
[2] See https://sepolia.etherscan.io/.
[3] see https://sepolia.etherscan.io/address/0xbe035cf1367c45A0C9517969F5ABDd3abF743ae7.
[4] https://github.com/LaGauffre/smart_parametric_insurance/.
[5] See https://remix.ethereum.org/.
[6] See https://metamask.io/.

https://doi.org/10.1017/asb.2025.10078 Published online by Cambridge University Press

Table 3. *Ethereum addresses of the smart contract users.*

ETH Address	Role
`0xE8e79B8B8c0481fa33a8E0fcA902ad5754BfE1C3`	*owner*
`0x2CF8ed1664616483c12Ef3113f6F62E68f1a810A`	*surplus provider*
`0xd34a37613A382bA503f1599F514C9788dF3659C4`	*policyholder*
`0xbe035cf1367c45A0C9517969F5ABDd3abF743ae7`	*smart contract*

Our smart contract allows for the underwriting of parametric insurance contracts to compensate policyholders if the height of precipitation exceeds a threshold $\bar{Q} = 5$ on a particular day in the year 2025 in either Marseille or Strasbourg. The Cornish–Fisher approximation is employed as soon as we reach 5 model points. The day of the year is represented as an integer between 1 and 365. We designed the following scenario to illustrate the functions available in the smart contract described in Section 4.

1. **Deployment**: The *Owner* initiates the process by deploying three key contracts: `PricingLogic`, `ModelPointsLogic`, and `InsuranceLogic`. The initial parameters are set as follows:

$$\eta = 0.1, \quad \alpha_{\text{SCR}} = 0.995, \quad \text{and} \quad \alpha_{\text{MCR}} = 0.85.$$

2. **Initial Funding**: A *Surplus Provider* kicks things off by sending 0.1 ETH to the smart contract, providing the initial capital needed to underwrite policies.
3. **Policy Underwriting**: *Policyholders* begin underwriting various insurance policies to protect against precipitation exceeding 5 units in Marseille or Strasbourg on specific days of the year. Here are the details of the policies they secure:

   - Policy #1: Covers Marseille on day 60, with a liability of 0.01 ETH.
   - Policy #2: Covers Strasbourg on day 60, with a liability of 0.01 ETH.
   - Policy #3: Covers Marseille on day 45, with a higher liability of 0.02 ETH.
   - Policy #4: Covers Strasbourg on day 80, with a liability of 0.005 ETH.
   - Policy #5: Covers Strasbourg on day 102, with a liability of 0.015 ETH.
   - Policy #6: Covers Strasbourg on day 206, with a liability of 0.01 ETH.
   - Policy #7: Similar to Policy #6, covering Strasbourg on day 206, with a liability of 0.01 ETH.
   - Policy #8: Covers Marseille on day 300, with a liability of 0.015 ETH.
   - Policy #9: Covers Marseille on day 282, with a liability of 0.005 ETH.
   - Policy #10: Covers Marseille on day 180, with a liability of 0.02 ETH.

4. **Settling one Contract and withdrawing of the investor**: The *Owner* starts settling the contracts based on the observed precipitation:

   - Contract #3 is settled with precipitation in Marseille on day 45 not exceeding the threshold.
   - A *Surplus Provider* decides to burn 0.035 tokens, adjusting their investment.

5. **Further Contract Settlements**: The *Owner* continues settling more contracts:

   - Contract #1 is settled with precipitation in Marseille on day 60 exceeding the threshold.
   - Contract #2 is settled with precipitation in Strasbourg on day 60 exceeding the threshold.
   - Contract #4 is settled with precipitation in Strasbourg on day 80 exceeding the threshold.
   - Contract #5 is settled with precipitation in Strasbourg on day 102 not exceeding the threshold.

https://doi.org/10.1017/asb.2025.10078 Published online by Cambridge University Press

Table 4. *Description of the transactions data frame associated with the contract.*

Variable name	Type	Example value	Description
blockNumber	object	8518222	The height of the block in the blockchain
timeStamp	object	1749558720	The date and time when the transaction was created or mined
hash	object	0xe4ca. . .	The unique hash value identifying the transaction
nonce	object	101	A number used to ensure that each transaction can only be processed once
blockHash	object	0x0248. . .	The unique hash value identifying the block that contains the transaction
transactionIndex	object	31	The position of the transaction within the block
from	object	0xe8e7. . .	The address of the sender who initiated the transaction
to	object	0xbe03. . .	The address of the recipient of the transaction
value	object	0	The amount of ETH sent with the transaction
gas	object	3958691	The maximum amount of gas allocated for the transaction
gasPrice	object	1501121993	The price of each unit of gas, specified in ETH
isError	object	0	Indicates whether an error occurred during the transaction
txreceipt_status	object	1	Indicates the status of the transaction receipt
input	object	0x6080. . .	The data field containing information necessary for the transaction
contractAddress	object	0xbe03. . .	The address of the contract created as a result of the transaction
cumulativeGasUsed	object	6358722	The total amount of gas used in the block up to and including this transaction
gasUsed	object	3926362	The amount of gas actually used by this specific transaction
confirmations	object	652	The number of blocks mined after the block containing this transaction
methodId	object	0x6080. . .	The identifier for the method being called in a smart contract
functionName	object		The name of the function called in a smart contract

The final event triggers the smart contract to file for bankruptcy, as the surplus X falls below the MCR threshold. Consequently, the premiums for policies #6, #7, #8, #9, and #10 are refunded to the policyholders. The remaining surplus is then returned to the surplus provider.

We do not suggest that the participants in this scenario acted strategically–in fact, quite the opposite. By withdrawing funds early, the surplus provider jeopardized the solvency of the smart contract. The scenario where multiple insurance contracts result in compensation is both highly unlucky and statistically improbable. Additionally, the reduction in the number of model points below the minimum required for the Cornish–Fisher approximation led to a drastic increase in the MCR, which ultimately exceeded the surplus. These choices were intentional, designed to illustrate the consequences of the smart contract going bankrupt and undergoing a reset.

Deploying the contract results in a "contract creation" transaction on the blockchain. Each subsequent function call also generates a transaction, which is recorded on the blockchain. These transactions can be retrieved and organized into a data frame, the structure of which is detailed in Table 4.

https://doi.org/10.1017/asb.2025.10078 Published online by Cambridge University Press

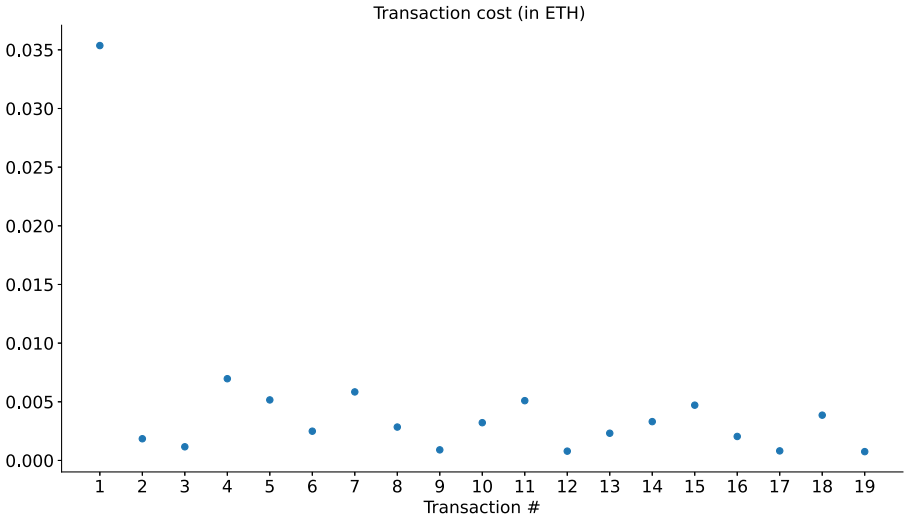


Figure 8. *Transaction costs in ETH.*

As mentioned earlier, each transaction incurs a cost to the sender, calculated as

$$\text{gasPrice} \times \text{gasUsed}.$$

The value of a unit of gas (`gasPrice`) fluctuates over time, as it depends on the level of activity on the Ethereum network. High network congestion leads to higher gas prices. Another factor influencing transaction costs is the amount of data being recorded, which is reflected in the `input` field of the transaction details. The "contract creation" transaction is by far the most expensive, with a cost of approximately ETH0.035, due to the significant amount of data required to deploy the smart contract. In contrast, subsequent transactions typically cost between ETH0.002 and ETH0.008, as illustrated in Figure 8.

In addition to the list of all transactions, we can retrieve the log of events emitted by the smart contract. A common practice is to associate a specific event with each function. Function calls impact the state of the system, and events provide a mechanism to monitor state transitions effectively. However, care must be taken when defining events, as events that contain a large amount of information will require more data storage and, consequently, increase transaction costs. The definition of the events is provided Table 5.

The events used in this study are defined in the Solidity code, which is included in the online supplementary materials accompanying this article. Using Python, we retrieve the event logs to track participant balances over time. Figure 9 illustrates the balances, in ETH and before transaction fees, of the Ethereum addresses associated with the *Owner*, the *Surplus Provider*, the *Policyholder*, and the *Smart Contract* after each function call.

We observe that the scenario was particularly beneficial to the *Policyholder*, who received three compensation payments and a refund for five policies. The event logs also enable us to visualize the trajectories of the various processes defined in Sections 3 and 4. Specifically, Figure 10 displays the trajectories of X (surplus), B (balance), SCR, and MCR as they evolved throughout the scenario.

Note the difference between the surplus and the balance of the smart contract: the balance of the smart contract increases at each "Underwrite" event, while the surplus remains constant. Until contract #5 is underwritten, the SCR and MCR are exactly equal to the sum of all potential compensations to be paid. Once contract #5 is underwritten, the number of model points reaches 5, and the solvency capitals are calculated via the Cornish–Fisher approximation, causing a drop in the value of the solvency capitals. After the settlement of contract #3, the exchange rate of protocol tokens against ETH appreciates, which is why the surplus provider decides to cash out part of her investment. The compensations paid after

https://doi.org/10.1017/asb.2025.10078 Published online by Cambridge University Press

Table 5. *Description of events generated by the smart contract.*

Event name	Short name	Logged information	Type	Example value	Description
Parameters Updated	Update	`newEta`	int	1000	premium loading
		`newQAlphaSCR`	int	25,758	Quantile of the normal distribution
		`newQAlphaMCR`	int	10,364	–
Fund	Fund	from	hex	0*x*123 . . .	address of the sender
		x	int	10,000	Amount of ETH
		y	int	10,000	Amount of token
Burn	Burn	from	hex	0*x*123 . . .	address of the sender
		x	int	10,000	Amount of ETH
		y	int	10,000	Amount of token
Insurance Under written	Underwrite	contractId	int	1	Index of the policy
		customer_address	hex	0*x*456 . . .	Ethereum address
		T	int	80	Day of the event with year 2025
		station	string	STRASBOURG-ENTZHEIM	location of the event
		l	int	1,000,000	Compensation if the event occurs
		cp	int	1,000,000	Commercial premium
		status	int	0	Status of the insurance policy
		SCR	int	100,000	Updated Solvency Capital Requirement
		MCR	int	100,000	Updated Minimum Capital Requirement
ClaimSettled	Settle	contractId	int	1	Index of the policy
		customer_address	hex	0*x*456 . . .	Ethereum address
		payoutTransferred	boolean	`True`	Indicates if a compensation has been paid
		SCR	int	100,000	Updated Solvency Capital Requirement
		MCR	int	100,000	Updated Minimum Capital Requirement

the settlements of contracts #1, #2, and #4, combined with the fact that the settlement of contract #5 reduces the number of model points below 5, cause the smart contract to reset as the surplus becomes smaller than the MCR. The status of all active policies changes to "cancelled." Premiums are refunded to the policyholders, and investors receive any remaining balance from the smart contract.

Remark 5.1. The drop in value of both the MCR and the SCR after including contract #5 might seem surprising at first glance. Under Solvency II regulations, there is an absolute floor on the MCR value, set at least to 2.7 million euros for non-life insurance companies. We have proposed an alternative method for defining this floor, based on the sum of exposures when the number of independent risks is too small (fewer than 5 in Figure 10 for illustration purposes, but typically fewer than hundreds or thousands in real-life scenarios). In practice, a decrease in solvency capital can occur during the transition from the

https://doi.org/10.1017/asb.2025.10078 Published online by Cambridge University Press

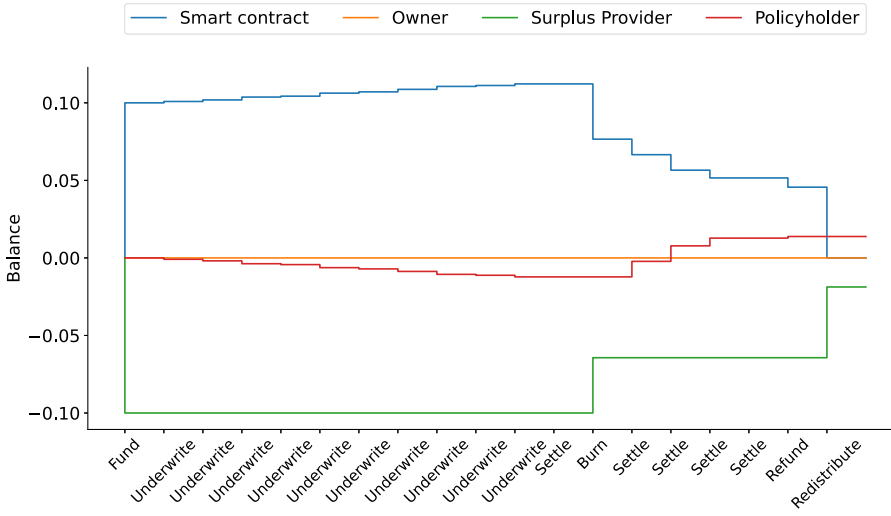


Figure 9. *Balance in ETH of the Ethereum address of the Owner, the Surplus Provider, the policyholder, and the Smart Contract.*

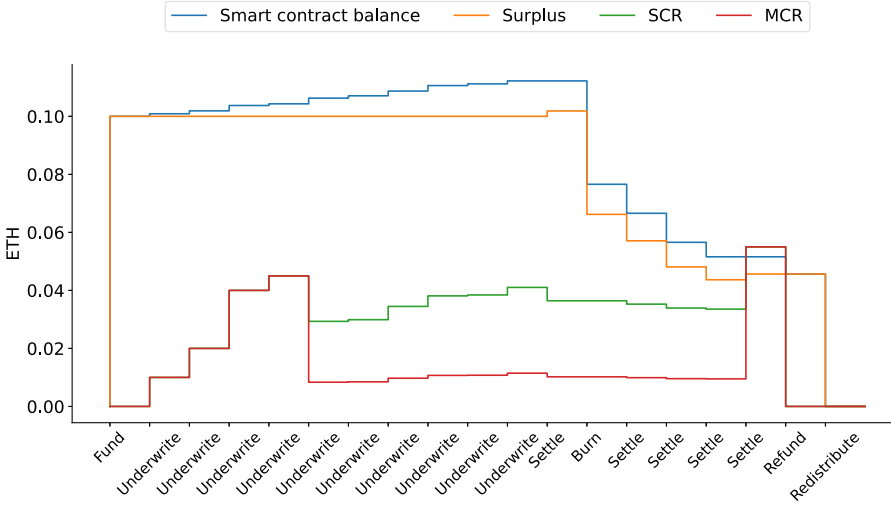


Figure 10. *Trajectories of X, B, SCR, and MCR.*

standard formula to internal models, which somewhat corresponds to adopting normal approximations in our example.

The Solidity code for the smart contract and the Python scripts for analyzing blockchain data are freely available in the smart_parametric_insurance GitHub repository.

6. Limitations and perspectives

This article proposes a framework for a smart contract suited to parametric insurance and demonstrates the feasibility of its implementation on a blockchain. Additionally, it makes this open-source prototype

https://doi.org/10.1017/asb.2025.10078 Published online by Cambridge University Press

available to other researchers, facilitating further exploration of its limitations and potential enhancements. In this section, we outline the main limitations of this initial approach and suggest possible solutions.

The underlying risks are considered stationary. However, in practice, many risks suitable for smart contracts with parametric triggers are influenced by regime changes, which can lead to event clustering or increased claim frequency. To mitigate adverse selection, a critical first step in risk management is introducing a waiting period before the guarantee becomes active. This measure helps prevent policyholders from exploiting arbitrage opportunities when they are aware of worsening risks. For instance, in the case of flight disruptions, events like strikes, conflicts, or volcanic eruptions can make delays almost certain in the short term. Similarly, for weather-related risks, extreme event warnings may be issued days in advance. It is essential to ensure that policyholders cannot exploit the smart contract under such circumstances while paying the same premium as usual.

In some cases, a waiting period may not be sufficient to mitigate risk, as underlying risks can be anticipated several months in advance. For example, the El Niño/La Niña cycle, which is partially predictable months ahead, increases the likelihood of certain extreme weather events while reducing others. Moreover, long-term factors such as climate change and sectoral inflation can exacerbate risks over time. If competitors actively adjust their premiums to reflect these changes while the smart contract does not, adverse selection becomes a likely outcome. Finding a way to update premiums dynamically is essential. This could be achieved either automatically – by scraping new data from competitors, insurance federations, or other relevant sources – or through decisions made by a "board" of customers and investors, who could leverage voting rights granted via tokens. A governance system where token-holding investors form a board with voting rights aligns with the concept of decentralized organizations, which are extensively studied in management science and have naturally emerged within blockchain applications. However, classical issues such as over- or under-reactions stemming from individual risk perceptions and cognitive biases could be amplified, especially if voter participation is low or voting power is concentrated due to coalitions. These challenges are not uncommon, as low or uneven participation rates have frequently been observed in various decentralized networks, we refer the reader to the work of Messias *et al*. (2023) for a comprehensive discussion.

Our smart contract is exposed to the risk of a "mass investor lapse," where investors may collectively withdraw their tokens. Stronger incentive mechanisms could be implemented to encourage investors to retain their tokens, particularly when the contract's balance approaches the risk capital threshold. One potential approach is to introduce a coin age principle, where tokens held for longer periods gain additional value. This concept has already been explored in the implementation of the cryptocurrency PeerCoin.[7] A traditional approach to managing the surplus of an insurance company is to implement a dividend policy that rewards investors. This can be achieved using classical optimal dividend strategies. Barrier or band strategies, for instance, have been shown to be optimal across various modeling frameworks (see the survey by Avanzi 2009). In our framework, we operate within a discrete-time risk model with capital injections and withdrawals. This approach is consistent with the structure of our system, where wealth is only inspected at discrete points–namely, during settlements or when an investor funds or withdraws from the contract.

In this paper, we have not addressed regulatory risks. In many countries, a smart contract like the one proposed would need to comply with insurance regulations. For instance, in the European Union, risk governance would need to be developed to align with the requirements of Pillar 2 of Solvency II, which presents significant challenges. Additionally, adhering to anti-money laundering (AML) regulations may necessitate incorporating a KYC (know your customer) module. While implementing such a module can be particularly challenging in a blockchain-based environment, some cryptocurrency wallet startups have successfully tackled this issue, providing potential avenues for further development.

[7] See https://www.peercoin.net/.

https://doi.org/10.1017/asb.2025.10078 Published online by Cambridge University Press

In our approach, surplus management is handled through proxies referred to as the MCR and SCR. If such a smart contract were to be classified as an insurance product, it would need to comply with the absolute minimum MCR threshold prescribed by Solvency II in Europe.

Exchange rate risk is a significant concern for potential customers, particularly as compensations are paid in ETH, a cryptocurrency subject to high volatility against fiat currencies. To mitigate this issue, one potential solution is to link our contract with another smart contract designed to exchange ETH for a stablecoin. This approach leverages the interoperability of blockchain applications, a key advantage of decentralized finance over traditional financial systems.

If the contract is not classified as an insurance product, it could lead to tax and regulatory implications. To ensure recognition as insurance, it is crucial to keep basis risk sufficiently low. Effective basis risk management not only protects policyholders but also mitigates the risk of the contract being reclassified as gambling. In some countries, policyholders may be required to provide a brief proof of loss to validate claims. While we assume that the occurrence of a compensation-triggering event inherently results in a loss for the policyholder, real-world scenarios might necessitate additional efforts from either the policyholder or the smart contract's governance system to address such requirements. Basis risk management is a central problem when dealing with parametric insurance solution and as such under investigation in many recent research work like that of Niakh *et al*. (2025).

Sustainability considerations also warrant further investigation. While blockchain technology consumes significant amounts of electricity and water, its environmental impact could be partially offset by reducing the carbon emissions associated with traditional insurance operations, such as staffing, office infrastructure, and claim management processes.

In conclusion, we have developed a modest beta version of a smart contract for parametric insurance on the Ethereum blockchain. The Solidity code for the smart contract and the Python scripts for analyzing blockchain data are freely available in the smart_parametric_insurance GitHub repository. Beyond serving as a proof of concept, this work identifies several research questions that need to be addressed to make the product fully operational. We hope that our open-source beta version will provide a foundation for other researchers to build upon, facilitating the development of enhanced features and overcoming the limitations we have outlined.

Acknowledgments. Pierre-Olivier Goffard's work is conducted as part of the Research Chair DIALOG,[8] under the auspices of the Risk Foundation, an initiative by CNP Assurances. His research is also supported by the ANR project BLOCKFI.[9] Stéphane Loisel acknowledges support from the Research Chair ACTIONS[10] (funded by BNP Paribas Cardif), the research initiative Sustainable Actuarial Science and Climatic Risks[11] (funded by Milliman Paris), and the ANR project DREAMES.[12]

References

Abdi, M.J., Raffar, N., Zulkafli, Z., Nurulhuda, K., Rehan, B.M., Muharam, F.M., Khosim, N.A. and Tangang, F. (2022) Index-based insurance and hydroclimatic risk management in agriculture: A systematic review of index selection and yield-index modelling methods. *International Journal of Disaster Risk Reduction*, 67, 102653. ISSN 2212-4209. doi: 10.1016/j.ijdrr.2021.102653.

Antonopoulos, A.M. and Gavin Wood Ph, D. (2018) *Mastering Ethereum: Building Smart Contracts and DApps*. *O'Reilly Media*. ISBN 9781491971949.

Avanzi, B. (2009) Strategies for dividend distribution: A review. *North American Actuarial Journal*, **13** (2), 0 217–251.

Barrieu, P. and Albertini, L. (2012) *The Handbook of Insurance-Linked Securities*. *Wiley*. ISBN 9781119206545. doi: 10.1002/9781119206545.

Blanchet-Scalliet, C., Dorobantu, D. and Salhi, Y. (2017) A model-point approach to indifference pricing of life insurance portfolios with dependent lives. *Methodology and Computing in Applied Probability*, **21** (2), 423–448. ISSN 1573-7713. doi: 10.1007/s11009-017-9611-2.

[8] See https://chaire-dialog.fr/en/welcome/.

[9] See https://anr.fr/Project-ANR-24-CE38-7885.

[10] See https://chaireactions.fr/.

[11] See https://sites.google.com/view/stephaneloisel/recherche/projets/actuariat-durable-et-risques-climatiques.

[12] See https://anr.fr/Projet-ANR-21-CE46-0002.

https://doi.org/10.1017/asb.2025.10078 Published online by Cambridge University Press

Buterin, V. (2014) A next-generation smart contract and decentralized application platform. white paper.
Clere, A. (2022) Timeline: Charting the history of parametric insurance. Online at InsurTech Digital. https://insurtechdigital.com/articles/timeline-charting-the-history-of-parametric-insurance.
Conradt, S., Finger, R. and Spörri, M. (2015) Flexible weather index-based insurance design. *Climate Risk Management*, **10**, 106–117. ISSN 2212-0963. doi: 10.1016/j.crm.2015.06.003.
Cornish, E.A. and Fisher, R.A. (1938) Moments and cumulants in the specification of distributions. *Revue de l'Institut International de Statistique/Review of the International Statistical Institute*, **5** (4), 307. ISSN 0373-1138. doi: 10.2307/1400905.
Cousaert, S., Vadgama, N. and Xu, J. (2022) *Token-Based Insurance Solutions on Blockchain*, pp. 237–260. Springer International Publishing. ISBN 9783030951085. doi: 10.1007/978-3-030-95108-5_9.
Dunn, P.K. (2004) Occurrence and quantity of precipitation can be modelled simultaneously. *International Journal of Climatology*, **24** (10), 1231–1239. ISSN 1097-0088. doi: 10.1002/joc.1063.
Feng, R. (2023) *Decentralized Insurance: Technical Foundation of Business Models*. *Springer International Publishing*. ISBN 9783031295591. doi: 10.1007/978-3-031-29559-1.
Figueiredo, R., Martina, M.L.V., Stephenson, D.B. and Youngman, B.D. (2018) A probabilistic paradigm for the parametric insurance of natural hazards. *Risk Analysis*, **38** (11), 2400–2414. ISSN 1539-6924. doi: 10.1111/risa.13122.
Fisher, S.R.A. and Cornish, E.A. (1960) The percentile points of distributions having known cumulants. *Technometrics*, **2** (2), 209–225. ISSN 1537-2723. doi: 10.1080/00401706.1960.10489895.
Goffard, P.-O. and Guerrault, X. (2015) Is it optimal to group policyholders by age, gender, and seniority for bel computations based on model points? *European Actuarial Journal*, **5** (1), 165–180. ISSN 2190-9741. doi: 10.1007/s13385-015-0106-7.
Goffard, P.-O., Rao Jammalamadaka, S. and Meintanis, S.G. (2022) Goodness-of-fit procedures for compound distributions with an application to insurance. *Journal of Statistical Theory and Practice*, **16** (3). ISSN 1559-8616. doi: 10.1007/s42519-022-00276-6.
Gweon, H. and Li, S. (2023) A hybrid data mining framework for variable annuity portfolio valuation. *ASTIN Bulletin*, **53** (3), 580–595. ISSN 1783-1350. doi: 10.1017/asb.2023.26.
Gweon, H., Li, S. and Mamon, R. (2020) An effective bias-corrected bagging method for the valuation of large variable annuity portfolios. *ASTIN Bulletin*, **50** (3), 853–871. ISSN 1783-1350. doi: 10.1017/asb.2020.28.
Kiermayer, M. and Weiß, C. (2020) Grouping of contracts in insurance using neural networks. *Scandinavian Actuarial Journal*, 1–28. ISSN 1651-2030. doi: 10.1080/03461238.2020.1836676.
Lipton, A. and Treccani, A. (2021) *Blockchain and Distributed Ledgers*. *WORLD SCIENTIFIC*. doi: 10.1142/11857.
Lopez, O. and Nkameni, D. (2025) Combination of traditional and parametric insurance: calibration method based on the optimization of a criterion adapted to heavy tail losses. Working paper or preprint. https://hal.science/hal-04959706.
Messias, J., Pahari, V., Chandrasekaran, B., Gummadi, K.P. and Loiseau, P. (2023) Understanding blockchain governance: Analyzing decentralized voting to amend defi smart contracts. arXiv preprint *arXiv:2305.17655*.
Mohan, V. (2022) Automated market makers and decentralized exchanges: A defi primer. *Financial Innovation*, **8** (1). ISSN 2199-4730. doi: 10.1186/s40854-021-00314-5.
Nakamoto, S. (2008) Bitcoin: A peer-to-peer electronic cash system. Available at https://bitcoin.org/bitcoin.pdf.
National Association of Insurance Commissioners. (2023) Parametric disaster insurance. Online on NAIC website. https://content.naic.org/insurance-topics/parametric-disaster-insurance.
Niakh, F., Bassière, A., Denuit, M. and Robert, C. (2025) Peer-to-peer basis risk management for renewable production parametric insurance.
Schär, F. (2020) Decentralized finance: On blockchain- and smart contract-based financial markets. *SSRN Electronic Journal*. ISSN 1556-5068. doi: 10.2139/ssrn.3571335.
World Economic Forum. (2025) What is parametric insurance and how is it building climate resilience? Online on the WEF website. https://www.weforum.org/stories/2025/01/what-is-parametric-insurance-and-how-is-it-building-climate-resilience/.

https://doi.org/10.1017/asb.2025.10078 Published online by Cambridge University Press