Course Project Report

Discrete Models for Pricing Asian and Exotic Options

1. Project Aim

The project aims to understand, implement, and analyze discrete-time numerical methods for pricing exotic options, focusing on Asian options (both European and American-style). The core methodology is Dynamic Programming (DP) applied on a discretized state space, benchmarked against Monte Carlo (MC) simulation. The goal is to produce a solver that is mathematically grounded, computationally efficient, and validated with clear numerical experiments. We also aim to visualize results such as convergence, exercise frontiers, and sensitivity to parameters.

2. Theoretical Background

2.1 Basics of Option Pricing

- European Option: Exercise only at maturity.
- American Option: Exercise anytime before maturity.

Valuation principle:

- Under risk-neutral measure, option price = discounted expected payoff.

$$V_0 = e^{-rT}\,\mathbb{E}^{\mathbb{Q}}[\text{Payoff}(S_T)]$$

with r = risk-free rate.

2.2 Exotic Options – Asian Options

$$A_T = \frac{1}{N}\sum_{i=1}^{N} S_{t_i}$$

Asian option payoff depends on the average price over its life.
Arithmetic average: A_T
Payoffs:
- Asian Call = max(A_T - K, 0)
- Asian Put = max(K - A_T, 0)

Asian options reduce price manipulation risk and are used in commodities, FX, and energy markets.

2.3 European vs American Asian Options

- European Asian: only exercisable at maturity.
- American Asian: can be exercised early, payoff depends on average up to that time.

Example payoff for Put: max(K - A_t, 0), with A_t the running average up to time t.

2.4 Dynamic Programming Formulation

5. Dynamic Programming Formulation

Let:

- $t = 0, 1, \ldots, N$ (discrete time steps)
- State = (S_t, A_t)
- Payoff at exercise = $\Phi(S_t, A_t)$

Bellman Equation:

At time step t:

$$V(t, S_t, A_t) = \max \left\{ \Phi(S_t, A_t),\ e^{-r\Delta t}\, \mathbb{E}[V(t+1, S_{t+1}, A_{t+1}) \mid S_t, A_t] \right\}$$

- First term = immediate exercise payoff.
- Second term = continuation value (expected discounted future).

Boundary condition:

$$V(N, S_N, A_N) = \Phi(S_N, A_N)$$

Transition Law:

- $S_{t+1} = S_t e^{(r - \frac{1}{2}\sigma^2)\Delta t + \sigma\sqrt{\Delta t} Z}$, with $Z \sim N(0,1)$.
- $A_{t+1} = \frac{tA_t + S_{t+1}}{t+1}$ (running arithmetic average).

6. Properties of the Value Function

The DP method proves (from your paper):

- **Monotonicity**: Value is increasing in S for calls, decreasing for puts.
- **Convexity**: Option value is convex in S.
- **Early Exercise Frontier**: Exists a boundary in (t, S, A)-space that separates "exercise now" from "continue".
- **Convergence**: As discretization (state grid + time steps) → finer, DP converges to true option value.

2.5 Grids and Discretization

1. Price grid for S (the underlying)

We need to cover *all possible future prices of the asset* S_t up to maturity T.

👉 Problem: asset prices can wander arbitrarily far in the Black–Scholes model, but we cannot make an infinite grid.

👉 Solution: cover a **reasonable range** that contains ~99.7% of the probability mass (3–4 standard deviations in log-space).

Why log-space?

- Asset price follows **lognormal** distribution.
- $\ln S_T \sim \mathcal{N}(\ln S_0 + (r - \frac{1}{2}\sigma^2)T,\ \sigma^2 T)$.
- So most outcomes lie within $\pm 3\sigma\sqrt{T}$ of the mean in log-space.

How to set range:

$$S_{\min} = S_0 \exp(-k\sigma\sqrt{T}), \quad S_{\max} = S_0 \exp(+k\sigma\sqrt{T}),$$

with $k \approx 3$ or 4.

- Example: if $S_0 = 100, \sigma = 0.2, T = 1$, then $\sigma\sqrt{T} = 0.2$.
  - With $k = 3$: $S_{\min} \approx 100\exp(-0.6) \approx 55$, $S_{\max} \approx 100\exp(0.6) \approx 165$.
  - With $k = 4$: range = $[45, 220]$.

This is wide enough that probabilities outside are negligible.

Number of grid points:

- N_S: how finely we discretize between $S_{\min}$ and $S_{\max}$.
- Rule of thumb: 80–200 grid points is good for a student project.
- If you pick $N_S = 121$, you will have **121 log-spaced points** between $S_{\min}$ and $S_{\max}$.

2. Average grid for A (running average)

Now we need a grid for the arithmetic average A_t.

Range for A:

- Since A_t is the *average of prices seen so far*, it will always lie between the min and max price trajectory.
- So safe choice:

$$A_{\min} \approx S_{\min}, \quad A_{\max} \approx S_{\max}.$$

- Example: using the earlier numbers ($[55, 165]$), you can take $A \in [55, 165]$.

Grid style:

- Use a **linear grid** (equally spaced in arithmetic scale).
- Number of points: $N_A = 60$–160. A good compromise is **101** points.

2.6 Gauss–Hermite Quadrature

3. Quadrature nodes (Gauss–Hermite integration)

At each DP step, you must compute

$$\mathbb{E}[V(S', A') \mid S, A] = \int_{-\infty}^{\infty} V(Se^{\mu+\nu z}, A')\phi(z)\, dz,$$

where $\phi(z)$ is the standard normal density.

This integral can't be solved exactly → approximate using **Gauss–Hermite quadrature**.

How Gauss–Hermite works:

- Approximate the integral by weighted sum over a few fixed nodes:

$$\int_{-\infty}^{\infty} f(z)e^{-z^2}\, dz \approx \sum_{k=1}^{K} w_k f(x_k).$$

- After transformation, this gives:

$$\mathbb{E}[f(Z)] \approx \frac{1}{\sqrt{\pi}} \sum_{k=1}^{K} w_k f(\sqrt{2}\, x_k).$$

- x_k = pre-tabulated Hermite nodes, w_k = weights.

How many nodes K?

- $K = 5$: fast, rough.
- $K = 7$: good balance (what people use in practice).
- $K = 9$: very accurate but slower.

EXAMPLE --->

1. Gauss–Hermite quadrature definition

For K nodes, we have pairs (x_k, w_k) such that

$$\int_{-\infty}^{\infty} f(x)\, e^{-x^2} dx \;\approx\; \sum_{k=1}^{K} w_k f(x_k).$$

The x_k are the roots of the Hermite polynomial $H_K(x)$, and w_k are computed from formulas (but usually we take them from a library).

---

2. Explicit case: $K = 3$

The 3 nodes and weights (to many decimals):

- $x_1 = -1.2247448714$, $w_1 = 0.2954089752$
- $x_2 = 0.0000000000$, $w_2 = 1.1816359006$
- $x_3 = +1.2247448714$, $w_3 = 0.2954089752$

---

3. Apply to $\mathbb{E}[Z^2]$

We want:

$$\mathbb{E}[Z^2] \;\approx\; \frac{1}{\sqrt{\pi}} \sum_{k=1}^{K} w_k \,(\sqrt{2}\, x_k)^2.$$

Plugging in:

- For $x_1 = -1.2247$, $(\sqrt{2}x_1)^2 \approx 2 \times (1.2247)^2 \approx 2 \times 1.5 = 3.0$.
  Contribution = $w_1 \times 3.0 \approx 0.2954 \times 3 = 0.886$.
- For $x_2 = 0$, value = 0. Contribution = 0.
- For $x_3 = +1.2247$, same as first = 0.886.

So sum = $0.886 + 0.886 = 1.772$.

Now multiply by $1/\sqrt{\pi} \approx 0.56419$:

$$1.772 \times 0.56419 \approx 0.9999 \approx 1.$$

🎉 Exactly 1 (up to numerical rounding).

3. How to get them in practice

You usually don't compute them by hand (roots of polynomials + weight formula are messy for large K). Instead, numerical libraries do it for you:

- **Python**: `numpy.polynomial.hermite.hermgauss(K)`
- **MATLAB**: `hermiteH` + custom routine
- **C++**: Boost's `quadrature::gauss_hermite`

These return arrays of (x_k, w_k) ready to use.

3. Implementation Plan

Implementation—step by step

1) Choose the meshes (inputs you control)

- **Time mesh**: indices $n = 0, \dots, N$ with $\Delta t = T/N$. Include all exercise dates; include monitoring dates (where averages update). (You can merge the two: just mark some n as "monitoring", some as "exercise"—I
- **Price grid for S** (non-uniform helps):
  - Work in log-space for stability. Let $x = \ln S$. Make a uniform grid for $x \in [\ln S_{\min}, \ln S_{\max}]$.
  - A practical choice: $S_{\min} = S_0 \exp(-k\sigma\sqrt{T})$, $S_{\max} = S_0 \exp(k\sigma\sqrt{T})$ with $k \in [3, 4]$.
  - Grid size: $N_S \in [80, 200]$ for a quick project; start with $N_S = 121$.
- **Average grid for A**:
  - Keep it linear (not log). Use a conservative range like $A \in [A_{\min}, A_{\max}]$ with $A_{\min} \approx S_{\min}$ and $A_{\max} \approx S_{\max}$.
  - Grid size: $N_A \in [60, 160]$; start with $N_A = 101$.
- **Quadrature nodes** (for the 1D normal expectation): Gauss–Hermite with $K \in \{5, 7, 9\}$ nodes is usually fine.

Further steps -→

1. What we're computing

At each discrete time step n in the DP recursion, we want the option value function:

$$V_n(S, A) \quad \text{for all grid points } (S_i, A_j).$$

- S = current asset price (grid of size N_S)
- A = current running average (grid of size N_A)
- So the **state space grid** has $N_S \times N_A$ points.

That's why V_n is stored as a **2D array** with shape:

$$V_n[i, j] \quad \leftrightarrow \quad V(t_n, S_i, A_j).$$

---

2. Memory-saving trick

We must step backwards in time:

$$V_n(S, A) \quad \leftarrow \quad \mathbb{E}[V_{n+1}(S', A')].$$

That means **to compute V_n**, we only need V_{n+1}.
We never need to store *all* V values for every time step.

👉 Therefore:

- Keep two arrays only:
  - `V_next` = option values at time $n + 1$ (future layer)
  - `V_now` = option values we are computing at time n (current layer)
- After finishing a step, set `V_next = V_now` (swap references).

This cuts memory from $O(N \cdot N_S \cdot N_A)$ to just $O(N_S \cdot N_A)$.

3. Arrays for grids

We also need to know **what asset price and average each index corresponds to**:

- `S_grid[i]` : an array of length N_S. Contains log-spaced asset prices S_i between $S_{\min}$ and $S_{\max}$.
- `A_grid[j]` : an array of length N_A. Contains linearly spaced averages between $A_{\min}$ and $A_{\max}$.

So if you want the value at state (i, j): it's

$$S = S_{\text{grid}}[i], \quad A = A_{\text{grid}}[j], \quad V = V[i, j].$$

---

4. Precomputed constants: μ and ν

When simulating the one-step transition of S, we use the Black–Scholes lognormal evolution:

$$S_{t+\Delta t} = S_t \cdot \exp\left((r - q - \tfrac{1}{2}\sigma^2)\Delta t + \sigma\sqrt{\Delta t}\, Z\right).$$

We define:

- $\mu = (r - q - \frac{1}{2}\sigma^2)\Delta t$
- $\nu = \sigma\sqrt{\Delta t}$

So that

$$S' = S \cdot \exp(\mu + \nu Z).$$

These are fixed for a given timestep, so we compute them once and reuse them for all grid evaluations at that time.

DP Step -→

5. Putting it together in practice

Imagine we're at timestep n:

- **Inputs:**
  - `V_next` = 2D array of size `[N_S][N_A]` (values at time t_{n+1})
  - `S_grid`, `A_grid` = arrays defining grids
  - μ, ν constants for this step
- **Loop:**

  For each grid node (i, j):
  1. Take $S = S_{\text{grid}}[i]$, $A = A_{\text{grid}}[j]$.
  2. Use Gauss–Hermite nodes to compute expectation over possible S'.
  3. Interpolate $V_{n+1}(S', A')$ using `V_next`.
  4. Discount and take max with payoff if it's an exercise date.
  5. Store result in `V_now[i,j]`.
- **After finishing loop:**

  Swap arrays: `V_next = V_now`.

EXPLAINATION FOR:

2. Use Gauss–Hermite nodes to compute expectation over possible S'.
3. Interpolate $V_{n+1}(S', A')$ using `V_next`.

(By an example:)

Setup (tiny toy)

- Parameters: $r = 5\%$, $q = 0$, $\sigma = 20\%$, $\Delta t = 1/12$.
- So $\mu = (r - q - \frac{1}{2}\sigma^2)\Delta t = (0.05 - 0.02) \cdot \frac{1}{12} = 0.0025$,
  $\nu = \sigma\sqrt{\Delta t} = 0.2\sqrt{1/12} \approx 0.057735$.
- Current grid node: $S = 100$, $A = 100$.
- Monitoring is ON at this step, and we've already observed $k = 5$ prices before now (so after we draw S', the updated average is $A' = \frac{5A + S'}{6}$).

Grids (very small, just for demo):

- S-grid (log-spaced but I'm writing rounded values): $[90,\ 100,\ 111]$
- A-grid (linear): $[98,\ 100,\ 102]$

You already have `V_next[i,j] = V_{n+1}(S_i, A_j)` from the previous time layer. To illustrate interpolation, I'll assign simple numbers:

- Cell around (100,100):
  $V_{n+1}(100, 100) = 12,\ V_{n+1}(111, 100) = 10,\ V_{n+1}(100, 102) = 13,\ V_{n+1}(111, 102) = 11.$
- Cell around (90,98):
  $V_{n+1}(90, 98) = 15,\ V_{n+1}(100, 98) = 13,\ V_{n+1}(90, 100) = 14,\ V_{n+1}(100, 100) = 12.$

(These are just placeholders so you can see the interpolation arithmetic.)

Step 1 — Gauss–Hermite nodes and weights (K=3)

For $K = 3$, the GH nodes x_k and weights w_k for $\int f(x)e^{-x^2}dx \approx \sum w_k f(x_k)$ are:

- $x = \{-1.224744871,\ 0,\ +1.224744871\}$
- $w = \{0.295408975,\ 1.181635901,\ 0.295408975\}$

To turn this into $\mathbb{E}[g(Z)]$ with $Z \sim \mathcal{N}(0, 1)$, use:

$$\mathbb{E}[g(Z)] \approx \frac{1}{\sqrt{\pi}} \sum_{k=1}^{K} w_k\, g(\sqrt{2}\, x_k).$$

So we'll evaluate at $Z_k = \sqrt{2}\, x_k = \{-1.7320508,\ 0,\ +1.7320508\}$.

Step 2 — Map each GH node to a candidate S' and A'

Use $S' = S \cdot \exp(\mu + \nu Z_k)$.

Here $\nu Z_k = 0.057735 \times 1.7320508 \approx 0.1000$. So:

- For $Z = +1.732$: $\mu + \nu Z \approx 0.0025 + 0.1000 = 0.1025$
  $S'_+ = 100 \cdot e^{0.1025} \approx 110.78$
- For $Z = 0$: $\mu + \nu Z = 0.0025$
  $S'_0 = 100 \cdot e^{0.0025} \approx 100.25$
- For $Z = -1.732$: $\mu + \nu Z \approx 0.0025 - 0.1000 = -0.0975$
  $S'_- = 100 \cdot e^{-0.0975} \approx 90.70$

Update the average (monitoring on; $k = 5$ prior samples):

$$A' = \frac{5A + S'}{6}.$$

So:

- $A'_+ = (500 + 110.78)/6 \approx 101.80$
- $A'_0 = (500 + 100.25)/6 \approx 100.04$
- $A'_- = (500 + 90.70)/6 \approx 98.45$

Step 3 — Bilinear interpolation of $V_{n+1}(S', A')$

We need V_{n+1} at off-grid (S', A'). Find surrounding grid points and blend.

Case 1: $Z = +1.732 \to (S', A') \approx (110.78,\ 101.80)$

Brackets:

- in S: between 100 (index i_0) and 111 (index i_1)
  $\alpha = \frac{110.78-100}{111-100} \approx \frac{10.78}{11} \approx 0.98$
- in A: between 100 (index j_0) and 102 (index j_1)
  $\beta = \frac{101.80-100}{2} \approx 0.90$

Corners (from our toy table):
$v_{00} = V(100, 100) = 12,\ v_{10} = V(111, 100) = 10,\ v_{01} = V(100, 102) = 13,\ v_{11} = V(111, 102) = 11.$

Bilinear:

$$V' = (1-\alpha)(1-\beta)v_{00} + \alpha(1-\beta)v_{10} + (1-\alpha)\beta v_{01} + \alpha\beta v_{11}.$$

Plugging $\alpha \approx 0.98,\ \beta \approx 0.898$ gives $V'_+ \approx 10.94$.

Case 2: $Z = 0 \to (S', A') \approx (100.25,\ 100.04)$

Brackets:

- S : between 100 and 111, $\alpha = \frac{0.25}{11} \approx 0.0227$
- A : between 100 and 102, $\beta = \frac{0.04}{2} \approx 0.0209$

Same cell corners as above. Bilinear gives $V'_0 \approx 11.98$.

Case 3: $Z = -1.732 \to (S', A') \approx (90.70,\ 98.45)$

Brackets:

- S : between 90 and 100, $\alpha = \frac{90.70-90}{10} = 0.07$
- A : between 98 and 100, $\beta = \frac{98.45-98}{2} = 0.225$

Use corners of the lower-left cell we defined:
$v_{00} = V(90, 98) = 15,\ v_{10} = V(100, 98) = 13,\ v_{01} = V(90, 100) = 14,\ v_{11} = V(100, 100) = 12.$

Bilinear gives $V'_- \approx 14.64$.

Step 4 — Form the GH expectation (continuation value)

Compute

$$\mathbb{E}[V_{n+1}(S', A')] \approx \frac{1}{\sqrt{\pi}} \left(w_1 V'_- + w_2 V'_0 + w_3 V'_+ \right).$$

Plug numbers:

- $w_1 = w_3 = 0.295408975,\ w_2 = 1.181635901$
- $V'_- \approx 14.64,\ V'_0 \approx 11.98,\ V'_+ \approx 10.94$

Weighted sum $\approx 0.2954 \cdot 14.64 + 1.1816 \cdot 11.98 + 0.2954 \cdot 10.94 \approx 21.713$.

Multiply by $1/\sqrt{\pi} \approx 0.56419$:
$\mathbb{E}[\cdot] \approx 12.251$.

Finally discount one step:

$$\text{Continuation } C = e^{-r\Delta t} \times 12.251 \approx e^{-0.05/12} \times 12.251 \approx 0.9958 \times 12.251 \approx \mathbf{12.20}.$$

That's your **continuation value** at the current node $(S = 100, A = 100)$.
If this is an exercise date, you take

$$V_n(100, 100) = \max\{\text{payoff}(100, 100),\ 12.20\};$$

otherwise $V_n = 12.20$.

4. Potential Results

A) Validation: European Asian (no early exercise)

Goal: Prove your solver is correct before adding early exercise.

- Fix baseline: $S_0 = 100,\ K = 100,\ r = 5\%,\ \sigma = 20\%,\ T = 1$.
- Monitor at $M \in \{12, 24, 52\}$ evenly spaced dates.
- **DP (European):** same engine, but set `is_exercise=False` everywhere.
- **MC baseline:** 10^5 paths with antithetic variates → price $\pm$ 95% CI.

Show:

- A small **table** for each M: DP price, MC mean, MC CI half-width, |DP − MC|.
- A **bar+errorbar plot**: MC with CI and a dot for DP.

Expected: DP sits inside MC confidence intervals as you refine the DP grid.

Monte Carlo -→

What you simulate

Under risk-neutral GBM (with dividend yield q, set $q = 0$ if none):

$$S_{t_{k+1}} = S_{t_k} \exp\left((r - q - \tfrac{1}{2}\sigma^2)\Delta t + \sigma\sqrt{\Delta t}\, Z_k\right), \quad Z_k \sim \mathcal{N}(0,1) \text{ i.i.d.}$$

- Total time T, monitor at M equally spaced dates, $\Delta t = T/M$.
- Arithmetic average on a path:

$$A_T = \frac{1}{M}\sum_{k=1}^{M} S_{t_k}.$$

- Payoff (call): $\max(A_T - K, 0)$ (put is $\max(K - A_T, 0)$).
- Discount to $t = 0$: e^{-rT}.

Plain MC estimator

Simulate P independent paths:

$$\widehat{V} = e^{-rT}\frac{1}{P}\sum_{p=1}^{P} \text{Payoff}^{(p)}.$$

- Sample variance of discounted payoffs:

$$\widehat{\text{Var}} = \frac{1}{P-1}\sum_{p=1}^{P}\left(e^{-rT}\text{Payoff}^{(p)} - \widehat{V}\right)^2.$$

- Standard error (SE): $\text{SE} = \sqrt{\widehat{\text{Var}}/P}$.
- 95% CI: $\widehat{V} \pm 1.96\,\text{SE}$.

B) Convergence studies (show your numerics are solid)

Pick one case (e.g., European Asian, $M = 24$) and do three mini-sweeps:

1. **State grid**: fix $N_A = 101, K = 7, N = 60$. Vary $N_S \in \{81, 101, 121, 161\}$.
2. **Average grid**: fix $N_S = 121$. Vary $N_A \in \{61, 81, 101, 121\}$.
3. **Quadrature nodes**: fix $N_S = 121, N_A = 101$. Vary $K \in \{5, 7, 9\}$.
4. **Time steps**: vary $N \in \{40, 60, 80, 100\}$ (monitor every step, or map monitors to a subset).

Show:

- **4 small line plots** (y-axis = |DP − MC| or variance-reduced DP diff, x-axis = each refinement).
- A final **runtime vs absolute error** chart (log–log makes you look pro).

Expected: Diminishing error as you refine (diminishing returns at the high end).

C) Exercise frontier (make this your hero figure)

Take a **Bermudan Asian put** (exercise monthly) with the baseline params.

Compute: For a few times t (e.g., $t/T = 0.25, 0.5, 0.75$), output a binary mask over the grid: `exercise_now = 1{ payoff(S,A) >= continuation(S,A) }`.

Show:

- **3 heatmaps** (or contour plots) in (S, A) space, colored Exercise vs Continue.
- A **thin white contour** for the boundary (where payoff = continuation).

Expected:

- Monotonic boundary: for puts, exercise region grows as S drops or A increases (depends on payoff structure).
- As time advances, boundary shifts (time value decays → more exercise).

One sentence claim: "We empirically verify the existence and monotonicity of the optimal exercise frontier for American-Asian puts."

D) Early-exercise premium

For the same parameters, compute:

$$\text{Premium} = V^{\text{Berm/Am}} - V^{\text{Euro}}$$

Show:

- **Line plot** of premium vs monitoring frequency M (e.g., 12, 24, 52).
- **Line plot** of premium vs σ (e.g., 10%...50%).
- **Plot** of premium vs K (deep ITM/ATM/OTM).

Expected: Premium larger when the put is ITM and when σ is moderate–high (more opportunities to optimally stop).

E) Sensitivity / Comparative statics

Hold everything fixed; vary one parameter at a time. Suggested grids:

- $\sigma \in \{0.10, 0.20, 0.30, 0.40, 0.50\}$
- $r \in \{0.00, 0.02, 0.05, 0.08\}$
- $K \in \{80, 90, 100, 110, 120\}$
- Monitoring frequency $M \in \{12, 24, 52\}$

Show:

- **4 tidy line charts**—very readable, labeled axes, units, legends.

Expected:

- Put price ↑ with K, ↑ with σ, ambiguous with r (discounting vs drift effects).
- More monitoring dates changes the averaging → usually lowers variance of A_T, affecting value subtly.

F) Runtime vs accuracy (practitioner's plot)

Pick a single case (e.g., Bermudan Asian put at baseline).
Sweep (N_S, N_A, K, N) in a small grid; record **wall-clock time** and **error** (vs a "reference" price computed on a very fine grid).

Show:

- **Scatter**: x = runtime (s), y = |price − reference|.
- Annotate a **sweet-spot** config (e.g., $N_S = 121, N_A = 101, K = 7, N = 60$) that is <1s and within your chosen tolerance.

Expected: clear frontier—after a point, more grid points cost a lot for small gains.