The Longstaff–Schwartz Method

American Monte Carlo

Martin Burke (with Claude)

Draft

Abstract

A European option is easy to price by simulation: scatter a few hundred thousand random futures, average the discounted payoff, done. An *American* option—one its owner may cash in at any time, not only at the end—breaks that recipe, and the reason is structural, not technical. Its value is the solution of an *optimal-stopping* problem: at every moment the holder must weigh the payoff from exercising now against the value of holding on, the *continuation value*, which is an expectation over all the futures that have not happened yet. Trees and finite-difference grids compute that expectation by folding *backward* through time, but they collapse once there is more than a handful of underlyings. Monte Carlo scales to high dimension effortlessly—but it runs *forward*, and a single simulated path never sees the conditional expectation it would need to make the exercise decision. The Longstaff–Schwartz method is the bridge between the two. Its one idea is disarmingly simple: at each exercise date the simulated paths already form a cloud of points—where the underlying is now, and what each path went on to pay—and a *least-squares regression* through that cloud *is* the continuation value. Estimate it, compare it to the immediate payoff, and you have an exercise rule; replay the rule and you have a price. This note builds the method from the optimal-stopping problem up to a working algorithm, prices a small example entirely by hand, and is candid about where the bias hides and what the method cannot do. Throughout, the spine is one sentence—*Monte Carlo runs forward, early exercise reasons backward, regression is the bridge*—and a diagram for every idea.

Contents

1 The early-exercise problem

Start with the easy case, so the hard one stands out by contrast. A *European* option pays off once, at a fixed maturity T. A call on a stock pays $\max(S_T - K, 0)$; a put pays $\max(K - S_T, 0)$; whatever the contract, there is a single payoff function g evaluated at a single future date. Under the risk-neutral pricing rule its value today is an expectation,

$$V_0 \;=\; \mathbb{E}^{\mathbb{Q}}\big[\, e^{-rT}\, g(S_T)\,\big], \tag{1}$$

where $\mathbb{Q}$ is the risk-neutral measure, r the (here constant) interest rate, and e^{-rT} the factor that pulls a future cashflow back to today's money. Equation (1) is a tailor-made job for Monte Carlo: simulate a large number of random paths for S, evaluate $g(S_T)$ at the end of each, discount, and average. The law of large numbers does the rest. Nothing about the method cares whether S is one stock or a basket of fifty, whether its volatility is constant or itself random: you simulate forward, you read off the end, you average. This is why Monte Carlo is the workhorse of derivatives pricing in high dimension.

American options shatter this simplicity. An American option may be exercised at *any* time up to maturity; a *Bermudan* option—the version we will actually compute, and the one that matters in practice—may be exercised on any of a finite set of dates $t_1 < t_2 < \cdots < t_M = T$. (Everything below treats the Bermudan case; a true American option is just the limit as the dates become dense, and is priced the same way on a fine grid.) The owner of such a contract is not a passive recipient of one final payoff. At every exercise date she faces a genuine decision:

> *take the payoff now, or hold on and keep the right to exercise later?*

The contract is worth whatever an owner who decides *optimally* can extract from it. Formally, if τ denotes the (random) time she chooses to exercise—a *stopping time*, meaning the decision at each date may use only information available by then, never a peek at the future—the value today is

$$V_0 \;=\; \sup_{\tau}\; \mathbb{E}^{\mathbb{Q}}\big[\, e^{-r\tau}\, g(S_\tau)\,\big]. \tag{2}$$

The contrast with (1) is the whole story of this note. The European price is a *plain* expectation of a *known* payoff at a *fixed* date. The American price is an expectation *maximised over all non-anticipating exercise strategies*. The payoff is no longer handed to us; we have to discover *when* to take it. That maximisation is the optimal-stopping problem, and it is what makes the forward sweep of Monte Carlo, on its own, not enough.

What "decide optimally" looks like. Consider the textbook example we will use throughout: an American put with strike $K = 40$ on a non-dividend stock worth $S_0 = 36$ today, with interest rate $r = 6\%$, volatility $\sigma = 20\%$, and one year to run. Figure 1 shows a handful of simulated stock paths together with the *early-exercise boundary*—the dividing line, one value $S^*(t)$ for each date, below which immediate exercise is the better choice. The optimal rule is simply stated: *exercise the first time a path falls to or below the boundary.* A put is exercised when the stock has fallen far enough that the cash in hand, $K - S_t$, beats the prospect of waiting for it to fall further. The boundary rises toward the strike as maturity approaches: with little time left there is scant chance of a further fall, so even a modest amount in the money is worth taking. The whole job of any American pricer—trees, grids, or the method of this note—is to find that boundary, or equivalently the continuation value that defines it.

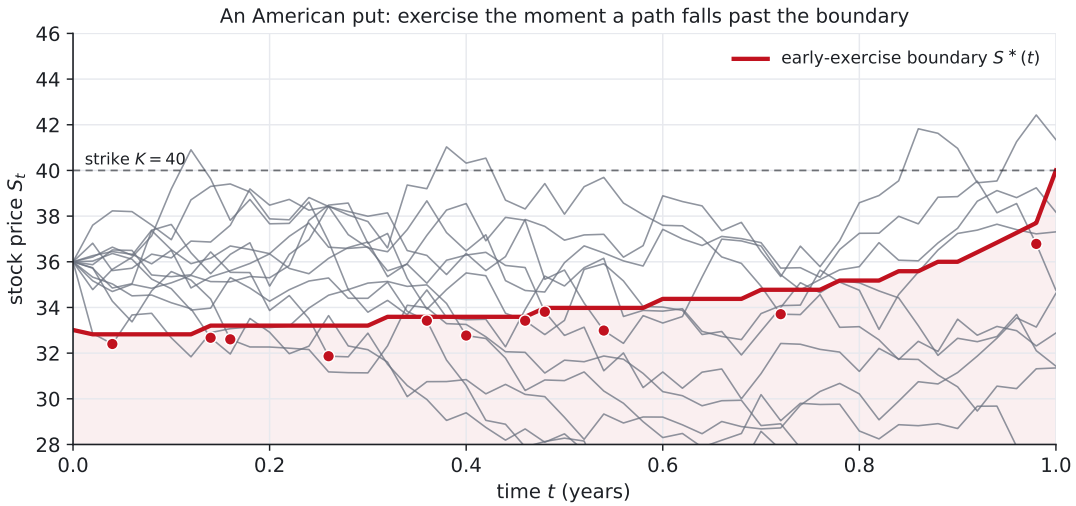


Figure 1: The early-exercise decision. Grey lines are simulated stock paths; the red curve is the early-exercise boundary $S^*(t)$ for an American put ($K = 40$, $S_0 = 36$, $r = 6\%$, $\sigma = 20\%$). The optimal policy exercises the instant a path drops into the shaded region (red dot). The boundary climbs toward the strike near maturity, because with little time left even a small gain is worth banking. Pricing the option means finding this boundary.

2 The value is a backward recursion

How does one find the boundary? By the one principle that tames every optimal-stopping problem: **dynamic programming**. Although the holder may exercise at any of many dates, the decision at each date is between exactly two things—*exercise* or *hold*—and the value of holding is itself just the value of the same problem starting one date later. That self-similarity lets us solve the whole problem *backward*, from the last date to the first.

Define the **continuation value** at date t_i, given that the underlying is currently at S_{t_i}, as the expected discounted value of *keeping* the option:

$$C_{t_i}(x) \;=\; \mathbb{E}^{\mathbb{Q}}\Big[\, e^{-r(t_{i+1}-t_i)}\, V_{t_{i+1}}\big(S_{t_{i+1}}\big) \;\Big|\; S_{t_i} = x \,\Big]. \tag{3}$$

This is the value, measured in today-at-t_i money, of all the optionality that lies *ahead*, averaged over every way the future could unfold from the present state x. With it, the holder's decision is a one-line comparison, and the option's value obeys the **Bellman recursion**

$$V_{t_M}(x) = g(x), \qquad V_{t_i}(x) \;=\; \max\Big\{ \underbrace{g(x)}_{\text{exercise now}}\,,\; \underbrace{C_{t_i}(x)}_{\text{hold}} \Big\} \quad (i = M-1, \dots, 1). \tag{4}$$

Read it right to left. At the final date there is no more choice: the option is worth its payoff, g. Step back one date: the option is worth the *better* of exercising now (g) and the continuation value C of carrying it forward—and C is built from the V we already computed at the later date. Keep stepping back, and at each date the max records the optimal decision while C carries the future value into the present. This backward fold is the engine inside a binomial tree and a finite-difference PDE solver alike; Figure 2 shows it on a single node. The exercise boundary of Figure 1 is exactly the set of states where the first branch of the max wins: $g(x) \geq C_{t_i}(x)$.

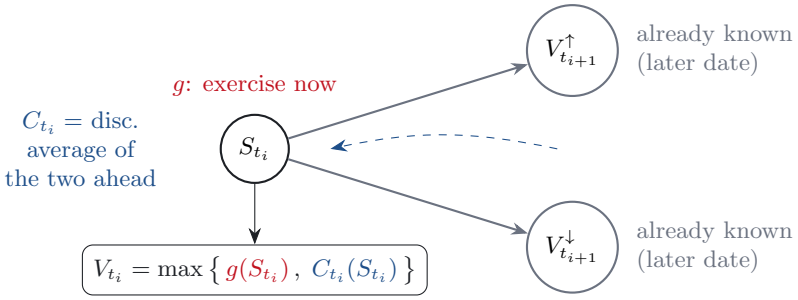


Figure 2: One step of the backward recursion. The values at the *later* date are already known; their discounted average, conditional on where we are now, is the continuation value C_{t_i}. The option's value here is the better of that and the immediate payoff g. Folding this rule from the last date back to the first solves the whole optimal-stopping problem—if one can compute C.

Everything hard is hiding in one object: the continuation value $C_{t_i}(x)$ of (3). It is a *conditional expectation*—a function of the current state, averaging over all futures. On a recombining tree it is trivial: a node has two or three children, so C is a weighted average of a couple of numbers you already hold. That triviality is exactly what fails us in high dimension, and what the next section is about.

3 Why Monte Carlo and early exercise clash

Here is the tension at the heart of American Monte Carlo, and it is worth stating sharply because the whole method is a response to it.

Monte Carlo runs forward in time. The Bellman recursion runs backward.

A Monte Carlo engine draws a random path and marches it from today to maturity, one step at a time. That is perfect for a European payoff, which is read off the end of the path. But the continuation value (3) asks a backward-looking question at every intermediate date: *given that I am here, now, what is the average value of everything that could happen next?* A single simulated path does not contain that answer. The path tells you *one* future—the one that happened to be drawn—not the *average* over all futures branching out of the current state. Standing at a point on one path, you cannot see the cloud of alternatives you would have to average. Figure 3 draws the mismatch.

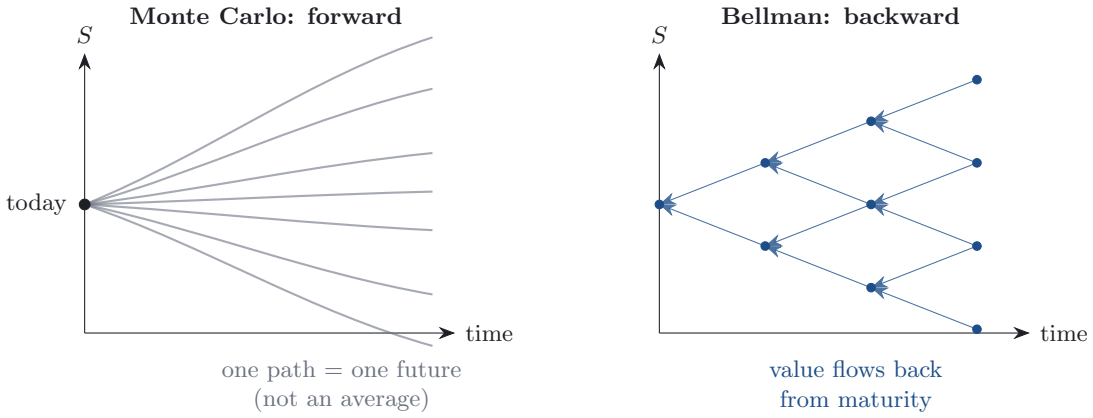
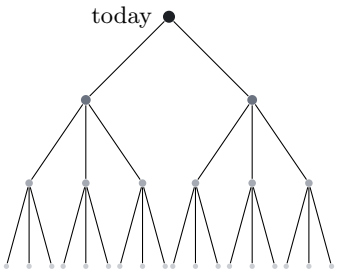


Figure 3: The clash. *Left:* Monte Carlo fans paths *forward*; each path is a single drawn future and carries no conditional average. *Right:* the Bellman recursion folds value *backward* from maturity, needing at every node the average over the futures ahead. The two run in opposite directions in time—which is why naïve simulation cannot, by itself, price early exercise.

The brute-force fix, and why it fails. One *could* force a forward engine to produce the needed averages: at every date on every path, stop and launch a fresh bundle of sub-simulations to estimate the continuation value from that state; and since each sub-simulation must itself decide whether to exercise later, it spawns *its own* sub-sub-simulations, and so on. This is *nested* Monte Carlo (Figure 4). The work multiplies at every exercise date: with M dates and b branches each, the cost explodes like b^M. For any realistic number of dates it is hopeless. We need the continuation value *without* re-simulating from every node.



cost $\sim b^M$: a fresh bundle of paths at every node, at every date

Figure 4: Nested simulation: the dead end the Longstaff–Schwartz method exists to avoid. To get a continuation value at each node by brute force, you re-simulate from it; each of those paths needs its own re-simulation at the next date, and so on. The effort branches geometrically with the number of exercise dates.

Why not just use a tree, then? On a low-dimensional recombining lattice the backward fold of §2 is cheap and exact, and for one or two underlyings that is the right tool. But the number of nodes in a grid grows like $(N+1)^d$ in the number of risk factors d: tractable at $d = 1$, strained at $d = 3$, and utterly out of reach by the time you reach a five-factor Bermudan swaption or an option on a basket of equities (Figure 5). Monte Carlo's cost, by contrast, grows only *linearly* in d—its great virtue, and the reason we are unwilling to abandon it just because early exercise is awkward. The Longstaff–Schwartz method keeps the forward simulation and buys back the backward recursion cheaply.

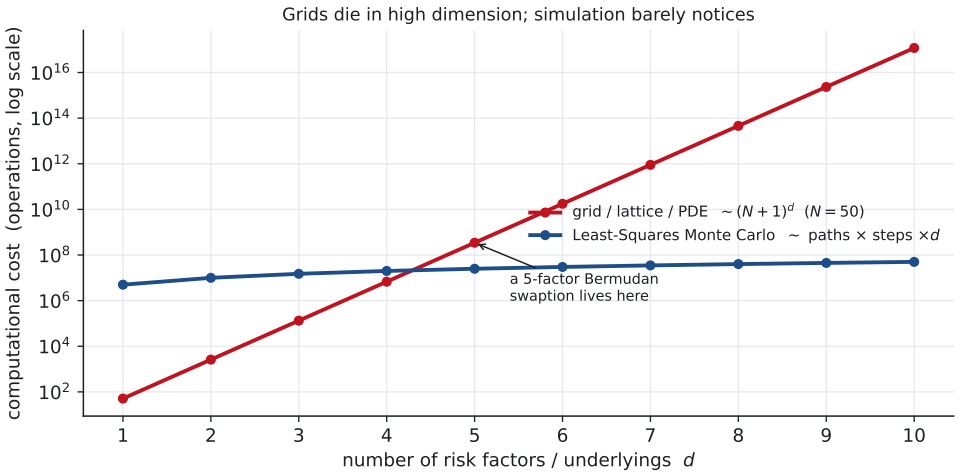


Figure 5: Why a simulation-based method is worth the trouble. A grid or lattice has $(N+1)^d$ nodes—exponential in the number of risk factors d—so it is overwhelmed by the time a problem has a handful of factors. Monte Carlo cost grows only linearly in d. Past a few dimensions, simulation is the *only* option, and we must teach it to handle early exercise.

4 The one idea: regress to get the continuation value

Here is the insight that makes everything work, and it is worth slowing down for because the entire method is this single move.

We are stuck because, standing on one path at date t_i, we cannot see the average over futures that the continuation value (3) demands. But we are not standing on *one* path—we have simulated tens of thousands of them. *Across the whole bundle*, paths that happen to be near the same state $S_{t_i} = x$ went on to many different futures, and *their realised discounted payoffs are exactly samples of the quantity we want to average.* The continuation value is, by definition, the *conditional mean* of those future payoffs given the current state. And estimating a conditional mean from a cloud of (input, output) pairs is the oldest job in statistics: **regression**.

Concretely, fix an exercise date t_i. For every simulated path j record two numbers:

- the state now, $X_j = S_{t_i}^{(j)}$ (the regressor), and
- the realised discounted cashflow that path *actually goes on to collect* from following the optimal policy after t_i, call it Y_j (the response).

Each path contributes one dot (X_j, Y_j). Plotted, they form a scatter (Figure 6)—noisy, because Y_j is one random future, but with a clear trend: the lower the stock, the more a put pays later. The continuation value $C_{t_i}(x)$ is the height of the *mean* of that cloud at x. We recover it by fitting a curve through the cloud by least squares: choose a small set of *basis functions* $\psi_1, \ldots, \psi_L$ of the state (a constant, the stock, its square, maybe a couple more) and solve

$$\widehat{\beta} \;=\; \arg\min_{\beta \in \mathbb{R}^L} \sum_{j \text{ in the money}} \Big(Y_j \;-\; \sum_{k=1}^{L} \beta_k \, \psi_k(X_j) \Big)^2, \qquad \widehat{C}_{t_i}(x) \;=\; \sum_{k=1}^{L} \widehat{\beta}_k \, \psi_k(x). \tag{5}$$

The fitted curve $\widehat{C}_{t_i}$ is our estimate of the continuation value—the blue curve in Figure 6. That is the entire trick. A conditional expectation we could not see along any single path is read straight off the *cross-section* of all paths, by the same least-squares fit one might run in a spreadsheet. Because the method fits the continuation value by least squares, it is also called **Least-Squares Monte Carlo** (LSM).

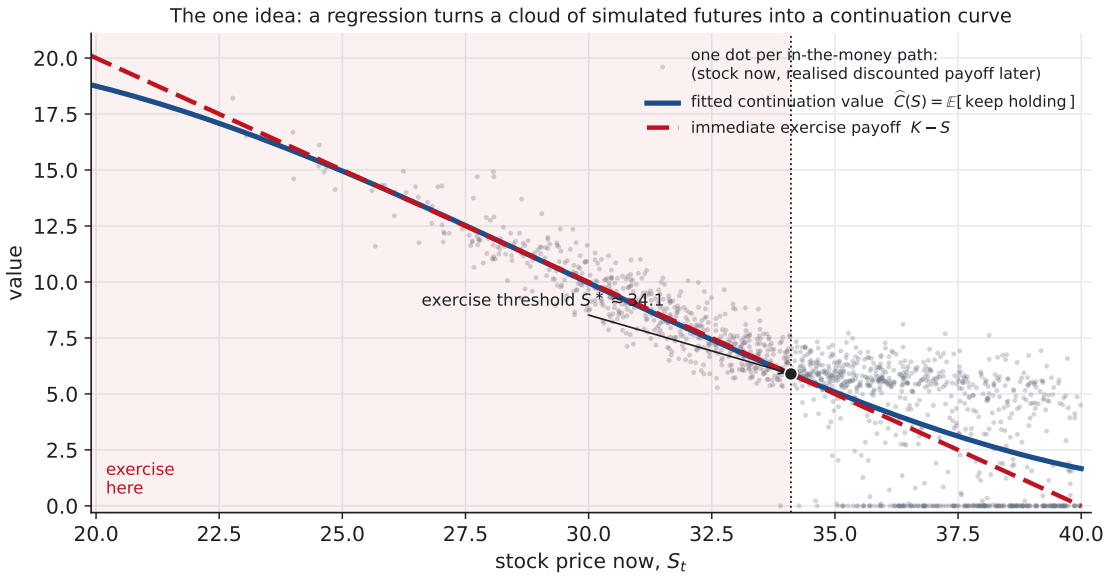


Figure 6: The one idea, in one picture. At a chosen exercise date, each in-the-money path contributes a grey dot: *where the stock is now* (horizontal) against *the discounted payoff that path goes on to collect* (vertical). The cloud is noisy, but its conditional mean—the blue regression curve—is the continuation value $\widehat{C}(S)$, the value of holding on. Compare it to the immediate exercise payoff $K - S$ (red): where exercising wins, to the left of their crossing, lies the exercise region. The crossing is the exercise threshold S^*.

Only the crossing matters. Look again at Figure 6. The exercise decision at this date is not "what is the continuation value?" but the cruder "*is the immediate payoff g bigger than the continuation value, or not?*"—a comparison of two curves, settled entirely by *which is higher*. All that the regression has to get right is the *side of the crossing* each path falls on. This is a forgiving target. The fitted curve can be wrong about the continuation value by a fair margin almost everywhere and still place nearly every path on the correct side of the threshold, hence still produce a near-optimal exercise rule. It is the reason a crude polynomial basis works so well in practice, and the reason the method is far more robust than fitting a full value surface would suggest. We are after the *boundary*, not the surface.

Why in-the-money paths only. The sum in (5) runs only over paths that are *in the money* at t_i (for the put, $S_{t_i} < K$). The reason is practical and pointed: the exercise decision is only ever live where the immediate payoff is positive—an out-of-the-money put would never be exercised, so its continuation value is irrelevant to any decision. Throwing the out-of-the-money paths into the regression would spend the basis functions fitting a region we never consult, pulling the fit away from where it must be accurate—near the crossing. Restricting to in-the-money paths concentrates the regression's effort exactly where the decision is made. This small choice noticeably sharpens the method.

5 The algorithm in full

We can now assemble the pieces. The continuation value comes from a regression; the Bellman recursion tells us to work backward; the only bookkeeping is to carry, along each path, the cashflow it will collect under the exercise policy as that policy is discovered date by date.

It is cleanest to picture the state of the computation as a **cashflow matrix**: one row per simulated path, one column per exercise date (Figure 7). We process the columns from right

(maturity) to left. At maturity every in-the-money path's cell holds its payoff. Stepping back to date t_i, we (i) discount whatever cashflow each path currently carries from its later exercise back to t_i; (ii) regress those discounted cashflows on the state, over in-the-money paths, to get $\widehat{C}_{t_i}$; (iii) for each in-the-money path, compare the immediate payoff $g(S_{t_i})$ with $\widehat{C}_{t_i}(S_{t_i})$, and if exercising wins, *overwrite* that path's cashflow with the immediate payoff and mark its exercise date as t_i (a later exercise it had pencilled in is cancelled—you cannot exercise twice). Repeat back to t_1. The price is the average, over all paths, of each path's single cashflow discounted from its exercise date to today.

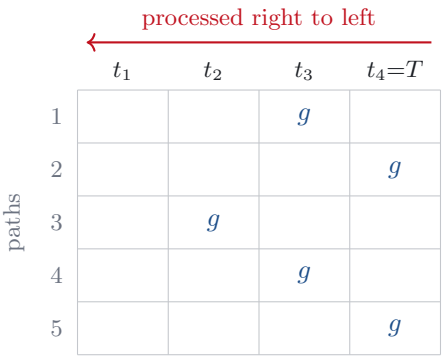


Figure 7: The bookkeeping. Rows are paths, columns are exercise dates. Sweeping right to left, the regression at each date decides which still-unexercised in-the-money paths should exercise *there*; each path ends with a single cashflow g in one column (its optimal stopping date). Averaging those, discounted to today, gives the price.

Algorithm 1 writes this out. Notice there are two distinct roles for the paths, and conflating them is the classic mistake (§7): the regression *learns the policy* (where to exercise), and a forward replay of that policy *values* it. The figure of merit produced inside the loop is already a price, but the cleanest, lowest-bias estimate comes from applying the learned policy to a *fresh* set of paths.

Input: exercise dates $t_1 < \cdots < t_M = T$; payoff g; basis $\{\psi_k\}_{k=1}^{L}$; N paths.

1. Simulate N risk-neutral paths $S_{t_1}^{(j)}, \ldots, S_{t_M}^{(j)}$.
2. Initialise each path's cashflow $c_j \leftarrow g\big(S_{t_M}^{(j)}\big)$ and stopping date $\tau_j \leftarrow t_M$.
3. **for** $i = M-1$ **down to** 1:
4. Discount carried cashflows one period: $c_j \leftarrow e^{-r(\tau_j - t_i)} c_j$ (bring each to date t_i).
5. Let $\mathcal{I} = \{\, j : g(S_{t_i}^{(j)}) > 0 \,\}$ (the in-the-money paths).
6. Fit $\widehat{\beta}$ by least squares: regress $\{c_j\}_{j\in\mathcal{I}}$ on $\{\psi_k(S_{t_i}^{(j)})\}$ (eq. 5).
7. **for** each $j \in \mathcal{I}$:
8. $\widehat{C} \leftarrow \sum_k \widehat{\beta}_k\, \psi_k(S_{t_i}^{(j)})$ (estimated continuation value).
9. **if** $g(S_{t_i}^{(j)}) \geq \widehat{C}$ **then** $c_j \leftarrow g(S_{t_i}^{(j)})$, $\tau_j \leftarrow t_i$ (exercise here).
10. **Price** $\widehat{V}_0 = \frac{1}{N} \sum_j e^{-r\tau_j}\, c_j$ (or, lower-biased, replay $\{\widehat{\beta}\}$ on fresh paths).

Algorithm 1: Longstaff–Schwartz / Least-Squares Monte Carlo (Bermudan option)

Choosing the basis. The basis functions $\{\psi_k\}$ are the method's one real degree of freedom. They must be rich enough to bend into the shape of the true continuation value, but no richer—a

basis with too many terms will chase the noise in the cloud rather than its mean, a textbook over-fit. Figure 8 shows the Goldilocks problem on our put: a straight line (degree 1) is too stiff to capture the curvature near the money; a degree-12 polynomial wriggles to touch stray points and would mis-rank paths near the boundary; a low-order polynomial (degree 3) tracks the truth. In practice a handful of terms suffices—powers of the underlying, or the weighted Laguerre polynomials of the original paper—and, crucially, the forgiving "only the crossing matters" property of §4 means the price is remarkably insensitive to the exact choice once the basis is rich enough to bend.

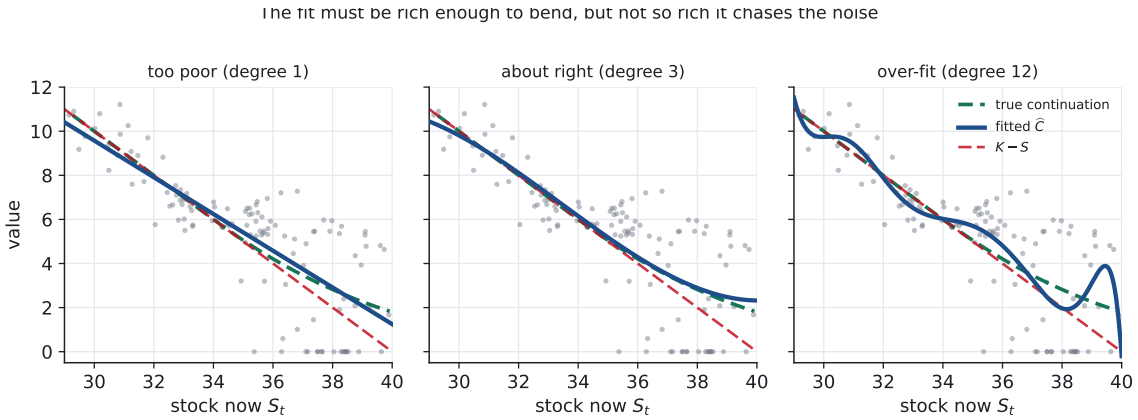


Figure 8: Choosing the regression basis. *Left:* a straight line cannot bend to the curvature of the continuation value (green dashed truth). *Centre:* a degree-3 polynomial tracks it well. *Right:* a degree-12 polynomial over-fits, wriggling to chase noise—harmless in the dense middle, dangerous at the sparse edges where it can mis-rank paths against the exercise line. Rich enough to bend, not so rich it chases noise.

Does it actually find the boundary? Yes. Although the method never solves for the exercise boundary directly—it only ever compares two numbers, path by path—the boundary emerges as the locus of crossings, date by date. Figure 9 overlays the boundary reconstructed from the LSM regressions on the exact one from a fine binomial tree. They agree. The regressions, run on nothing but a forward simulation, have recovered the same early-exercise frontier a backward tree would give—which is the whole claim of the method.

Figure 9: The regressions recover the right boundary. Red dots: the exercise frontier implied, date by date, by the LSM continuation-value fits. Green: the exact boundary from a fine binomial tree. A purely forward, regression-based method reconstructs the same early-exercise frontier as the backward lattice.

6 A worked example, entirely by hand

Nothing fixes the algorithm in the mind like grinding through a tiny instance, and the original paper's eight-path example is perfect for it. The contract is an American put with strike $K = 1.10$, exercisable at three dates $t = 1, 2, 3$, on a stock starting at $S_0 = 1$, with interest rate 6% (so the one-period discount factor is $e^{-0.06} = 0.9418$). Table 1 lists eight simulated stock-price paths.

Table 1: Eight simulated stock-price paths (Longstaff–Schwartz, 2001).

Path	$t = 0$	$t = 1$	$t = 2$	$t = 3$
1	1.00	1.09	1.08	1.34
2	1.00	1.16	1.26	1.54
3	1.00	1.22	1.07	1.03
4	1.00	0.93	0.97	0.92
5	1.00	1.11	1.56	1.52
6	1.00	0.76	0.77	0.90
7	1.00	0.92	0.84	1.01
8	1.00	0.88	1.22	1.34

Last date, $t = 3$. There is no choice left: every in-the-money path takes its payoff $\max(1.10 - S_3, 0)$. Only paths 3, 4, 6 and 7 are in the money, paying $0.07, 0.18, 0.20, 0.09$ respectively; the rest pay nothing.

Step back to $t = 2$. Five paths are in the money: 1, 3, 4, 6, 7 (those with $S_2 < 1.10$). For each, the regressor X is the stock now, and the response Y is the $t = 3$ payoff *discounted back one period* to $t = 2$ (multiply by 0.9418); paths that paid nothing at $t = 3$ contribute $Y = 0$. Regressing Y on $1, X, X^2$ over these five points gives

$$\widehat{C}_2(X) \;=\; -1.070 + 2.983\,X - 1.814\,X^2. \tag{6}$$

Now compare, path by path, the immediate payoff $1.10 - S_2$ with $\widehat{C}_2(S_2)$ (Figure 10, left). Exercising wins for paths 4, 6 and 7 (deep enough in the money that the cash beats holding); for paths 1 and 3 the continuation value is higher, so they hold. The three exercising paths have their cashflow set to the immediate payoff at $t = 2$; their $t = 3$ entries are cancelled.

Step back to $t = 1$. Again five paths are in the money: 1, 4, 6, 7, 8. The response Y is now each path's *realised* cashflow—whatever it collects at $t = 2$ or $t = 3$ under the policy just fixed—discounted back to $t = 1$. Regressing on $1, X, X^2$ gives

$$\widehat{C}_1(X) \;=\; 2.038 - 3.335\,X + 1.356\,X^2, \tag{7}$$

and comparing immediate payoff to continuation (Figure 10, right) sends paths 4, 6, 7 and 8 to exercise at $t = 1$, while path 1 holds.

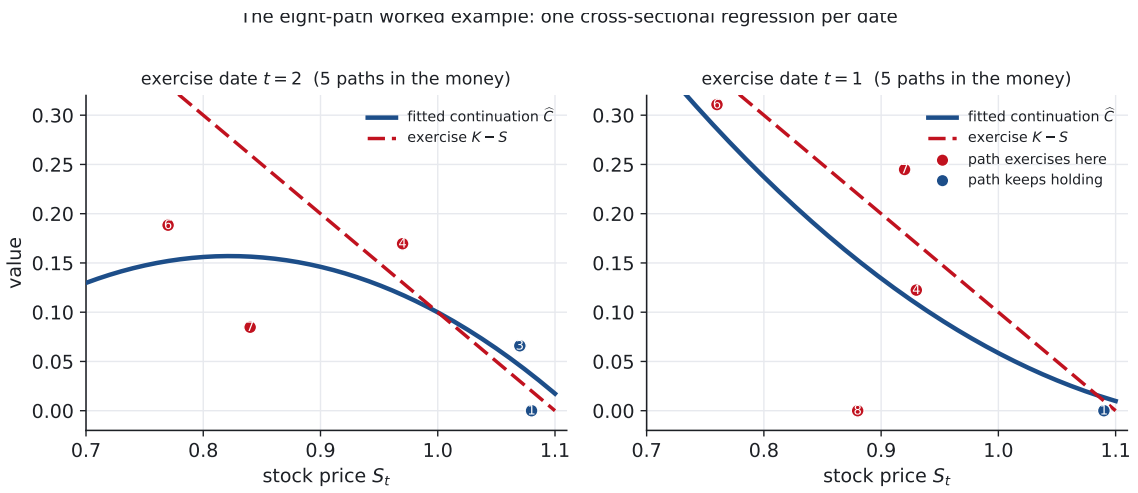


Figure 10: The eight-path example, one regression per date. Each numbered dot is an in-the-money path at (S_t, discounted realised payoff later); the blue curve is the fitted continuation $\widehat{C}$, the red dashed line the immediate payoff $K - S$. A dot is red if that path exercises here (payoff above the continuation curve's verdict) and blue if it holds. *Left:* $t = 2$, exercise 4, 6, 7. *Right:* $t = 1$, exercise 4, 6, 7, 8.

Collect the price. Each path now has exactly one cashflow, at its optimal stopping date. Discount each to $t = 0$ and average over all eight:

$$\widehat{V}_0 \;=\; \frac{1}{8}\sum_{j=1}^{8} e^{-0.06\,\tau_j}\, g\big(S^{(j)}_{\tau_j}\big) \;=\; 0.1144. \tag{8}$$

For comparison, the value of the *European* put on the same paths—exercise only allowed at $t = 3$—is 0.0564. The right to exercise early has roughly doubled the option's worth, and the eight-path machine has measured exactly that, using nothing but two little least-squares fits. Appendix D gives a forty-line program that reproduces (6)–(8) and scales straight up to the fifty-date, forty-thousand-path version of the put from §1.

7 Bias, and the two estimators

The method is not free of subtlety, and the subtlety is all about *bias*—small systematic errors that do not wash out no matter how many paths you throw at it. Being clear-eyed about them is what separates a toy implementation from a production one.

Two ways to use the paths, two biases. The regressions learn an exercise policy. There are two things one might then do, and they err in opposite directions.

- **Value the policy in-sample.** Use the very same paths that trained the regressions to also report the price (as line 10's first form does). This is biased *high*. The fitted continuation value has tuned itself to the particular noise in these paths, so the exercise decisions enjoy a sliver of hindsight—a "foresight bias"—that inflates the value. The richer the basis, the more it can over-fit, and the larger this upward bias grows.
- **Value the policy out-of-sample.** Freeze the trained policy $\{\widehat{\beta}\}$ and apply it to a *fresh, independent* set of paths, exercising each strictly by the rule. This is biased *low*, but only because the estimated policy is necessarily a little sub-optimal, and *any* fixed exercise rule is worth no more than the optimal one. Crucially this gives a genuine *lower bound* on the true price—a number you can trust to be conservative.

Figure 11 shows both as the basis is enriched. The true, out-of-sample estimate climbs as the basis improves and then levels off just below the true value; the in-sample estimate sits above it and, tellingly, keeps drifting *up* as the basis grows—the signature of over-fitting. The practical rule follows immediately: *train on one set of paths, price on another.*

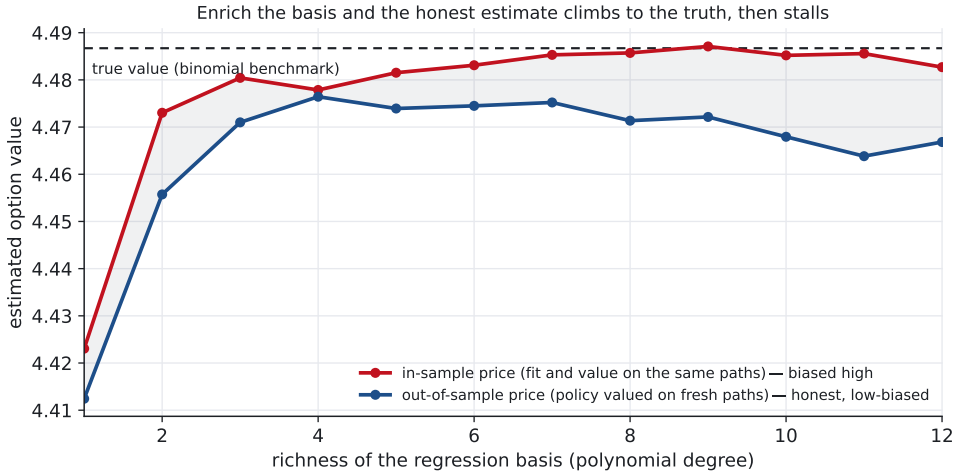


Figure 11: The two estimators bracket the truth. Both use the same regressions, but the in-sample price reuses the training paths and is biased high—worse as the basis grows richer (over-fit)—while the out-of-sample price replays the policy on fresh paths and is genuinely low-biased, a true lower bound. Report the latter.

Monte Carlo error, on top of bias. Separate from bias is plain sampling noise: with finitely many paths the price wobbles run to run. That error shrinks at the usual Monte Carlo rate—like $1/\sqrt{N}$ in the number of paths N—and is easily quantified from the spread of pathwise cashflows. Figure 12 shows the estimate tightening onto the benchmark as N grows. (Note the small-N estimates here sit *above* the answer: with few paths the in-sample foresight bias dominates, another reminder to separate training from pricing.)

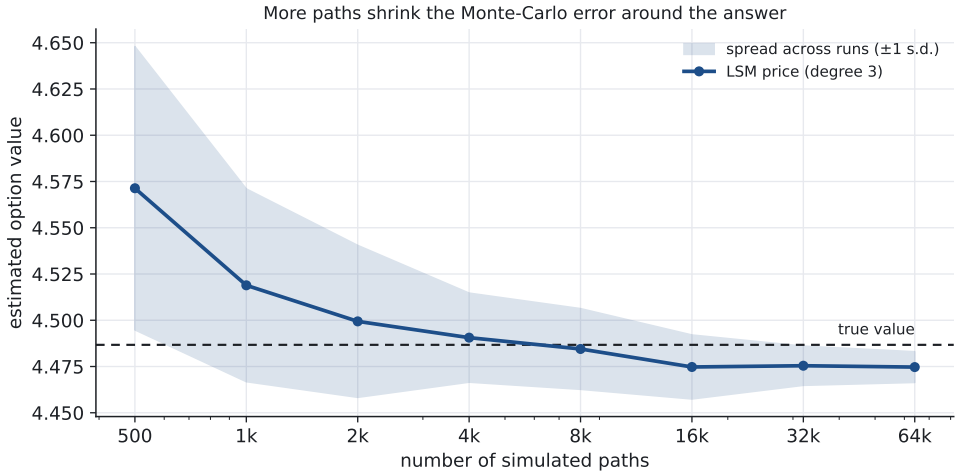


Figure 12: Convergence in the number of paths. The shaded band is the run-to-run spread (±1 s.d.); it narrows like $1/\sqrt{N}$ as paths are added, and the estimate settles onto the binomial benchmark. Sampling error and bias are distinct: more paths cure the former, not the latter.

An upper bound, to close the gap. A lower bound alone leaves you unsure how much value the sub-optimal policy is leaving on the table. A beautiful piece of theory —the *dual* formulation of optimal stopping, due to Rogers and to Haugh and Kogan, made practical by Andersen and Broadie—turns any exercise policy into an *upper* bound as well, by adding a cleverly constructed martingale "penalty" for the foresight a perfect policy would have. Computing it costs more (it typically involves sub-simulations to build the martingale), but it brackets the true price from above. With the LSM lower bound below and the dual bound above, one can state the price to within a guaranteed interval rather than trusting a single number—the gold standard for a careful Bermudan valuation. The mechanics are beyond this note; the point to carry away is that the low-biased LSM price is one half of a rigorous bracket, not a lone estimate to be taken on faith.

8 Who holds the call: two kinds of callability

So far the early-exercise right has belonged to the option's *owner*, who exercises to make the contract worth as much as possible to themselves. That is one of two ways a product can be callable, and they sit on opposite sides of the same optimal-stopping problem. The method handles both; the difference is a single sign.

Holder-exercisable: American and Bermudan options. This is the running example of the whole note. The *holder* chooses when to exercise, and chooses to *maximise* their own value, so at each date the value is

$$V_{t_i}(S) \;=\; \max\Big\{ \underbrace{g(S)}_{\text{exercise now}} \,,\; \underbrace{C_{t_i}(S)}_{\text{hold on}} \Big\}, \tag{9}$$

and the holder exercises where the immediate payoff beats the continuation value—the left of the crossing in Figure 6. A *Bermudan* option carries this right on a fixed schedule of dates $t_1 < \cdots < t_M$; an *American* option carries it continuously, and is priced (§9) as the refinement limit of a Bermudan on a fine grid. Either way the optionality *adds* value: more rights to the holder can only raise the price.

Issuer-callable: the right belongs to the seller. Many structured notes, callable bonds, and cancellable swaps instead give the *issuer* the right to redeem the product early, usually for a pre-agreed *call price* K^{call} (the outstanding notional, perhaps plus a small premium). The issuer is not trying to help the holder—they call to *minimise* the value of their own liability, which is the same as minimising the value to the holder. The Bellman comparison therefore flips from a max to a min:

$$V_{t_i}(S) \;=\; \min\Big\{ \underbrace{K^{\text{call}}}_{\text{call now}} \,,\; \underbrace{C_{t_i}(S)}_{\text{leave alive}} \Big\}. \tag{10}$$

The issuer redeems exactly when the continuation value of the liability *exceeds* the call price—when it has become more expensive to keep the note alive than to buy it back—so the call region is now where C is *high*, the mirror image of the holder's. This optionality *subtracts* value: an issuer call right can only lower the note's worth to its holder, which is why callable notes pay a richer coupon to compensate.

The canonical case: rates, where calling means refinancing. The purest example is interest-rate debt, and the intuition is the one every homeowner knows. An issuer who sold a callable bond is locked into paying a *fixed* coupon, and the state S driving the continuation value is now the prevailing level of interest rates. If rates *fall*, that fixed coupon becomes expensive relative to what the issuer could pay on fresh debt: the continuation value of the liability

C_{t_i}—the present value of all those now above-market coupons still to come—climbs above the call price K^{call}. So the issuer calls the old bond and *refinances* at the new, lower rate, exactly as a homeowner refinances a mortgage when rates drop. If rates *rise*, the locked-in coupon is now cheap, a bargain worth keeping, and the issuer holds. The abstract rule "call when $C_{t_i} \geq K^{\text{call}}$" thus reads, in rates, as *"call once rates have fallen far enough to make refinancing worthwhile."* This is precisely the optionality embedded in callable bonds and the cancellable interest-rate structures of §10: the regression state is one or more yield-curve factors, and the continuation value is fitted as a function of them by the same least-squares cross-section as before.

The same engine, but not merely a flipped sign. The bridge of this note—regress the cross-section of paths to recover the continuation value C_{t_i}—is identical in both cases. The scatter, the basis, the least-squares fit, the backward sweep, the cashflow matrix: the machinery that *estimates* the continuation value is unchanged, which is why one pricer handles both and why the method became the market workhorse for the callable exotics of §10. But the issuer-callable case is *not* simply the holder's recursion with max swapped for min. Two things genuinely change, and both follow from *who* optimises and *what* they hold.

- **The optimal stopping runs in the *min* direction.** The holder stops to make the contract worth as much as possible (eq. 9, a max); the issuer redeems to make their liability worth as little as possible (eq. 10, a min). Step 9 of Algorithm 1 reverses to match: "exercise if $g(S) \geq \widehat{C}$" becomes "call if $\widehat{C} \geq K^{\text{call}}$."

- **The regression runs over *all* paths, not the in-the-money ones.** Recall why the put regressed over in-the-money paths only (§4): an out-of-the-money put is never exercised, so its continuation value enters no decision and throwing it into the fit would only blur it. A callable coupon note has no such dead region. It carries value—the present value of the coupons still to come—on *every* path, so its continuation value is positive everywhere and the issuer's call decision is live on every path, whatever the level of rates; the immediate value of calling, the call price K^{call}, is positive too. With nothing to gate on, the in-the-money filter admits the entire sample, and the regression is fitted across the *whole* cross-section of paths. Dropping that filter is a substantive change to the algorithm, not a cosmetic one.

A decision rule the issuer exercises against the holder is thus found by the very same forward simulation and least-squares fit that values the holder's own right—with the optimisation reversed and the in-the-money restriction lifted.

9 Practical choices and pitfalls

A short field guide to making the method behave, gathering the threads above.

- **Separate training from pricing.** The single most important habit, for the bias reason of §7: fit the regressions on one set of paths, then value the resulting policy on an independent set. Reusing paths quietly inflates the price.

- **Keep the basis modest.** Three to six terms in the underlying—low-order monomials, or Laguerre/Hermite polynomials for numerical conditioning—are usually ample. Because only the side of the crossing matters (§4), resist the urge to add terms; richer is not safer, it over-fits (Figure 8). In several underlyings, build the basis from each underlying and a few cross terms, not a full high-degree tensor product, which explodes.

- **Regress in-the-money only.** It both sharpens the fit where decisions are made and saves work (§4).

- **Mind the conditioning.** Powers of the underlying become badly correlated as the degree rises, making the least-squares normal equations ill-conditioned; orthogonal polynomials, scaling the underlying to $O(1)$, or a small ridge penalty all help.
- **Exercise dates.** A Bermudan is priced on its actual exercise schedule; a continuously exercisable American is approximated by a fine grid of dates, the price rising toward the true American value as the grid is refined. More dates means more regressions, but the cost is linear in their number.
- **Greeks need care.** The exercise policy introduces kinks, so naïve bump-and-revalue sensitivities are noisy. Re-using the *same* policy and the *same* random numbers across bumps, or pathwise/adjoint differentiation holding the policy fixed, gives far steadier Greeks.
- **Variance reduction stacks cleanly.** Antithetic variates (used for every figure here), control variates—often the European version of the same option, whose price is known—and low-discrepancy sequences all reduce the Monte Carlo error of §7 without touching the regression machinery.

10 Where it shines, and what it cannot do

The Longstaff–Schwartz method earns its keep precisely where the alternatives fail: in *high dimension*. Because it is built on forward simulation, its cost grows linearly, not exponentially, in the number of risk factors (Figure 5), so it prices instruments a lattice cannot touch:

- **Bermudan swaptions** and other callable interest-rate structures, where the state is several yield-curve factors;
- **American/Bermudan options on baskets** of many equities, and best-of/worst-of structures;
- **Callable and cancellable exotics**—callable range accruals, autocall features, employee stock options with early exercise—where the payoff is path-dependent and the state high-dimensional.

In each case one simply enlarges the state fed to the regression and proceeds exactly as in §5. That generality, on top of an ordinary Monte Carlo engine, is why the method became the market standard for early-exercise products within a few years of its publication. Table 2 places it against the alternatives.

Table 2: Pricing early exercise: rough comparison.

	Lattice / tree	**PDE grid**	**Longstaff–Schwartz**
Dimension it handles	1–2 (3 at a push)	1–3	many (10+)
Cost in dimension d	$\sim (N{+}1)^d$	$\sim N^d$	$\sim N{\cdot}M{\cdot}d$ (linear)
Path-dependence	awkward	awkward	natural
Early exercise	exact, easy	exact, easy	via regression
Error type	discretisation	discretisation	bias + MC noise
Gives bounds?	converges	converges	lower bound (+ dual upper)

The method's limits. The method is an *approximation*, and its weaknesses are the flip side of its strengths.

- **It is biased, not exact.** Unlike a convergent tree or PDE, no number of paths removes the bias from a sub-optimal fitted policy. Trustworthy use means the lower/upper bracket of §7, not a single figure.
- **The basis is a modelling choice.** A poorly chosen basis can mis-place the boundary, especially deep in or out of the money and near maturity where the continuation value bends sharply. There is no fully automatic recipe; some judgement or experimentation is unavoidable.
- **It targets the boundary, not the surface.** The method delivers a good *price* and exercise rule, but the fitted continuation values away from the boundary are only roughly right—so do not read them as a trustworthy value surface, and treat regression-based Greeks with the caution of §9.
- **Very high dimension still bites the basis.** Linear cost in d is the good news; the bad news is that capturing the continuation value's *shape* in very many dimensions may need a large or cleverly chosen basis, which is where modern variants reach for richer regressors—kernels, or neural networks in place of the polynomial fit, the same idea with a more expressive curve through the cloud.

None of this dents the central achievement. An optimal-stopping problem that seemed to demand a backward sweep, and so to be off-limits to forward simulation, is solved by forward simulation after all—because a single least-squares regression through the cross-section of paths recovers the one backward-looking object, the continuation value, that the whole problem turns on. Monte Carlo runs forward, early exercise reasons backward, and regression is the bridge.

A Optimal stopping and the Snell envelope

The Bellman recursion (4) is the discrete-time face of a classical object, the *Snell envelope*. Given the discounted payoff process $\tilde{g}_i = e^{-rt_i} g(S_{t_i})$, the Snell envelope U_i is the smallest supermartingale dominating it, defined backward by

$$U_M = \tilde{g}_M, \qquad U_i = \max\{\, \tilde{g}_i,\ \mathbb{E}^{\mathbb{Q}}[\, U_{i+1} \mid \mathcal{F}_{t_i}\,]\,\}. \tag{11}$$

Undiscounting, $U_i = e^{-rt_i} V_{t_i}$, which is exactly (4) with $C_{t_i}(S_{t_i}) = e^{rt_i}\mathbb{E}^{\mathbb{Q}}[U_{i+1} \mid \mathcal{F}_{t_i}]$. The theory of optimal stopping guarantees that the policy "stop the first time $\tilde{g}_i \geq \mathbb{E}^{\mathbb{Q}}[U_{i+1} \mid \mathcal{F}_{t_i}]$"—exercise as soon as immediate payoff reaches continuation value—attains the supremum in (2). So the optimal stopping time is

$$\tau^* = \min\{\, t_i : g(S_{t_i}) \geq C_{t_i}(S_{t_i})\,\}, \tag{12}$$

the first crossing of payoff over continuation. This is precisely the rule the algorithm applies path by path; the boundary of Figure 1 is the set $\{x : g(x) = C_{t_i}(x)\}$ traced out over t_i. Everything in the note is an estimate of the one conditional expectation $\mathbb{E}^{\mathbb{Q}}[U_{i+1} \mid \mathcal{F}_{t_i}]$ inside (11).

B Why the regression *is* the conditional expectation

The move in §4—"regress the future payoff on the current state to get the continuation value"—is not a heuristic; it is exact in the limit. The reason is a clean fact from probability: the conditional expectation is the *best least-squares predictor*. Among all (square-integrable) functions h of the current state $X = S_{t_i}$, the one minimising the mean squared error to the future payoff Y is the conditional expectation itself,

$$\mathbb{E}\big[\,(Y - h(X))^2\,\big] \text{ is minimised over } h \text{ by } h^*(x) = \mathbb{E}[\,Y \mid X = x\,] = C_{t_i}(x). \tag{13}$$

In the language of Hilbert spaces, $\mathbb{E}[Y \mid X]$ is the orthogonal projection of Y onto the subspace of functions of X: the part of the future payoff that is *predictable* from today's state. Least squares is how one computes a projection. The regression (5) simply restricts that projection to the finite-dimensional subspace spanned by the chosen basis $\{\psi_k\}$—it finds the best predictor of the form $\sum_k \beta_k \psi_k(X)$. As the basis is enriched toward a complete system and the number of paths grows, the fitted $\widehat{C}$ converges to the true continuation value C. The two approximations are exactly the method's two error sources from §7: a *finite basis* (the projection is onto a subspace, not the whole space) and *finite paths* (the projection is estimated from samples). This is why "run a regression" is the right thing to do, and not merely a convenient one.

C Computing the least-squares fit in practice

Equation (5) is written as a minimisation, but in code it is one linear-algebra call. Here is what that call actually does.

Fix a date t_i and let $\mathcal{I}$ be its in-the-money paths, $n = |\mathcal{I}|$. Stack the basis values into an $n \times L$ **design matrix** Ψ and the realised discounted payoffs into a length-n vector $\mathbf{y}$,

$$\Psi_{jk} = \psi_k(X_j), \qquad \mathbf{y}_j = Y_j, \qquad j \in \mathcal{I},$$

so each row is one path's basis values and each column is one basis function evaluated across all the paths. The objective (5) is then the ordinary linear least-squares problem $\min_\beta \|\mathbf{y} - \Psi\beta\|_2^2$, whose minimiser solves the **normal equations**

$$(\Psi^\top \Psi)\,\widehat{\beta} \;=\; \Psi^\top \mathbf{y}, \qquad \text{equivalently} \qquad \widehat{\beta} \;=\; (\Psi^\top \Psi)^{-1} \Psi^\top \mathbf{y}. \tag{14}$$

The fit is cheap. Although n may be tens of thousands of paths, $\Psi^\top\Psi$ is only $L \times L$—a 4×4 matrix for a cubic basis. Forming it is one pass over the paths ($O(nL^2)$) and solving it is $O(L^3)$, negligible either way. It is the *same* tiny solve whether you threw 10^3 or 10^6 paths at it; the simulation, not the regression, dominates the run time. And it is run once per exercise date, $M - 1$ times in all.

But do not invert the normal equations. Forming $\Psi^\top\Psi$ explicitly squares the condition number of Ψ, and a raw monomial basis $(1, S, S^2, \dots)$ on prices in the tens is already ill-conditioned: over a narrow range of S the columns S^2 and S^3 are nearly proportional, so $\Psi^\top\Psi$ sits close to singular and inverting it amplifies sampling noise into wild coefficients. Every numerical library therefore solves $\min_\beta \|\mathbf{y} - \Psi\beta\|$ *directly* from Ψ, via a QR or SVD factorisation that never squares the conditioning. This is exactly what `numpy.linalg.lstsq` does—and what `polyfit` in Appendix D calls underneath. The single line `coef = P.polyfit(...)` *is* the solve of (14), done stably; you are not expected to assemble Ψ by hand.

Two habits that keep it clean. **Centre and scale** the regressor (fit on $(S - \bar{S})/\mathrm{sd}(S)$ rather than raw S), or use a basis whose columns are already well separated—Chebyshev or Hermite polynomials, or the *weighted Laguerre* polynomials of the original Longstaff–Schwartz paper. These span the same low-order space as plain powers but with near-orthogonal columns, so the fit is numerically clean and the coefficients are interpretable. By "only the crossing matters" (§4) the price itself barely moves, but a conditioned basis removes any doubt that a wobbly coefficient is an artefact of the solver rather than of the data.

D A forty-line implementation

The pricer below reproduces the eight-path example (0.1144) and, with the realistic parameters of §1, the American-put value of about 4.48. It is the whole method: simulate, sweep backward, regress, compare, average.

```
import numpy as np
from numpy.polynomial import polynomial as P

def lsm_american_put(S, K, r, T, deg=3):
    # S: array (n_paths, n_steps+1) of simulated prices; column 0 = today.
    n_steps = S.shape[1] - 1
    dt   = T / n_steps
    disc = np.exp(-r * dt)
    cashflow = np.maximum(K - S[:, -1], 0.0)       # exercise at maturity if ITM
    for t in range(n_steps - 1, 0, -1):            # sweep backward over dates
        cashflow *= disc                           # discount carried cashflow to t
        exercise = np.maximum(K - S[:, t], 0.0)
        itm  = exercise > 0                         # regress in-the-money only
        coef = P.polyfit(S[itm, t], cashflow[itm], deg)
        cont = P.polyval(S[:, t], coef)            # estimated continuation value
        do   = itm & (exercise > cont)             # exercise where it beats holding
        cashflow = np.where(do, exercise, cashflow)  # overwrite w/ immediate payoff
    return cashflow.mean() * disc                  # discount t=1 -> t=0, average

def gbm_paths(S0, r, sigma, T, n_steps, n_paths, seed=0):
    rng = np.random.default_rng(seed)
    dt  = T / n_steps
    z   = rng.standard_normal((n_paths, n_steps))
    z   = np.vstack([z, -z])                       # antithetic variates
    drift = (r - 0.5 * sigma**2) * dt
    incr  = drift + sigma * np.sqrt(dt) * z
    logS  = np.log(S0) + np.cumsum(incr, axis=1)
```

```
    S = np.empty((z.shape[0], n_steps + 1))
    S[:, 0] = S0; S[:, 1:] = np.exp(logS)
    return S

S = gbm_paths(36.0, 0.06, 0.20, 1.0, n_steps=50, n_paths=40000)
print(lsm_american_put(S, K=40.0, r=0.06, T=1.0))  # ~ 4.48
```

For a production figure one would, per §7, fit the coefficients on one set of paths and evaluate the policy on a second, independent set; the change is mechanical and the eight extra lines are left as the obvious exercise.

References

[1] F. A. Longstaff and E. S. Schwartz. Valuing American options by simulation: a simple least-squares approach. *Review of Financial Studies*, 14(1):113–147, 2001.

[2] J. N. Tsitsiklis and B. Van Roy. Regression methods for pricing complex American-style options. *IEEE Transactions on Neural Networks*, 12(4):694–703, 2001.

[3] J. F. Carrière. Valuation of the early-exercise price for options using simulations and nonparametric regression. *Insurance: Mathematics and Economics*, 19(1):19–30, 1996.

[4] L. C. G. Rogers. Monte Carlo valuation of American options. *Mathematical Finance*, 12(3):271–286, 2002.

[5] M. B. Haugh and L. Kogan. Pricing American options: a duality approach. *Operations Research*, 52(2):258–270, 2004.

[6] L. Andersen and M. Broadie. Primal–dual simulation algorithm for pricing multidimensional American options. *Management Science*, 50(9):1222–1234, 2004.

[7] E. Clément, D. Lamberton, and P. Protter. An analysis of a least squares regression method for American option pricing. *Finance and Stochastics*, 6(4):449–471, 2002.

[8] P. Glasserman. *Monte Carlo Methods in Financial Engineering*. Springer, 2004.