Adjoint Differentiation for Path-Dependent Assets with Payment Jumps: A Generic Θ–Scheme Framework

Niels Rom[*][†]

November 20, 2025

Abstract

A generic, fully discrete adjoint method for pricing and computing risk sensitivities of path-dependent contracts with both scheduled and stochastic payments in one-factor models is presented. Between payment dates the contract value is advanced in the diffusive state by a Theta-scheme. Path dependence is represented by a static coordinate (a half state variable) that is frozen between dates and updated only at payment times, so one tridiagonal solve per static level suffices and the PDE state dimension does not increase. Each payment date is handled by an affine update that injects cash, applies a nodewise survival/scale factor, and assembles the continuation value by interpolating across static-level rows according to an updated level.

Differentiating both the Theta-scheme and the affine payment update yields an adjoint sweep in which auxiliary vectors satisfy a transpose recurrence; this cancels explicit state variations and reduces the total gradient to a sum of local terms stored during pricing (step matrices, driver values and derivatives, and interpolation indices and weights). All Greeks with respect to all model parameters are obtained in a single adjoint pass—no bump-and-rerun.

The framework is applied to a callable mortgage-backed bond with scheduled amortisation and state-dependent prepayment. Adjoint bucketed-volatility vegas and the mean-reversion sensitivity match central finite differences to machine precision while producing the full gradient set in essentially one pricing-like run, delivering a measured speed-up of about 7.5x. The approach is storage-local, exploits tridiagonal sparsity, and is directly deployable in production risk engines across one-factor models.

1 Introduction

We present a general adjoint method for pricing and for computing risk sensitivities of path-dependent contracts with scheduled payment events between $t > 0$ and maturity. Between payment dates the contract value satisfies a one-factor parabolic partial differential equation (PDE), which we advance in time using a Θ–scheme. At payment dates, we apply a contract-specific update that (i) injects any cashflow due at that date, (ii) applies the product's continuation rule for what value survives past that date, and (iii) redistributes that surviving continuation value across the existing static-coordinate levels to reflect the updated balance (for example, after amortisation or prepayment).

Path dependence is handled by introducing a second state coordinate that is not diffusive between payment dates but is updated at payment dates. We call this coordinate a static coordinate or half state variable: it stays constant between payment dates, so it does not appear as a PDE dimension, but it is still carried forward in time and still matters for valuation. Typical

[*]Contact: https://www.linkedin.com/in/nielsrom.

[†]Thanks to Brian Huge and Bjarke Jensen for valuable inputs.

examples include an outstanding balance, a pool factor, or remaining notional. We discretise this static coordinate into levels $\{p_i\}$, and for each level we evolve one one-dimensional PDE in the diffusive state x. At each payment date we then redistribute value between these discrete levels according to the product's update rule. In this way we reproduce the effect of a two-dimensional state (x, p) without solving a two-dimensional PDE: we solve one one-dimensional PDE per static level, and only couple those levels at the discrete payment dates.

The computation is organised into two passes. In the first (pricing) pass we roll values backward in calendar time. Between payment dates we solve, for each static level, the same one-dimensional Θ–scheme PDE in the diffusive variable x. At each payment date we then apply an affine jump: a linear map that, for each static level, (i) adds an explicit term defined at that date, and (ii) forms the value that continues past the date by taking a linear combination of values from one or more static levels and multiplying it by a nodewise scaling factor. All objects required by the adjoint — local system matrices, interpolation weights, and any state-dependent drivers — are stored during this pricing pass.

In the second (adjoint) pass we compute all parameter sensitivities in a single sweep. We introduce an auxiliary sequence of adjoint vectors and propagate them through the transpose of the Θ–scheme between payment dates and through the transpose of the affine jump at each payment date. This propagation lets us express the derivative of the final price with respect to any model parameter as a sum of local terms that were already stored during pricing, multiplied by those adjoint vectors. No reruns per parameter are required: all Greeks with respect to all model parameters are obtained from one adjoint sweep.

This construction extends the adjoint finite-difference PDE machinery of Capriotti et al. (2015) in two ways. First, it supports path-dependent contracts by introducing a static coordinate (half state variable) that is carried as a label between payment dates and is only updated at payment dates, rather than being added as a full diffusive dimension. Second, it generalises cashflow handling: intermediate payments are applied via an explicit affine update (cash insertion plus survival/relabeling of the continuation), while the maturity payoff is imposed as a terminal condition with no continuation. Both extensions are achieved without increasing the PDE state dimension: between payment dates we still solve a one-dimensional PDE in the diffusive state x, one row per static level, and coupling across static levels appears only at the discrete jump dates.

The paper is organised as follows. Section 2 defines the Θ–scheme between payment dates, introduces the static coordinate, and describes the affine jump at each payment date, including how value can be reassigned across static-coordinate levels. Section 3 differentiates the Θ–scheme and derives its discrete adjoint. Section 4 differentiates the affine jump, shows how jump and PDE contributions enter the total gradient, and describes how the adjoint propagates across a payment date. Section 5 shows how the general framework applies to a callable, path-dependent fixed-income instrument by instantiating the abstract objects in Sections 2–4. Section 5 reports numerical accuracy and runtime, and Section 6 concludes.

2 The Θ–scheme and static coordinates

We work on a fixed sequence of payment dates

$$t_0 < t_1 < \cdots < t_M,$$

indexed by m in increasing calendar time. Between two consecutive dates t_m and t_{m+1} we assume that the contract value depends on two state variables:

- a diffusive state variable x that evolves stochastically under the risk–neutral measure, and
- a static coordinate (also called a half state variable) that is constant between t_m and t_{m+1} but can change at t_m itself when the payment is applied.

The static coordinate captures the contract's path dependence; in the callable mortgage-bond use case this is the pool factor. Between payment dates it is carried as a label and does not diffuse.

Dynamics and PDE

For the diffusive state x_t[1] we assume a risk–neutral dynamics of the form

$$\mathrm{d}x_t \;=\; \mu(x_t,t)\,\mathrm{d}t \;+\; \sigma(x_t,t)\,\mathrm{d}W_t, \qquad \text{with discount (killing) rate } \rho(x,t). \tag{1}$$

Here $\mu(x,t)$ is the drift of x_t under the risk–neutral measure, and $\sigma(x,t)$ is the local volatility (the instantaneous diffusion coefficient). Both may depend on the current state x and on time t. The function $\rho(x,t)$ is the instantaneous discount (or killing) rate used for valuation. We do not assume a particular parametric model for μ, σ, or ρ; they may vary in both x and t.

By the Feynman–Kac representation, the discounted continuation value $u(x,t)$ satisfies the one-dimensional parabolic PDE

$$\partial_t u(x,t) \;=\; \mathcal{L}(t)\,u(x,t), \qquad \mathcal{L}(t)u \;=\; \frac{1}{2}\,\sigma^2(x,t)\,\partial_{xx}u \;+\; \mu(x,t)\,\partial_x u \;-\; \rho(x,t)\,u. \tag{2}$$

The discount term $-\rho(x,t)u$ in $\mathcal{L}(t)$ ensures that cashflows which occur at payment dates are already valued at those dates, so there is no need to add further discounting inside the payment update.

We discretise x on a spatial grid $\{x_n\}_{n=0}^{N-1}$ and sample $u(x,t_m)$ on that grid. We denote this sampled vector by $u_m \in \mathbb{R}^N$ at time t_m.

Let $\Delta t_m = t_{m+1} - t_m$. A single Θ–scheme step on $[t_m, t_{m+1}]$ is written in matrix form as

$$A_\Theta^{(m)}\,u_m \;=\; B_\Theta^{(m)}\,u_{m+1}, \qquad A_\Theta^{(m)} = I - \Theta\,\Delta t_m\,\mathcal{L}(t_{m+1}), \qquad B_\Theta^{(m)} = I + (1-\Theta)\,\Delta t_m\,\mathcal{L}(t_m), \tag{3}$$

with $\Theta \in [0,1]$ (Crank–Nicolson corresponds to $\Theta = \frac{1}{2}$). In backward induction, u_{m+1} is known and we solve

$$u_m \;=\; \big(A_\Theta^{(m)}\big)^{-1} B_\Theta^{(m)}\,u_{m+1}. \tag{4}$$

To implement (3) we discretise the operator $\mathcal{L}(t)$ in space. On a uniform spatial grid with spacing Δx, we approximate spatial derivatives by

$$(\partial_{xx}u)_n \approx \frac{u_{n+1} - 2u_n + u_{n-1}}{\Delta x^2}, \qquad (\partial_x u)_n \approx \frac{u_{n+1} - u_{n-1}}{2\Delta x}.$$

Let $a_n = \frac{1}{2}\sigma^2(x_n,t)$, $b_n = \mu(x_n,t)$, $c_n = \rho(x_n,t)$. The resulting finite-difference operator is tridiagonal:

$$(L^h(t)\,v)_n \;=\; \alpha_n(t)\,v_{n-1} + \beta_n(t)\,v_n + \gamma_n(t)\,v_{n+1}, \tag{5}$$

where

$$\alpha_n(t) = \frac{a_n}{\Delta x^2} - \frac{b_n}{2\Delta x}, \quad \beta_n(t) = -\frac{2a_n}{\Delta x^2} - c_n, \quad \gamma_n(t) = \frac{a_n}{\Delta x^2} + \frac{b_n}{2\Delta x}.$$

Replacing $\mathcal{L}$ by L^h in (3) produces the banded matrices $A_\Theta^{(m)}$ and $B_\Theta^{(m)}$. For each interval $[t_m, t_{m+1}]$ (and for each substep inside that interval, if we substep in time), we build the lower, main, and upper diagonals of $A_\Theta^{(m)}$ and $B_\Theta^{(m)}$ and solve the tridiagonal system (4).

At the spatial boundaries we impose a zero-gradient (Neumann-type) closure using linear extrapolation,

$$u(x_0) = 2u(x_1) - u(x_2), \qquad u(x_{N-1}) = 2u(x_{N-2}) - u(x_{N-3}), \tag{6}$$

[1] In implementations x is often a transformed state (log/shift/monotone map) to improve stability or boundary placement. The coefficients μ, σ and the operator $\mathcal{L}(t)$ are written in the x–coordinate.

which is second-order accurate. In practice we either (a) enforce (6) by explicitly projecting the boundary nodes after each solve, or (b) embed these relations directly into the first and last rows of $A_\Theta^{(m)}$ and $B_\Theta^{(m)}$ so that the boundary conditions are already built into the linear systems. These finite-difference choices (grid construction, substepping for stability, and boundary treatment by projection or embedded rows) follow applied practice in production PDE solvers; see Andreasen (2023a,b) for further discussion.

Static coordinate between payments

The contract is path dependent because it keeps track of a second state coordinate that carries path information forward in time. This quantity is, in general, state dependent—for example remaining balance, outstanding notional, or pool factor (for callable mortgage bonds, the pool factor). The key modelling assumption is that this static coordinate is constant between payment dates and only changes at payment dates. This freeze-and-update treatment is used in Rom (2025) to value path-dependent callable mortgage bonds.

We handle this static coordinate without increasing the PDE dimension. Suppose we choose a finite set of discrete levels $\{p_i\}_{i=0}^{P-1}$ for this coordinate. For each level p_i and each payment date t_m, we store one solution vector

$$u_{m,i} \in \mathbb{R}^N,$$

which represents the contract value on the x–grid at time t_m assuming the static coordinate is equal to p_i. Stacking these P rows at time t_m gives a $P \times N$ array, which we denote by U_m; the ith row of U_m is $u_{m,i}$.

Between t_{m+1} and t_m, each row evolves independently in x using the same Θ–scheme in (3)–(4):

$$A_\Theta^{(m)}\, u_{m,i} \;=\; B_\Theta^{(m)}\, u_{m+1,i}.$$

The static coordinate p_i is simply carried along as a label; it does not appear as a PDE dimension.

At this stage, between two payment dates, there is no interaction between different rows. The only point where rows interact is precisely at a payment date, as described below.

Payment update at a payment date

After rolling back from t_{m+1} to t_m with the Θ–scheme for each static level p_i, we apply the payment update at t_m (we show the affine version in (7)). The relation between $C_i(s)$, $S_i(s)$, and $V_{\text{next},i}(s)$ need not be affine; we present the affine case because it covers many contracts.

Formally, let $u_i^+ \in \mathbb{R}^N$ denote the row associated with static level p_i before applying the payment at t_m[2], and let $u_i^- \in \mathbb{R}^N$ denote the row associated with static level p_i after the payment at t_m. We assume

$$u_i^- \;=\; C_i(s) \;+\; S_i(s)\, V_{\text{next},i}(s), \tag{7}$$

where $C_i(s)$ is the cashflow vector paid at t_m, $S_i(s)$ is a nodewise multiplicative survival or scale factor, and $V_{\text{next},i}(s)$ is the continuation value that survives past t_m for that row. All three depend on a scalar driver

$$s = s(x, t^\star),$$

evaluated at some representative time $t^\star \in [t_m, t_{m+1}]$ (for example $t^\star = (1-\Theta)t_m + \Theta t_{m+1}$). The driver s depends on the diffusive state x and time $t^\star$, but, at a given spatial node x_n, it does not depend on which static level p_i we are considering. We store s and its parameter sensitivities during the rollback on (t_m, t_{m+1}).

[2]Throughout, "pre-payment" (u^+) and "post-payment" (u^-) are defined with respect to the backward pricing sweep: u^+ is the grid just before the update at t_m (after rolling back from t_{m+1} to t_m), and u^- is the grid immediately after applying the payment at the same t_m. In the adjoint sweep, ψ^- is attached to u^- and is pulled back through the payment to give ψ^+.

The remaining ingredient in (7) is $V_{\mathrm{next},i}(s)$. This quantity represents "what value continues after t_m," and it also captures any jump in the static coordinate. We model the jump rule explicitly as a deterministic update map

$$Q_i(s) \;=\; g_i(s),$$

evaluated node by node in x. For each static level p_i, the function g_i encodes the contract's rule for how the static coordinate should change at t_m: starting the period at level p_i, and given the realised driver s, the static coordinate immediately after t_m is declared to be $Q_i(s)$. We refer to $Q_i(s)$ as the updated static level associated with row i. We assume g_i depends on s (and on any fixed contractual parameters at t_m), but not on the continuation values u_j^+ themselves.

Because $Q_i(s)$ will not in general coincide with one of the discrete grid levels $\{p_j\}$, we realise this jump in the static coordinate by interpolation across neighbouring rows of U_m^+. Suppose at a given spatial node x_n the updated level $Q_i(s)_n$ lies between two neighbouring grid levels p_{i_L} and p_{i_R} (so $p_{i_L} \le Q_i(s)_n \le p_{i_R}$). We define the nth component of the surviving continuation value by

$$\big(V_{\mathrm{next},i}(s)\big)_n = (1 - w_{i,n})\,\big(u_{i_L}^+\big)_n + w_{i,n}\,\big(u_{i_R}^+\big)_n, \qquad w_{i,n} \;=\; \frac{Q_i(s)_n - p_{i_L}}{p_{i_R} - p_{i_L}}, \qquad 0 \le w_{i,n} \le 1. \tag{8}$$

In other words, for each spatial node x_n we: 1. evaluate the driver $s_n = s(x_n, t^\star)$, 2. apply the contractual update rule $Q_i(s)_n = g_i(s_n)$ to get the post-payment static level, 3. find the two bracketing static levels p_{i_L} and p_{i_R}, and 4. interpolate linearly between the corresponding entries of $u_{i_L}^+$ and $u_{i_R}^+$.

Stacking these nodewise definitions over n gives the full vector $V_{\mathrm{next},i}(s) \in \mathbb{R}^N$. If, for a given row i, the contract never changes the static level at t_m, then g_i simply returns p_i at every node, so $Q_i(s)_n = p_i$, we have $i_L = i_R = i$, and the formula above reduces to $V_{\mathrm{next},i}(s) = u_i^+$.

Equation (7), together with the interpolation rule just described, is the affine jump used later for the adjoint analysis in Section 4, where we denote by ψ_i^- and ψ_i^+ the adjoint vectors attached to u_i^- and u_i^+, respectively.

Price extraction and terminal condition

After we have marched all the way back to the valuation time t_0, we evaluate the contract price from the stored grid. At t_0 we have $U_0 \in \mathbb{R}^{P\times N}$, which is the full grid across static levels and across the x–grid. The price is extracted by applying a bilinear functional,

$$P \;=\; \Phi(U_0) \;=\; \boldsymbol{w}_p^\top U_0\, \boldsymbol{w}_x, \tag{9}$$

where $\boldsymbol{w}_p \in \mathbb{R}^P$ and $\boldsymbol{w}_x \in \mathbb{R}^N$ are interpolation weight vectors.

The vector $\boldsymbol{w}_p$ selects the contract's initial level of the static coordinate. If the initial level p^* lies exactly on one of the grid points p_i, then $\boldsymbol{w}_p$ is the unit vector with a 1 at index i. If p^* lies between two neighbouring grid levels p_{i_L} and p_{i_R}, we take linear interpolation weights

$$w_p^{(i_L)} = \frac{p_{i_R} - p^*}{p_{i_R} - p_{i_L}}, \qquad w_p^{(i_R)} = \frac{p^* - p_{i_L}}{p_{i_R} - p_{i_L}},$$

and set the corresponding entries of $\boldsymbol{w}_p$ to these two values, with all other entries zero.

Similarly, $\boldsymbol{w}_x$ selects the contract's initial value x^* of the diffusive state on the spatial grid $\{x_n\}$. If x^* lies between x_{n_L} and x_{n_R} we define

$$w_x^{(n_L)} = \frac{x_{n_R} - x^*}{x_{n_R} - x_{n_L}}, \qquad w_x^{(n_R)} = \frac{x^* - x_{n_L}}{x_{n_R} - x_{n_L}},$$

and set those two components of $\boldsymbol{w}_x$ accordingly (or take a unit vector if x^* falls exactly on a grid node).

In other words, $\boldsymbol{w}_p$ and $\boldsymbol{w}_x$ simply perform linear interpolation in the static coordinate and in x. The bilinear form $\boldsymbol{w}_p^\top U_0 \boldsymbol{w}_x$ extracts the model value of the contract at its actual initial state.

At maturity t_M there is no continuation beyond that date. We therefore set the final grid U_M directly from the contractual payoff:

$$U_M = \text{payoff at maturity}, \tag{10}$$

which is typically constant across spatial nodes if the final payoff is deterministic. This U_M serves as the starting point for the backward roll.

3 Differentiating the Θ–scheme

We now derive the contribution of the between-payment Θ–scheme to the gradient of the final price with respect to a generic model parameter θ. We work on a single interval $[t_m, t_{m+1}]$ and then sum over all intervals and substeps. Throughout this section we fix one static level p_i, so $u_m \in \mathbb{R}^N$ denotes the solution vector on the x–grid at time t_m for that specific p_i. Between payment dates each static level evolves independently (Section 2), so we can treat each row separately here.

For notational clarity, write the Θ–scheme step for that row as

$$A^{(m)}(\theta)\, u_m(\theta) \;=\; B^{(m)}(\theta)\, u_{m+1}(\theta), \tag{11}$$

where $A^{(m)}$ and $B^{(m)}$ are the step matrices in (3) after spatial discretisation (including boundary treatment), and u_m is the discrete solution vector at t_m. Here, and in the remainder of this section, we index $m = 0, \ldots, M-1$ in increasing calendar time $t_0 < t_1 < \cdots < t_M$. In the actual pricing pass we solve (11) backward in time from t_{m+1} to t_m. In the adjoint sweep we will traverse these steps in forward calendar order.

Our goal is to compute the sensitivity of the final price P with respect to a model parameter θ. Since $P = \Phi(u_0)$, where u_0 is the solution vector on the x–grid at t_0 for the static level under consideration, we may write

$$\frac{\partial P}{\partial \theta} \;=\; \frac{\partial \Phi}{\partial u_0}\,\frac{\partial u_0}{\partial \theta}.$$

The first factor $\partial\Phi/\partial u_0$ is known directly from (9). The hard part is $\partial u_0/\partial\theta$.

Intuitively, $\partial u_0/\partial\theta$ is obtained by accumulating the effect that θ has on every single backward time step of the Θ–scheme: each step in (11) depends on θ through its matrices $A^{(m)}(\theta)$ and $B^{(m)}(\theta)$, and differentiating those steps tells us how a change in θ would perturb the solution at that step. If we were to propagate all of those perturbations back to t_0 and add them up, we would recover $\partial u_0/\partial\theta$.

Doing that propagation explicitly for each model parameter would be expensive. As seen from (14), computing $\partial_\theta u_m$ requires solving an additional linear system at each time step, and $\partial_\theta u_m$ must be carried all the way back to t_0. If θ is actually a vector of model parameters (for example, one volatility per bucket), we would have to repeat this sensitivity solve separately for each component. In other words, the cost would scale like bump-and-rerun: one full PDE-style sweep per parameter.

To avoid this cost we introduce, for each t_m, auxiliary vectors $\psi_m \in \mathbb{R}^N$ defined by the transpose recurrence

$$A^{(m)\top}\psi_m \;=\; B^{(m)\top}\psi_{m+1}, \qquad m = 0, \ldots, M-1, \qquad \psi_0 \;=\; \frac{\partial\Phi}{\partial u_0}. \tag{12}$$

Here $\psi_0 = \partial\Phi/\partial u_0$ follows from $P = \Phi(u_0)$. Concretely, with $U_0 \in \mathbb{R}^{P\times N}$ and $P = \boldsymbol{w}_p^\top U_0 \boldsymbol{w}_x$, the derivative with respect to the row corresponding to p_i is proportional to the x–interpolation weights $\boldsymbol{w}_x$, which we use to seed ψ_0; the recurrence then yields $\psi_1, \dots, \psi_M$ in forward calendar order. From (12) we also have the identity

$$\psi_{m+1}^\top A^{(m)} \;=\; \psi_m^\top B^{(m)}. \tag{13}$$

Differentiating (11) with respect to θ gives

$$(\partial_\theta A^{(m)})\, u_m + A^{(m)}\, \partial_\theta u_m \;=\; (\partial_\theta B^{(m)})\, u_{m+1} + B^{(m)}\, \partial_\theta u_{m+1}. \tag{14}$$

Left-multiply (14) by $\psi_{m+1}^\top$ and use (15) to replace $\psi_{m+1}^\top A^{(m)}$ by $\psi_m^\top B^{(m)}$:

$$\psi_{m+1}^\top(\partial_\theta A^{(m)})\, u_m + \psi_m^\top B^{(m)}\, \partial_\theta u_m = \psi_{m+1}^\top(\partial_\theta B^{(m)})\, u_{m+1} + \psi_{m+1}^\top B^{(m)}\, \partial_\theta u_{m+1}. \tag{15}$$

Now sum (17) over $m = 0, \dots, M-1$, where each m represents an individual stored substep (we may substep within a coupon period):

$$\begin{aligned}\sum_{m=0}^{M-1} \psi_{m+1}^\top(\partial_\theta A^{(m)})\, u_m \;+\; \sum_{m=0}^{M-1} \psi_m^\top B^{(m)}\, \partial_\theta u_m \\ = \sum_{m=0}^{M-1} \psi_{m+1}^\top(\partial_\theta B^{(m)})\, u_{m+1} \;+\; \sum_{m=0}^{M-1} \psi_{m+1}^\top B^{(m)}\, \partial_\theta u_{m+1}.\end{aligned} \tag{16}$$

Bring all $\partial_\theta u$ terms to the left. We obtain

$$\begin{aligned}\sum_{m=0}^{M-1} \psi_{m+1}^\top(\partial_\theta B^{(m)})\, u_{m+1} - \sum_{m=0}^{M-1} \psi_{m+1}^\top(\partial_\theta A^{(m)})\, u_m = \\ \sum_{m=0}^{M-1} \psi_m^\top B^{(m)}\, \partial_\theta u_m - \sum_{m=0}^{M-1} \psi_{m+1}^\top B^{(m)}\, \partial_\theta u_{m+1}.\end{aligned} \tag{17}$$

The right-hand side of (19) has a telescoping structure. Writing it term by term,

$$\begin{aligned}\sum_{m=0}^{M-1} \psi_m^\top B^{(m)}\, \partial_\theta u_m - \sum_{m=0}^{M-1} \psi_{m+1}^\top B^{(m)}\, \partial_\theta u_{m+1} = {} & [\psi_0^\top B^{(0)}\, \partial_\theta u_0] - [\psi_1^\top B^{(0)}\, \partial_\theta u_1] \\ & + [\psi_1^\top B^{(1)}\, \partial_\theta u_1] - [\psi_2^\top B^{(1)}\, \partial_\theta u_2] \\ & + [\psi_2^\top B^{(2)}\, \partial_\theta u_2] - [\psi_3^\top B^{(2)}\, \partial_\theta u_3] \\ & + \dots \\ & + [\psi_{M-1}^\top B^{(M-1)}\, \partial_\theta u_{M-1}] - [\psi_M^\top B^{(M-1)}\, \partial_\theta u_M].\end{aligned}$$

All interior terms cancel. What remains is

$$\psi_0^\top B^{(0)}\, \partial_\theta u_0 - \psi_M^\top B^{(M-1)}\, \partial_\theta u_M.$$

At maturity t_M the terminal vector u_M for each static level is set directly from the contractual payoff. In typical applications this payoff does not depend on the model dynamics parameters θ (for example drift, volatility, or discounting buckets), so $\partial_\theta u_M = 0$ and the last term vanishes. If the terminal payoff itself depends on θ, an additional boundary term $\psi_M^\top B^{(M-1)}\, \partial_\theta u_M$ must be included.

The remaining boundary term $\psi_0^\top B^{(0)}\, \partial_\theta u_0$ is exactly the chain-rule quantity $\left(\partial\Phi/\partial u_0\right)^\top \partial_\theta u_0 = \partial P/\partial\theta$, because $\psi_0 = \partial\Phi/\partial u_0$ by construction. In other words, the telescoping argument has expressed $\partial P/\partial\theta$ entirely in terms of local stepwise quantities and the adjoint vectors.

We are left with the fully local expression

$$\frac{\partial P}{\partial \theta}\Big|_{\mathrm{PDE}} = \sum_{m=0}^{M-1} \psi_{m+1}^{\top}\Big[(\partial_\theta B^{(m)})\, u_{m+1} - (\partial_\theta A^{(m)})\, u_m\Big], \tag{18}$$

where the sum runs over all stored substeps between payment dates, for the fixed static level under consideration. This is the contribution of the between-payment PDE solves to the sensitivity of the final price with respect to θ.

Operationally:

- **Forward pass between payments:** for each substep (for each static level p_i), we store u_m, u_{m+1}, the bands of $A^{(m)}$ and $B^{(m)}$, and their parameter derivatives $\partial_\theta A^{(m)}$, $\partial_\theta B^{(m)}$. Boundary conditions are either embedded directly in those bands or applied afterwards via a projection like (6), in which case we also store what is needed to replay the transpose of that projection.
- **Reverse pass between payments:** at t_0 we initialise ψ_0 from the price extraction functional (9) (this defines the contribution of each static level to the final price). We then run the adjoint forward in calendar time from t_0 to t_M, solving the transpose systems $A^{(m)\top}\psi_m = B^{(m)\top}\psi_{m+1}$ to obtain ψ_{m+1} from ψ_m, and on the way we accumulate the contribution (20). If the forward pass enforced boundary projection, we apply the transpose of that projection to ψ at the matching points in the reverse pass.

This gives the PDE (between-payment) part of the gradient. The missing piece is the effect of the payment updates themselves: at each payment date, cashflows are injected and value can move between different static levels p_i. Those effects, and their adjoint, are handled in Section 4.

4 Differentiating the payment and gradient assembly

At each payment date t_m the contract applies the affine jump (7) with continuation built by the mixing rule (8) from Section 2. We now compute the sensitivity contribution from that jump and show how the adjoint propagates across it. The scalar driver $s = s(x, t^\star)$ is evaluated on $[t_m, t_{m+1}]$, depends on the diffusive state x (not on the static index i), and is stored nodewise together with its parameter sensitivities $\partial_\theta s$.

Payment-date contribution to the gradient

At t_m the forward pass has already produced the post-payment grid

$$u_{i,n}^- = C_{i,n}(s) + S_{i,n}(s)\, V_{\mathrm{next},i,n}(s).$$

In the adjoint sweep we are at the same time with the adjoint $\psi_{i,n}^-$ attached to these post-payment entries. Differentiating the jump with respect to a parameter θ gives

$$\partial_\theta u_{i,n}^- = \Big[(\partial_s C_i)_n + V_{\mathrm{next},i,n}(\partial_s S_i)_n + S_{i,n}(\partial_s V_{\mathrm{next},i})_n\Big](\partial_\theta s)_n.$$

Weighting this nodewise variation by the post-payment adjoint and summing over rows and nodes yields

$$\frac{\partial P}{\partial \theta}\Big|_{t_m} = \sum_{i=0}^{P-1}\sum_{n=0}^{N-1} \psi_{i,n}^-\Big[(\partial_s C_i)_n + V_{\mathrm{next},i,n}(\partial_s S_i)_n + S_{i,n}(\partial_s V_{\mathrm{next},i})_n\Big](\partial_\theta s)_n. \tag{19}$$

It is helpful to view $\psi_{i,n}^-$ as the discrete derivative $\partial P/\partial u_{i,n}^-$, in which case (21) is the discrete chain rule:

$$\frac{\partial P}{\partial \theta} = \sum_{i,n} \frac{\partial P}{\partial u_{i,n}^-}\frac{\partial u_{i,n}^-}{\partial \theta}.$$

For the linear interpolation in (8), with $Q_i(s)_n = g_i(s_n)$ and $p_{i_L} \le Q_i(s)_n \le p_{i_R}$, we have

$$V_{\text{next},i,n} = (1-w_{i,n})\, u^+_{i_L,n} + w_{i,n}\, u^+_{i_R,n}, \qquad w_{i,n} = \frac{Q_i(s)_n - p_{i_L}}{p_{i_R} - p_{i_L}},$$

and hence

$$\partial_s V_{\text{next},i,n} = \frac{u^+_{i_R,n} - u^+_{i_L,n}}{p_{i_R} - p_{i_L}}\, \partial_s Q_i(s)_n = \frac{u^+_{i_R,n} - u^+_{i_L,n}}{p_{i_R} - p_{i_L}}\, g_i'(s_n). \tag{20}$$

If $Q_i(s)_n \equiv p_i$ (no change of static level), then $i_L = i_R = i$ and $\partial_s V_{\text{next},i,n} = 0$. If $Q_i(s)_n$ lies outside the tabulated levels, we clamp to the nearest level and take $\partial_s V_{\text{next},i,n} = 0$ at that node.

Adjoint propagation across the payment

After adding (21), we must move the adjoint from the post-payment grid to the pre-payment grid so we can continue over (t_m, t_{m+1}) with the transpose Θ–scheme. Let U_m^+ and U_m^- denote the pre- and post-payment grids at t_m (with rows u_i^+ and u_i^-, respectively). We collect the corresponding adjoints in matrices

$$\Psi_m^- = \{\psi^-_{i,n}\} \equiv \frac{\partial P}{\partial U_m^-}, \qquad \Psi_m^+ = \{\psi^+_{i,n}\} \equiv \frac{\partial P}{\partial U_m^+},$$

so $\psi^-_{i,n} = \partial P/\partial u^-_{i,n}$ attaches to U_m^- and $\psi^+_{i,n} = \partial P/\partial u^+_{i,n}$ attaches to U_m^+. We initialise Ψ_m^+ to zero and accumulate into it by pulling back Ψ_m^- through the payment map.

For a node that does not mix across static levels (so $V_{\text{next},i,n} = u^+_{i,n}$), the jump map is $u^-_{i,n} = C_{i,n} + S_{i,n} u^+_{i,n}$, hence $\partial u^-_{i,n}/\partial u^+_{i,n} = S_{i,n}$ and the adjoint pullback is

$$\psi^+_{i,n} = S_{i,n}\, \psi^-_{i,n},$$

i.e. we multiply by $S_{i,n}$ because only the surviving part continues past the jump.

If mixing was used, the forward map at (i,n) is $u^-_{i,n} = C_{i,n} + S_{i,n}\big[(1-w_{i,n})\, u^+_{i_L(i,n),n} + w_{i,n}\, u^+_{i_R(i,n),n}\big]$, so the only nonzero Jacobian entries are $\partial u^-_{i,n}/\partial u^+_{i_L(i,n),n} = S_{i,n}(1-w_{i,n})$ and $\partial u^-_{i,n}/\partial u^+_{i_R(i,n),n} = S_{i,n} w_{i,n}$. Accordingly, the adjoint is distributed back to the contributing pre-payment rows by

$$\psi^+_{i_L(i,n),\,n} \mathrel{+}= S_{i,n}(1-w_{i,n})\, \psi^-_{i,n}, \qquad \psi^+_{i_R(i,n),\,n} \mathrel{+}= S_{i,n} w_{i,n}\, \psi^-_{i,n},$$

accumulating when multiple (i,n) map to the same pre-payment entry. The resulting Ψ_m^+ is then marched to t_{m+1} by the transpose Θ–scheme, while we accumulate the PDE contribution as in Section 3.

Full gradient assembly

We now collect everything. The total sensitivity of the price P with respect to a parameter θ is the sum of

1. the PDE (between-payment) contribution from Section 3, (20), and 2. all payment-date contributions (21), one for each payment date t_m.

To get both parts we run two sweeps.

Forward sweep (pricing and storage).

1. Set the terminal grid U_M at maturity t_M from the payoff.
2. For $m = M-1, M-2, \dots, 0$:
   (a) Roll back from t_{m+1} to t_m: for each static level p_i solve the Θ–scheme to get the pre-payment value u_i^+ at t_m, and store the step matrices $A^{(m)}$, $B^{(m)}$, their parameter derivatives, the driver s and $(\partial_\theta s)_n$, and any boundary data.

(b) Apply the affine jump at t_m to get the post-payment values u_i^- from u^+ using $C_i(s)$, $S_i(s)$, and the interpolation data (i_L, i_R, w) that come from $Q_i(s)$, and store these jump objects.

(c) Relabel $U_m \leftarrow U_m^-$.

3. At t_0 extract the price

$$P = \boldsymbol{w}_p^\top U_0 \boldsymbol{w}_x.$$

Adjoint sweep (sensitivities).

1. At t_0 build the initial adjoint from the price extraction functional. This gives, for each static level, an adjoint on the x–grid. This is the starting adjoint at t_0.
2. For $m = 0, 1, \dots, M-1$:
   (a) At the payment date t_m we currently have the adjoint Ψ_m^- attached to the post-payment grid (this adjoint has just been transported from t_{m-1} to t_m by the transpose Θ–scheme). With this Ψ_m^-:
      i. add the payment-date contribution to the gradient using (21);
      ii. propagate the adjoint back through the jump to get the adjoint on the pre-payment grid, using

$$\psi^+_{i_L(i,n),n} \mathrel{+}= S_{i,n}(1 - w_{i,n})\, \psi^-_{i,n}, \qquad \psi^+_{i_R(i,n),n} \mathrel{+}= S_{i,n} w_{i,n}\, \psi^-_{i,n},$$

   (and $\psi^+_{i,n} = S_{i,n}\psi^-_{i,n}$ in the no-mixing case).

   (b) Between t_m and t_{m+1}: starting from this pre-payment adjoint Ψ_m^+, run the transpose Θ–scheme forward in calendar time to t_{m+1}, and at the same time accumulate the PDE part of the gradient by

$$\left.\frac{\partial P}{\partial \theta}\right|_{\text{PDE}} \mathrel{+}= \psi_{m+1}^\top \big[(\partial_\theta B^{(m)})\, u_{m+1} - (\partial_\theta A^{(m)})\, u_m\big].$$

   The result of this transpose step is the next post-payment adjoint Ψ_{m+1}^-, ready for the next payment date.

Because the same adjoint sequence is reused for every parameter θ, all Greeks (with respect to all model parameters that entered $A^{(m)}$, $B^{(m)}$, the driver s, or the jump) are obtained in this single adjoint sweep.

5 Callable mortgage-backed bond as a special case

A callable mortgage-backed bond (MBB) with scheduled coupons, scheduled principal, and state-dependent prepayment is obtained by instantiating the generic framework in Sections 2–4. We keep the same structure: one diffusive state x evolved between payment dates by the Theta-scheme, one static coordinate (the pool factor) frozen between dates and only updated at each payment date, and the affine jump

$$u_i^- \;=\; C_i(s) \;+\; S_i(s)\, V_{\text{next},i}(s)$$

at t_m as in (7), with the continuation $V_{\text{next},i}(s)$ constructed by the mixing rule (8). For additional background on callable mortgage bonds and numerical considerations, see Rom (2025). Between payment dates, rows (pool-factor levels) are uncoupled and each solves the same one-dimensional PDE in x; coupling across rows appears only through the jump-time interpolation.

On the interval $[t_m, t_{m+1}]$ we evaluate, at each spatial node x_n, a scalar driver

$$s_n \;=\; s(x_n, t^\star),$$

which feeds the prepayment model; s_n and its parameter sensitivities are stored in the forward sweep.

Let H_m denote outstanding notional just before t_m, O_m the scheduled principal at t_m, and R_m the interest due at t_m. The prepayment model delivers, for each pool-factor row i and node n, a prepayment fraction $\lambda_{m,n,i} \in [0,1]$ that depends on s_n. The cash actually paid per unit of outstanding notional is

$$C_i(s)_n \;=\; \frac{\lambda_{m,n,i}\,(H_m - O_m) + O_m + R_m}{H_m}, \tag{21}$$

and the redeemed fraction (scheduled + unscheduled) is

$$U_{m,n,i} \;=\; \frac{\lambda_{m,n,i}\,(H_m - O_m) + O_m}{H_m}. \tag{22}$$

Hence the surviving / amortisation factor that multiplies the continuation is

$$S_i(s)_n \;=\; 1 - U_{m,n,i} \;=\; 1 - \frac{\lambda_{m,n,i}\,(H_m - O_m) + O_m}{H_m}, \tag{23}$$

which is exactly the $S_i(s)$ in (7).

The static coordinate (pool factor) is updated deterministically at the payment. If the pool factor entering t_m on row i is p_i, then after scheduled amortisation and prepayment the remaining fraction at node n is

$$Q_i(s)_n \;=\; (1 - \lambda_{m,n,i})\,p_i, \tag{24}$$

which is the mortgage-specific choice of the updated static level $Q_i(s) = g_i(s)$. Because $Q_i(s)_n$ will not in general coincide with one of the discrete levels $\{p_j\}$, we obtain the continuation by the same interpolation rule (8): find $p_{i_L} \le Q_i(s)_n \le p_{i_R}$ and set

$$\big(V_{\text{next},i}(s)\big)_n = (1 - w_{i,n})\,\big(u^+_{i_L}\big)_n + w_{i,n}\,\big(u^+_{i_R}\big)_n, \qquad w_{i,n} = \frac{Q_i(s)_n - p_{i_L}}{p_{i_R} - p_{i_L}}, \quad 0 \le w_{i,n} \le 1. \tag{25}$$

This interpolation is the only point where different pool-factor rows interact; between payment dates they remain decoupled.

Optional masking (implementation detail). In the actual implementation we also allow for an optional nodewise masking of the payment step. If, at a given (i,n), the value is assessed as below par using some classification rule, we override the prepayment on that node: we set the prepayment fraction to zero, use only the scheduled cashflow, and skip pool-factor mixing by taking $V_{\text{next},i,n} = u^-_{i,n}$. The mask is stored together with the jump data $(C_i, S_i, V_{\text{next},i}, i_L, i_R, w_{i,n})$ and is replayed in the adjoint sweep; masked nodes contribute no jump sensitivity and their adjoint update is the identity, $\psi^+_{i,n} = \psi^-_{i,n}$.

Substituting (23), (25), and (27) into (7) gives, on row i and node n,

$$u^-_{i,n} \;=\; C_i(s)_n \;+\; S_i(s)_n\,\big(V_{\text{next},i}(s)\big)_n, \tag{26}$$

which is the usual mortgage payment step: cash today plus surviving continuation on the reduced pool. Because this is the same affine jump as in Section 4, the adjoint there applies verbatim. The payment-date gradient uses the post-payment adjoint ψ^- via (21), and the adjoint is then propagated back across the payment using the stored indices $(i_L(i,n), i_R(i,n))$ and weights $w_{i,n}$ as in "Adjoint propagation across the payment."

Numerical results

The model price at t_0 is

$$P = 1.0602570088.$$

Figure 1 shows the adjoint vegas $\partial P/\partial \sigma_i$ for 15 piecewise-constant volatility buckets in the one-factor short-rate model. These match central finite-difference sensitivities to machine precision across all reported Greeks (every volatility-bucket vega and $\partial P/\partial \kappa$), with absolute differences on the order of 10^{-8} for this grid and timestep choice.

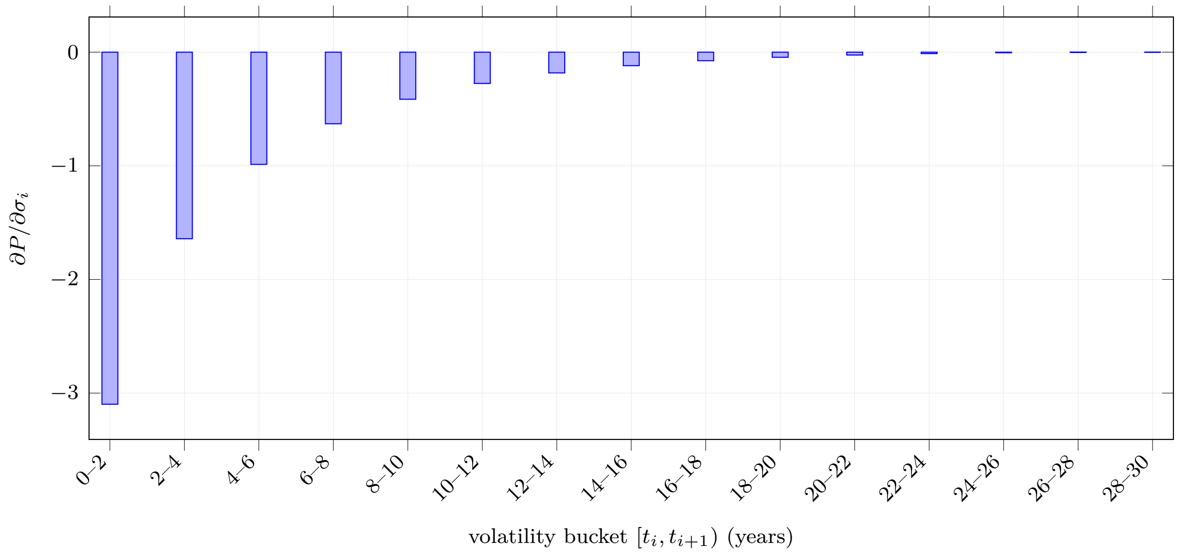

Figure 1: Adjoint vegas $\partial P/\partial \sigma_i$ for the callable MBB use case (one-factor short-rate with bucketed volatilities).

Quantity	Adjoint	Finite difference
P (price)	1.0602570088	—
$\partial P/\partial \kappa$	0.460690	0.460690
Max. abs. diff. across *all* Greeks	$< 1 \times 10^{-8}$	—

Table 1: Accuracy summary for the callable MBB use case: adjoint vs. central finite differences. All reported risk numbers match to machine precision.

Efficiency comparison (single run for adjoint vs. many bumps for finite differences):

Method	Number of solves	Total time (s)
Adjoint (all Greeks)	1	3.378
Finite differences (32 bumps)	32	25.228

Table 2: Runtime comparison. Speedup $\approx 25.228/3.378 \approx 7.5\times$.

In this setup the adjoint produces the full gradient set (all bucket vegas and $\partial P/\partial \kappa$) in essentially one pricing-like run, with a measured speedup of about $7.5\times$ relative to a naive central-difference sweep, while matching bump-and-rerun results to machine precision.

6 Conclusion

We presented a fully discrete adjoint method for pricing and computing risk sensitivities of path-dependent contracts with repeated payment jumps. The construction relies on two structural components.

Between payment dates, each row of the product state evolves independently in the Markov driver x under a Θ–scheme. This yields banded (tridiagonal in 1D) linear systems and strictly local storage per substep. Quantities such as the pool factor (or other path dependent quantities) are carried as a static coordinate: they do not diffuse between dates, so they travel only as a label and do not increase the PDE state dimension.

At each payment date the contract is advanced by an affine jump

$$u^- \;=\; C(s) + S(s)\, V_{\text{next}}(s),$$

where $C(s)$ is additive, $S(s)$ is a nodewise multiplier, and $V_{\text{next}}(s)$ is the continuation value that survives the payment. In path-dependent settings $V_{\text{next}}(s)$ is assembled by linearly mixing rows of the pre-payment grid according to the updated static level $Q(s)$; this jump kernel is the only product-specific component and, through its linear structure, provides the Jacobians needed by the adjoint.

The adjoint method is obtained by differentiating the Θ–scheme (Section 3), introducing an auxiliary sequence ψ that satisfies the transpose recurrence (12), and using that recurrence to telescope away explicit state variations. As a result, both the between-payment and the payment-date contributions (Section 4) reduce to sums of quantities already stored on the pricing sweep, weighted by ψ marched forward in calendar time. All sensitivities with respect to all model parameters are produced in a single adjoint sweep; no reruns per parameter are required. The jump redistribution across static levels is handled exactly via the stored interpolation indices and weights.

From a computational point of view, if P is the number of static-coordinate levels and N the number of spatial nodes, each substep in the forward sweep solves one tridiagonal system per level, for cost $O(PN)$ per substep. The adjoint sweep reuses the same sparsity pattern with transpose solves of identical complexity. Memory grows linearly with the number of stored substeps. Crucially, cost is essentially independent of the number of model parameters because a single pass of ψ yields all Greeks.

In the callable mortgage-backed bond use case (Section 5)—with scheduled amortisation, prepayment, and pool-factor mixing via $Q_i(s) = (1-\lambda)p_i$—all reported adjoint Greeks (bucketed volatility vegas and the mean-reversion sensitivity) matched central finite-difference results to machine precision (absolute differences $\sim 10^{-8}$ for the chosen grid/timestepping), while producing the full gradient set in one run and achieving a speedup of about $7.5\times$ over a naive bump-and-rerun sweep.

The combination of (i) dimension control via the static coordinate, (ii) a generic affine jump that encodes cashflow/survival/relabeling rules, and (iii) a discrete adjoint that walks the same grids and linear systems as pricing, makes the approach directly usable in a production risk engine: the pricing sweep automatically collects what the adjoint needs, and one adjoint sweep returns all Greeks—including jump and path-dependence effects—at essentially the cost of a single valuation run.

References

N. Rom. Callable Mortgage Bonds: Numerical Methods and Valuation Models for Pricing and Risk Analysis. Springer, 2025.

J. Andreasen. Catch Up. Wilmott, issue 123, 2023, pp. 40–45.

J. Andreasen. Fun with Finite Difference. Wilmott, issue 125, 2023, pp. 34–41.

L. Capriotti, Y. Jiang, and A. Macrina. Real-Time Risk Management: An AAD-PDE Approach. International Journal of Financial Engineering, 2(4):1550039, 2015.