Goldman Sachs

Goldman Sachs India Hackathon

Quant Division

Sidhant Thalor
Undergraduate in Department of Economics
IIT Kanpur

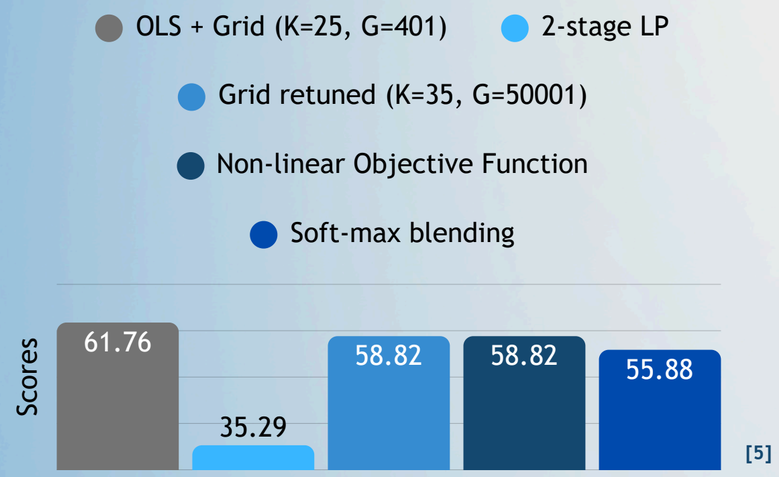
Optimal Hedging Strategy

Hedging is a risk management strategy used to offset potential losses in one asset by taking an opposing position in a related asset.

Portfolio Hedging by selecting K stocks that minimize portfolio VaR while keeping Total Capital Cost low [1]

OLS estimates β_0 and β_1 in $Y=\beta_0+\beta_1X+\varepsilon$ by minimising the sum of squared residuals

Grid Search converts w0 into integer lot sizes while explicitly trading off tail risk ($VaR_{95}(P+Rw)$) and capital cost ($\Sigma_i|w_i|*C_i$)



[5]

Ordinary Least Squares with Grid Search Approach

1. Compute Pearson correlations between each stock's returns and the unhedged P&L
2. ***Optimization Objective Function***
   $\Sigma^n_{i=1} (R_i \cdot w + P_i)^2 = || Rw + P ||^2_2$
3. Objective function stores Pareto-optimal weights
4. Stock liquidity supports ±5× the OLS magnitude
5. ***Time Complexity*** - $O(TK^2 + G(T+K))$
   Space Complexity - $O(TK + T)$
6. **Scalability:** Algorithm accommodates additional assets by adjusting the basket-size and can be extended to support intraday hedging
7. **Shortcomings:** The coarse 401-point grid can skip better integer hedges, and the low cost weight under-penalizes inventory, risking cost-cap breaches

[2]

Assumptions

- Linearity
- Independence of Observations
- Homoscedasticity
- Normality
- No multicollinearity

Alternative Approaches

Mixed Integer Program for VaR+Cost [3]

$\min_{w,y,t} [t+L\Sigma_i c_i*|w_i|]$

- MIP finds integer w minimizing 95%-VaR (t) plus weighted cost
- Eliminates arbitrariness and limited scope of 1-D grid search

Transaction Cost Aware, Volatility-weighted Adaptive K [4]

- Hedge Efficiency Score (s_i)
  $s_i = |corr(R_i, P)| * stddev(R_i) / C_i$
- Ensures we don't waste capital on low-leverage or high-cost names and include enough assets to cover our tail-risk

Sources: [1] [2] [3] [4] [5]

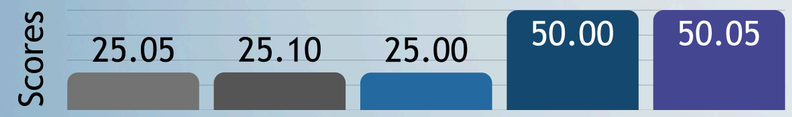
Automated Market Making

Continuously posting two-way limit orders maximising expected spread capture while controlling inventory and adverse-selection risk within tick-grid and size limits

Assumptions

- EMA half-life mapping assumes constant time spacing
- Perfect knowledge of next midprice for volatility update

- Linear inventory skew only
- Depth imbalance, volatility cushion
- Tick-inside hunt, dynamic spread floor
- EMA volatility, flow & depth skews
- Skew multipliers 1.5 ×



Volatility by Exponential Moving Average [EMA][2]

1. Risk Limits and Tuning Multipliers added
   EMA weight (α) = 1/(ema_half_life+1)
2. Volatility Update
3. Trade Processing
4. ***Quote Construction***
   - Book Imbalance $=d_{bid}+d_{ask}/d_{bid}-d_{ask}$
   - mid=(bid1+ask1)/2 ; $\sigma=(ema\_var)^{0.5}$
   - Half Spread $(h)=\max(tick, base\_half, k_{vol}\sigma, k_{inv}|inv|\times tick)$
   - Skew(s)= $-inv_{skew}*inv/max_{inv}+flow_{skew}*flow_{imb}+book_{skew}*book_{imb}$
5. Apply Cooldowns and Safety buffers, Update inventory before each quote
6. ***Time Complexity*** - O(N)
   Space Complexity - O(N)
7. **Scalability:**
   1. Each of the three skews captures a different predictor, reducing multicollinearity
   2. No future-peek, passes any audit
   3. Group-by + index lookup scales linearly

Shortcomings

- EMA half-life mapping assumes constant time spacing
- Perfect knowledge of next midprice for volatility update

Alternative Approach

Linear-Quadratic Regulator (LQR)[1]

- **Control Law:** $u_t = -Kx_t$ (positive means push both bid & ask upward, negative downward)
- **Dynamics** (linearised)
  $$x_{t+1}=x_t-k_f u_t+w_t$$
  k_f - empirical fill-slope
- **Cost function** (quadratic)
  $$J=\Sigma^{\infty}(x_t^{\top}Q.x_t + u_t^{\top}R.u_t)$$
- **Optimal feedback gain**
  $$K=0.5[(1+4k_f^2\lambda_l/\lambda_u)^{0.5}-1)*(1/k_f)$$
- **Time Complexity** - $O(N^3)$
  Space Complexity - $O(N^2)$
- **Improvements:**

1. Inventory variance minimised for a given quote aggressiveness budget
2. No sudden quote jumps; skew changes smoothly with I_t

Sources: [1] [2]

Exotic Option Pricing using Monte Carlo Simulation

A **Knock-out option** grants the holder the right to buy or sell at strike K only if the underlying never breaches a barrier B before maturity

The diffusion process is a continuous time stochastic model used to describe the random evolution of variables [1] [2]

$$dS(t)=\mu(S(t),t)dt+\sigma(S(t),t)dW(t)$$

Assumptions

- All vanilla quotes are invertible, 20% σ fallback if inversion fails
- Underlyings follow risk-neutral log-normal GBM with constant σ per maturity bucket
- Local σ depends solely on maturity, ignoring strike-driven skew

Shortcomings

- Using one σ per maturity ignores strike skew, leads to systematic mispricing of convex barrier payoffs
- Relying on few plain Monte Carlo paths without variance reduction invites discretization bias and high sampling noise

Pricing Pipeline [3]

Black Scholes Analytical Pricing
with fallback C=max(S−K,0) if T=0 or σ=0

$C=S*\Phi(d_1)-K*e^{(-rT)}*\Phi(d_2)$

→ **Implied Volatility Inversion**
Brent's method inverts Black Scholes price to σ_{imp}

$C_{BS}(S,K,T,r,\sigma_{imp})=P_{mkt}$

→ **Local Volatility Surface Calibration**
stabilises pricing by yielding one σ_{loc} per maturity

→ **Monte Carlo Path Simulator**
Cholesky Factor (L) imposes correlation, At each step generate iid normals (Z), correlate and apply GBM update

$S_{t+\Delta t}=S_t\exp((r-0.5\sigma^2)\Delta t+\sigma(LZ)(\Delta t)^{1/2})$

→ **Arithmetic Basket Knock-Out Pricer**
up-and-out barrier test to drop knocked-out scenarios, computes each surviving path's average payoff, discounts them and returns mean as option price

Time Complexity - O(P×S)
Space Complexity - O(P×S×d)
P: No. of paths; **S**: time-steps per path; **d**: assets per path

Alternative Approaches

Dupire Local Volatility Inversion [4]

- Exact vanilla fit across all strikes and maturities
- Volatility moves naturally with the spot S(t), improving barrier and convex payoffs

$$\sigma^2_{Dupire}(K,T)=\frac{(\partial_T C(K,T)+rK\partial_K C(K,T))}{0.5K^2\partial_{KK}C(K,T)}$$

Brownian-Bridge Barrier Correction [5]

- Between grid points t_k and t_{k+1}, approximate the probability of breaching B (barrier level)

$P_{hit}=\exp(-2(B-X_k)(B-X_{k+1})/\sigma^2\,\Delta t)$

Sources: [1] [2] [3] [4] [5]

Types of Hedging

Hedging Type	Primary Use Case	Instruments Used	Goal
Delta Hedging	Options trading	Underlying stocks, options	Neutralize directional exposure (Δ = 0)
Portfolio/Index Hedging	Equity portfolio protection	Index futures, ETF options	Hedge systematic market risk
Cross-Asset Hedging	No direct hedge available	Related futures, ETFs	Hedge using correlated asset
Volatility Hedging	Event risk or implied vol shifts	Options (straddles), VIX futures, variance swaps	Protect against large price swings
Statistical Arbitrage	Quant & HFT strategies	Long/short correlated assets	Hedge market exposure, isolate alpha
Duration Hedging	Fixed income / bond portfolios	Interest rate futures, swaps	Neutralize interest rate sensitivity
Currency Hedging	Foreign investments / FX exposure	FX forwards, futures, options	Lock in exchange rate, reduce currency noise
Gamma/Vega Hedging	Advanced options book management	Options (varied strikes/maturities)	Neutralize convexity & volatility risk

Final Implementation - Pseudo Code

```
p ← read P&L vector
Load metadata, returns; keep last n rows; get cost
for common stocks
corr ← abs(corrwith(returns, p)); sel ← top K by
corr
R, c ← returns[sel], cost[sel]
w0 ← lstsq(R, -p)
best, best_obj ← None, ∞
for s in linspace(-5,5,G):
  w ← round(s * w0)
  if any(w):
    v ← -quantile(p + R@w,1-A) + L * sum(abs(w)*c)
    if v < best_obj: best_obj, best ← v, w
if best is None:
  i ← argmax(abs(w0)); best ← zero; best[i] ←
sign(w0[i])
for i, q in zip(sel, best):
  if q ≠ 0: print(i, q)
```

MIP Implementation - Pseudo Code

Variables:

$w[i] \in \mathbb{Z}$	integer shares for each stock i
$t \in \mathbb{R}$	VaR threshold
$y[j] \in \{0,1\}$	indicator: scenario j breaches VaR

Model: Big-M linearization of $t = VaR_{0.95}$ $\forall\ j \in 1\ldots n$:

$$p[j] + \Sigma_i\, R[j][i] \cdot w[i] \geq -t - M \cdot (1 - y[j])$$

$$p[j] + \Sigma_i\, R[j][i] \cdot w[i] \leq -t + M \cdot y[j]$$

Quantile constraint: at most $(1-\alpha) \cdot n$ breaches

$$\Sigma_j\, y[j] \leq n \cdot (1-\alpha)$$

Objective: minimize VaR + cost penalty

$$\text{minimize} \quad t + L \cdot \Sigma_i\, |w[i]| \cdot c[i]$$

Output: list of (i, w[i]) with $w[i] \neq 0$

Adaptive K Implementation Pseudo Code

```
Compute for each stock i:
 corr[i] = |PearsonCorr(R[i], p)|
 σ[i] = StdDev(R[i])
 score[i] = corr[i]·σ[i] / c[i]

Sort stocks by score ↓
total_vol = ∑ᵢ σ[i]
coverage = 0
budget = 0
selected = []

for each i in sorted list:
 selected.append(i)
 coverage += σ[i]
 budget += σ[i]/total_vol * B
 if coverage/total_vol ≥ τ or budget ≥ B:
 break
```

Experiments Conducted during Hackathon

Non-Linear Objective Functions:
e.g, $J=\alpha \cdot VaR+(1-\alpha) \cdot (1/Cost)$; $\alpha \gg 1$

2-Stage Linear Program
α+(1/1−ß)∑sPsns (CVaR) and among CVaR-optimal, minimise ∑|qi| Ci

ß-smooth Softmax Blending (LSE):
LSEß=(1/ß)*log(∑j e^(ßxj))
ß→∞⇒max, ß→0⇒mean

Linear Quadratic Regulator Implementation

```
# 1. Calibrate
k_f = 0.5        # lots filled per tick of skew (estimate from sample)
lambda_I = 1.0   # risk weight
lambda_u = 0.2   # quote quality weight

# scalar discrete LQR gain
K = ( ( (1 + 4*(k_f**2)*lambda_I/lambda_u )**0.5 - 1 ) / (2*k_f) )

# 2. Inside strategy() replace inventory skew
def lqr_skew(inv):
    return -K * inv                 # optimal tick skew (can clip)

# 3. Build quotes
skew_ticks = lqr_skew(inventory) + flow_skew*flow_imb + book_skew*book_imb
bid = mid - half + skew_ticks * tick
ask = mid + half + skew_ticks * tick
```

Final Implementation - Pseudo Code

```
SETUP
  • load order-book & trade tape (first 3000 rows)
  • constants: tick=0.1, lot=2, invCap=±20
  • params: α (EMA), kVol, kInv, baseHalf, invSk, flowSk, bookSk
  • state = {inv=0, emaMid=?, emaVar=?, flowDeque=[],
activeBid/Ask=None,
            validFrom=None, coolSide=None, coolLeft=0}

FOR each timestamp t
  # 1. handle fills from quote posted at t-1
  IF quote is live AND trades hit it THEN
        inv ← inv ± lot
        cancel filled side, start coolDown(coolSide, coolLeft=3)
        validFrom ← ∞          # suspend until we post again # 2. update
fast signals
  append trade sign (+1 buy, -1 sell) to flowDeque(maxlen=40)
  flowImb ← mean(flowDeque)
  mid ← (bid1 + ask1)/2
  UPDATE emaMid/emaVar with α -> σ ← √emaVar
  bookImb ← (depthBid12 - depthAsk12)/(depthBid12 + depthAsk12)
```

```
 # 3. quote maths
  half ← max(tick, baseHalf, kVol·σ, kInv·|inv|·tick)
  skew ← -invSk·inv/invCap + flowSk·flowImb + bookSk·bookImb
  bid  ← mid - half + skew·tick
  ask  ← mid + half + skew·tick
  IF coolDown active: widen filled side by 2·tick
  CLAMP bid one tick below bestBid1, ask one tick above bestAsk1
  GRID-ROUND both prices; if bid ≥ ask shift apart by one tick

  # 4. risk guards
  IF inv ≥ +20: bid=None
  IF inv ≤ −20: ask=None
  IF t past 95 % of session: quote only side that flattens inventory
  coolLeft ← max(coolLeft-1, 0)  (disable when reaches 0)

  # 5. post quote
  activeBid, activeAsk ← bid, ask
  validFrom ← t + 1
  save {t, bid or "", ask or ""}
END FOR

RETURN quotes → submission.csv
```

Mathematics behind Risk-neutral measure

The risk-neutral measure(Q) is a probability measure under which the discounted price process of any traded asset is a martingale. Under it, the expected discounted payoff of a derivative gives its no-arbitrage price.

- Change of measure: Starting from the real-world measure P, one applies Girsanov's theorem to shift the drift of the asset's SDE from the empirical expected return μ to the risk-free rate r. Concretely, if under P

$$dS(t)=\mu S(t)\,dt+\sigma S(t)\,dW_P(t)$$

  then under Q, the Brownian motion changes to $W_Q(t)$ and the drift becomes r S(t)

- Martingale property: Define the discounted price $e^{-rt}S(t)$. Under Q,

$$d(e^{-rt}S(t))=e^{-rt}\,\sigma(S(t),t)\,dW_Q(t)$$

  which has zero drift, so $E_Q[\,e^{-rt}S(t)\mid F_s\,]=e^{-rs}S(s)$

- Pricing formula: For any derivative with payoff H=H(S(T)) at maturity T,

$$\text{Price at } t=0 = e^{-rT}\,E_Q[H(S(T))]$$

The risk-neutral expectation replaces the need to forecast real-world returns, collapsing all risk premia into the drift r.

Mathematics behind The Diffusion Process

A diffusion process is a continuous-time stochastic model used to describe the random evolution of variables such as asset prices. In our context, each stock price S(t) follows a stochastic differential equation (SDE) of the form:-

$$dS(t) = \mu(S(t),t)\,dt + \sigma(S(t),t)\,dW(t)$$

where:

- Drift term $\mu(S,t)dt$ captures the deterministic trend or expected instantaneous rate of return.
- Diffusion term $\sigma(S,t)\,dW(t)$ injects randomness via $dW(t)$, the increment of a standard Brownian motion (Wiener process).
- $\sigma(S,t)$ is the local volatility, which can depend on both the current state S(t) and time t, allowing for richer dynamics than constant-volatility models.

Key properties of diffusion processes:

- Continuity: Paths are almost surely continuous in t.
- Markov property: Future evolution depends only on the current state S(t), not on the past trajectory.
- Itô calculus: Integrals and differentials follow Itô's rules; for a twice-differentiable function f(S,t),
-

In the local-volatility SDE we use for exotic pricing under the risk-neutral measure, the drift $\mu(S,t)$ is set to $r\,S(t)$, and the diffusion coefficient is $\sigma(S,t)$, giving

$$dS(t)=r\,S(t)\,dt + \sigma(S(t),t)\,dW(t)$$

Final Implementation - Pseudo Code

```
# 1. Black–Scholes call price
FUNCTION bs_call_price(S,K,T,r,σ):
  IF T==0 OR σ==0: RETURN max(S−K,0)
  d1←(ln(S/K)+(r+0.5σ²)T)/(σ√T); d2←d1−σ√T
  RETURN SΦ(d1)−K e^{−rT}Φ(d2)

# 2. Implied-vol inversion via Brent
FUNCTION implied_vol_call(P_mkt,S,K,T,r):
  FIND σ∈[1e−6,5] s.t.
bs_call_price(S,K,T,r,σ)=P_mkt
  IF fails: RETURN 0.2

# 3. Calibrate vols per maturity
implied_vols←{}
FOR EACH (stock,K,T,P_mkt) IN calib_data:

implied_vols[(stock,T)]←implied_vol_call(P_mkt,100
,K,T,0.05)
```

```
# 4. Simulate correlated GBM
FUNCTION simulate_gbm(S0,vols,ρ,T,n_steps,n_paths):
 dt←T/n_steps; L←Cholesky(ρ)
 paths←zeros(n_paths,n_steps+1,len(vols));
paths[:,0]←S0
 FOR t IN 1..n_steps:
 Z←randn(n_paths,len(vols)); dW←Z·Lᵀ√dt
 FOR i IN assets:
 paths[:,t,i]←paths[:,t-1,i]*exp((r−0.5vols[i]²)dt +
vols[i]dW[:,i])
 RETURN paths

# 5. Knock-out basket pricer
FUNCTION price_knockout(type,K,T,B,S0,vols,ρ):
 P←simulate_gbm([S0]*3,vols,ρ,T,252,1000)
 B_paths←mean(P,axis=2)
 alive←ALL(B_paths[:,:-1]<B, axis=1)
 payoff←IF type=="Call" THEN max(B_paths[:,-1]−K,0)
ELSE max(K−B_paths[:,-1],0)
 RETURN mean(exp(−rT)*(payoff*alive))
```

Dupire Local-Volatility Inversion Implementation

```
# 1. Fit smooth spline or bicubic to C_market[K,T].
# 2. Precompute derivatives ∂_T C, ∂_K C, ∂_KK C on grid.
def local_vol(K,T):
    dC_dT = d_spline_T(K,T)
    dC_dK = d_spline_K(K,T)
    d2C_dKK= d2_spline_KK(K,T)
    return sqrt((dC_dT + r*K*dC_dK) / (0.5*K*K*d2C_dKK))

# 3. In GBM simulator, replace vols[i] with local_vol(S_prev, t_prev).
```

Brownian-Bridge Barrier Correction Implementation

```
for each path n:
  knocked = False
  for k in 0..n_steps-1:
    if X[k]>=B or X[k+1]>=B:
      knocked = True; break
    # Brownian-bridge check
    p_hit = exp(-2*(B-X[k])*(B-X[k+1])/(vol**2*dt))
    if rand() < p_hit:
      knocked = True; break
```