GRID TRADING SERIES · หนังสือเล่มที่ 8

Grid Trading Mastery

จาก Grid พื้นฐาน สู่ Grid ระดับ Quant

ครอบคลุมทุกมิติ: Close System, Zone Tricks, Soft Martingale, Bloating Pattern, Volatility-Adaptive Sizing, Regime Detection, Pair Grid, Grid Snowball และ Grid-Framework + Indicator Execution — ประยุกต์ความรู้จากทุกเล่มในซีรีส์

16 ตอน + ภาคผนวก · ≈40 ชั่วโมงเนื้อหา · ตัวอย่าง BTC/USDT Bybit ตลอดเล่ม

ปรัชญาหลักก่อนเริ่มอ่าน

Grid คือ Grid — Enhancement ไม่ใช่ Magic

Grid trading framework มี alpha จริง: ดักจับ mean-reversion + บังคับวินัยการซื้อขาย — แต่ทุก enhancement (indicator, zone trick, martingale) ช่วยปรับ **คุณภาพของ equity curve** มากกว่าเพิ่ม total return อย่างมีนัย

แม้โยนหัวก้อยเพื่อเลือกว่าจะเปิด order หรือไม่ — Grid framework ก็ยังสร้างกำไรได้ เพราะ alpha มาจากโครง grid เอง Indicator ช่วยลด DD และทำให้ Sharpe ดีขึ้น — ไม่ใช่สร้าง alpha ใหม่จากอากาศ

สารบัญรวม

รากฐาน (Foundation)

Part 0	**รู้จัก Grid Trading + ปรัชญา** — กลไก, ประเภท, เปรียบเทียบ, ปรัชญา	~40 นาที	พื้นฐาน
Part 1A	**Close System + Zone Tricks + Soft Martingale** — ระบบปิด, Zone Waterfall, Bell Curve, Front-Loading, r-parameter	~2.5 ชม.	กลาง
Part 1B	**Buy-and-Sell Grid (Bidirectional)** — สองทาง, Long/Short Layer, Capital Allocation ทั้งสองฝั่ง	~2.5 ชม.	กลาง
Part 1C	**Grid Hedge (Spot-Perp)** — ป้องกัน inventory risk ด้วย perp short, delta-neutral grid	~2.25 ชม.	กลาง

Zone & Sizing (จัดการ Zone และขนาด)

Part 2	**Bloating Patterns + Step Pyramid** — บวมบน/บวมล่าง/บวมกลาง, dynamic sizing, step vs size tradeoff	~2 ชม.	กลาง
Part 3	**Zone Sizing — Vol/ATR/Bootstrap/Donchian/Keltner** — Volatility-adaptive step, Monte Carlo + Block Bootstrap, channel-based zone	~2.5 ชม.	กลาง
Part 3B	**Indicator Filter + Order Accumulation** — MA/EMA/ADX filter, รวบ order ที่พลาดไป, ลดต้นทุน DD	~1.75 ชม.	กลาง

Regime & Risk (ระบอบตลาดและความเสี่ยง)

Part 4	**Regime Detection — เมื่อไหร่ Grid ทำงาน เมื่อไหร่หยุด** — Hurst, ADX, Bollinger, HMM, Regime-Switching Grid	~2.25 ชม.	สูง
Part 5	**Sizing & Risk — Kelly, Position Limit, Drawdown Control** — Kelly criterion บน grid, fractional Kelly, max notional per level	~2.5 ชม.	กลาง

Advanced Strategy (กลยุทธ์ขั้นสูง)

Part 6	**Pair Grid + Relative Value Switching** — Cointegration-based pair, Inverse Correlation Grid, BTC↔ETH grid	~2.25 ชม.	สูง
Part 6B	**Grid Snowball + DCA Stack** — ทบผลกำไร 3 แบบ, Grid→DCA Wealth Port, Smart DCA by IV/Hurst	~1.75 ชม.	สูง
Part 7	**Advanced Integrations — IV, Funding Rate & Options** — IV rank → step size, funding rate bias, CSP เป็น grid level	~2.75 ชม.	สูง
Part 7B	**Grid-Framework + Indicator Execution** — Grid คิดทุน/risk, Indicator ตัดสินใจเปิด order: 4 styles + Composite Score	~2 ชม.	สูง

Supplementary & Implementation (กลยุทธ์เสริมและการปฏิบัติ)

Part 8	**5 Supplementary Strategies** — Rebalancing Grid, Stablecoin Grid, Flash Crash Grid, Cross-Exchange Arb Grid, Laddered CSP Grid	~3 ชม.	สูง
Part 9	**Python Implementation Guide — WebSocket, Backtest & Live** — State machine, async engine, backtester, live trading on Bybit	~4 ชม.	สูง
Appendix	**สูตร, Glossary, Cross-Book Index** — ตารางสูตรทั้งหมด, คำศัพท์ไทย/อังกฤษ, ดัชนีข้ามเล่ม	~1.5 ชม.	อ้างอิง

เส้นทางการอ่านที่แนะนำ

เส้นทาง "เริ่มต้น Grid จากศูนย์"

1. **Part 0** — เข้าใจ Grid พื้นฐานและปรัชญา
2. **Part 1A** — Close System คือหัวใจของ risk control
3. **Part 5** — จัดขนาดและ risk ก่อนลงทุนจริง
4. **Part 3** — ปรับ step ตาม volatility
5. **Part 9** — ลงมือ implement

ถ้าคุณ...	เริ่มที่
เพิ่งรู้จัก Grid ครั้งแรก	Part 0 → Part 1A
อยากทำ Grid Hedge (ถือ Spot ป้องกัน)	Part 1A → Part 1C
อยากใช้ Indicator กรอง entry	Part 3B → Part 7B
อยากทำ Pair Grid (BTC↔ETH)	Part 5 → Part 6
อยาก compound กำไรจาก Grid	Part 6B — Grid Snowball + DCA Stack
สนใจ Options เสริม Grid	Part 7 → Part 8

การเชื่อมต่อกับซีรีส์

หนังสือในซีรีส์ที่ใช้ความรู้มาต่อ

ความรู้จากเล่ม	ใช้ใน Grid Part
Statistical Arbitrage — Hurst, OU, Cointegration, Kelly	Part 3, 4, 5, 6
Volatility Mastery — IV, ATR, GARCH, Vol Regime	Part 3, 7
Arbitrage — Funding rate, Spot-Perp, Cash & Carry	Part 1C, 7, 8
Payoff Mastery — CSP, Short Strangle, Payoff diagram	Part 7, 8
คณิตศาสตร์สำหรับ Options — Normal dist, OLS, Optimization	Part 1A, 3, 5

GRID TRADING MASTERY · PART 0

รู้จัก Grid Trading + ปรัชญา

กลไกพื้นฐาน · เปรียบเทียบ Buy-Hold/DCA · ประเภท Grid · ปรัชญา "Grid คือ Grid"

ตัวอย่าง: BTC/USDT บน Bybit

แก่นของ PART นี้ — อ่านก่อน

Grid Trading ไม่ใช่เวทมนตร์ — มันคือ **กรอบวินัย** ที่บังคับให้ซื้อต่ำขายสูงซ้ำๆ อย่างเป็นระบบ Alpha ที่แท้จริงมาจาก **โครงสร้าง Grid เอง** (ดักจับ mean-reversion) — ไม่ใช่จาก indicator หรือ trick ที่ใส่เพิ่ม

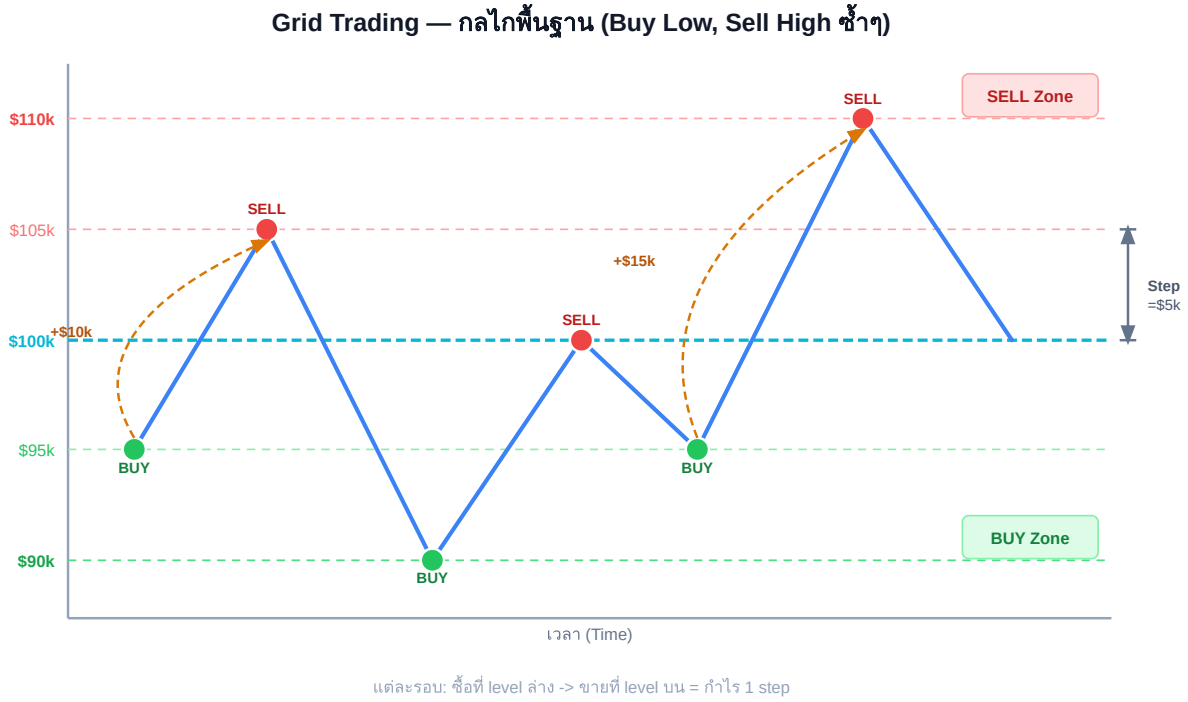
```
กำไรต่อรอบ = Q × (P_sell − P_buy) − Fees    [ทุกรอบที่ราคาแกว่งผ่าน step หนึ่งครั้ง]
```

0.1 Grid Trading คืออะไร

ลองนึกภาพคุณตั้ง "กับดัก" ราคาไว้หลายระดับ:

- **ซื้อ** เมื่อราคาตกลงถึงระดับที่กำหนด
- **ขาย** ที่ระดับถัดขึ้นไป (ต่างกัน 1 step)
- ทำซ้ำ — ตลอดเวลา ไม่ดูแผนภูมิ ไม่ตัดสินใจเพิ่ม

ผลลัพธ์: ทุกครั้งที่ราคา **แกว่งขึ้น-ลง** ผ่านระดับ grid หนึ่งคู่ คุณได้กำไร 1 step โดยอัตโนมัติ



กลไก Grid: BUY ที่ระดับล่าง → SELL ที่ระดับบน = กำไรต่อรอบ ทุกครั้งที่ราคาแกว่งผ่าน

🔁 **Running Example: BTC/USDT บน Bybit ตลอดเล่ม**

Setup พื้นฐาน: BTC/USDT อยู่ที่ $100,000 · Grid 5 ระดับ · Step = $2,000 · ขนาด order = 0.01 BTC/ระดับ

ระดับ	ราคา	Action	กำไรถ้า Fill ทั้งคู่
+2	$104,000	SELL	+$20 (0.01 BTC × $2k)
+1	$102,000	SELL / BUY	
0	$100,000	ราคาปัจจุบัน	—
-1	$98,000	BUY / SELL	+$20 (0.01 BTC × $2k)
-2	$96,000	BUY	

Capital ที่ต้องใช้: ซื้อครบทุก level ล่าง = 0.01 × ($98k + $96k) = $1,940 (ไม่นับ margin)

0.2 กลไกพื้นฐาน — สิ่งที่ต้องรู้ก่อน

ตัวแปรหลัก 4 ตัว

ตัวแปร	คำอธิบาย	ตัวอย่าง
Zone (เขต)	ช่วงราคาที่ grid ทำงาน [ล่าง, บน]	$90,000 – $110,000
Step (ก้าว)	ระยะห่างระหว่าง grid level	$2,000 (2% ของราคา)
N (จำนวน level)	= (Zone_top – Zone_bottom) / Step	(110k–90k)/2k = 10 levels
Q (ขนาด order)	จำนวน BTC ต่อ order ต่อ level	0.01 BTC ≈ $1,000/level

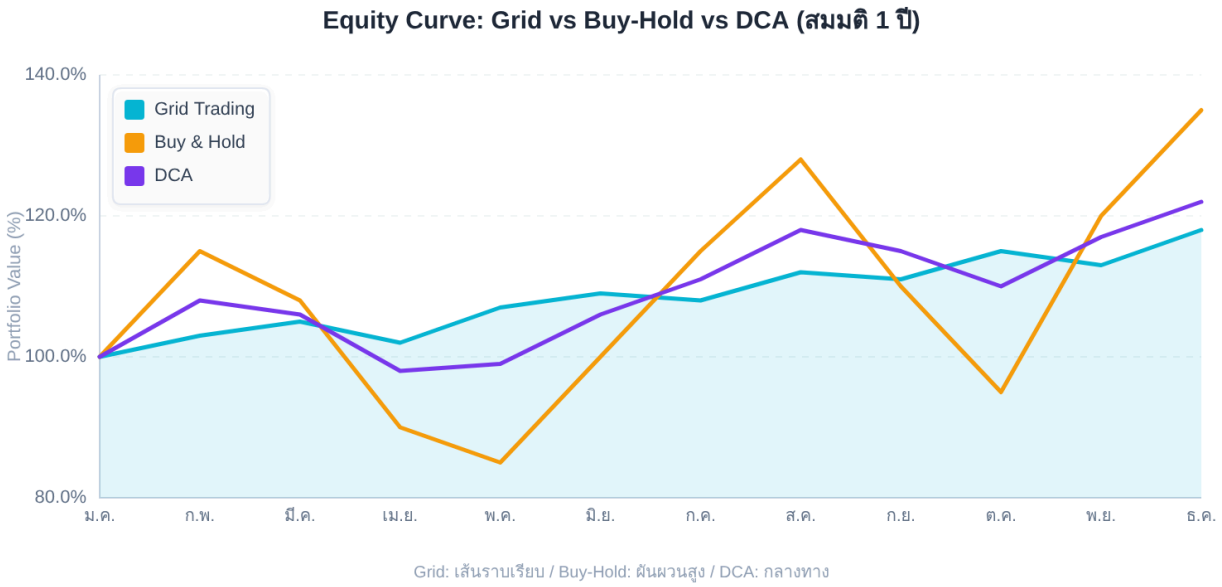
สูตรหลักที่ต้องจำ

```
กำไรต่อรอบ = Q × Step − Fees = 0.01 BTC × $2,000 − ($1,000 × 0.1% × 2) =
$20 − $2 = $18 สุทธิ Capital ทั้งหมด (Close System ถึง zone_bottom): C_total
= Q × (P_1 + P_2 + ... + P_N_below) = 0.01 × ($98k + $96k + $94k + ...
+ $90k) ← ทุก BUY level = 0.01 × Σ P_i [ถ้า N=5 levels ด้านล่าง = 0.01 ×
$490k = $4,900] รอบต่อวัน (ประมาณ) = ATR_daily / Step = $3,000/วัน /
$2,000 = 1.5 รอบ/วัน (โดยเฉลี่ย) กำไรต่อวัน (ประมาณ) ≈ 1.5 × $18 = $27/วัน บน
capital $4,900 ≈ 0.55%/วัน
```

Step เล็กหรือใหญ่ — Tradeoff สำคัญ

- **Step เล็ก:** รอบเยอะ → cashflow สูง — แต่ inventory ค้างมาก ถ้าราคาดิ่งลงแรง
- **Step ใหญ่:** รอบน้อย → cashflow ต่ำ — แต่ fill เร็ว ไม่ค้าง inventory นาน
- **Rule of thumb:** Step ≥ 2× fee เพื่อให้กำไรสุทธิเป็นบวกทุกรอบ

0.3 Grid เทียบกับ Buy-Hold และ DCA



Grid ให้ equity curve ราบเรียบที่สุด — total return ใกล้เคียงกัน แต่คุณภาพการเดินทางต่างกันมาก

มิติ	Grid Trading	Buy & Hold	DCA
กำไรหลัก	Cashflow จากการแกว่ง	Capital gain (ราคาขึ้น)	ซื้อถัวเฉลี่ย ขายเมื่อพร้อม
ต้องการ trend?	ไม่ต้อง (ต้องการ sideway)	ต้องการ (uptrend)	ได้ประโยชน์ถ้า uptrend
Drawdown	น้อย (ถ้า zone ออกแบบดี)	สูงมาก (−30% ถึง −70%)	ปานกลาง
ความซับซ้อน	กลาง (ต้องออกแบบ zone+step)	ต่ำ (ซื้อและรอ)	ต่ำ-กลาง
ดีที่สุดเมื่อ	ตลาด sideway-choppy (H < 0.5)	Bull market ยาว	ตลาดผันผวน ระยะยาว
แย่ที่สุดเมื่อ	ตลาด trending แรง (H > 0.6)	Bear market ยาว	Sideways แบบไม่ขึ้น

เมื่อไหร่ Grid ล้มเหลว

> Grid สมมติว่าตลาดแกว่ง (mean-reverting) — ถ้าราคา **ดิ่งลงไม่กลับ** หรือ **ทะยานขึ้นไม่หยุด** grid จะขาดทุนจาก inventory ที่ค้างอยู่
>
> ตัวอย่าง: BTC $100k → $60k โดยไม่กลับ → grid ซื้อตลอดทาง → inventory ค้าง → unrealized loss สูง แต่ถ้า **Close System** ออกแบบดี capital ไม่หมดจนกว่าจะถึง floor (ดู Part 1A)

0.4 ปรัชญา "Grid คือ Grid"

หลักการที่ทำให้ GRID คือ GRID

"Enhancement เสริมคุณภาพ equity curve — ไม่ได้เพิ่ม total P&L อย่างมีนัย"

Grid framework เอง (ไม่มี indicator ใดเลย) ก็มี alpha จากการดักจับ mean-reversion
Indicator ช่วยลด DD, เพิ่ม Sharpe — แต่ P&L รวมใกล้เคียงเดิม

นี่ไม่ได้หมายความว่า Enhancement ไม่มีค่า — ตรงกันข้าม:

Enhancement มีค่า — แต่ไม่ใช่แบบที่คิด

- **ลด DD:** ถ้า DD จาก 20% → 10% คุณนอนหลับได้ดีขึ้น ไม่ panic sell → *นี่คือคุณค่าที่แท้จริง*
- **เพิ่ม Sharpe:** Sharpe จาก 0.8 → 1.2 หมายถึง risk-adjusted return ดีขึ้น → ขยาย capital ได้มากขึ้น
- **ลดเวลาใน DD:** ฟื้นตัวเร็วขึ้น → ใช้ capital ได้ productive มากขึ้น

สิ่งที่ไม่ควรคาดหวัง: indicator จะทำให้ P&L เพิ่มขึ้น 50% จาก grid ปกติ — นั่นคือ overfitting

หลักฐาน: Coin Flip เป็น Indicator ก็ยังมีกำไร

ลองสมมติ: ทุก 8 ชั่วโมง โยนเหรียญ → หัว = เปิด grid, ก้อย = หยุด grid

ผลที่ได้: กำไร ≈ 50% ของ grid ที่รันตลอด 24 ชั่วโมง (เพราะเปิดแค่ครึ่งเวลา) — แต่ DD ลดลงด้วย

ถ้า ADX filter ทำได้ดีกว่าโยนเหรียญ → มีค่า | ถ้าเท่ากัน → ใช้ random filter ประหยัดเวลา backtest

0.5 ประเภทของ Grid ที่เราจะเรียน

Buy-Only Grid (Close System)

ซื้ออย่างเดียว ขายเพื่อ take profit เท่านั้น — ไม่มี SL ยกเว้น 0 หรือ give-up point

เหมาะกับ: ผู้ถือ long-term view, ต้องการ passive income จากการแกว่ง

→ Part 1A

Buy-and-Sell Grid (Bidirectional)

มีทั้งฝั่งซื้อ (inventory long) และฝั่งขาย (inventory short) — profit จากทั้งขาขึ้นและขาลง

เหมาะกับ: ตลาด choppy, ผู้ใช้ perpetual futures

→ Part 1B

Grid Hedge (Spot + Perp)

ถือ Spot BTC ไว้ + short perp เป็น hedge — เก็บ funding rate และกำไรจาก grid

เหมาะกับ: ผู้ถือ spot asset อยู่แล้ว อยากลด risk

→ Part 1C

0.6 สามเสาหลักของ Grid ทุกแบบ

ไม่ว่าจะเป็น Grid แบบไหน ต้องตัดสินใจ 3 อย่าง:

เสาที่ 1: Zone — "เล่นที่ไหน"

กำหนดช่วงราคา [ล่าง, บน] ที่ grid จะทำงาน — Zone กว้างเกินไป = capital ล็อคมากเกินไป | Zone แคบเกินไป = ราคาออกนอก zone บ่อย

เครื่องมือ: Donchian Channel (price-based), Bollinger Band (σ-based), ATR multiple, Monte Carlo + Block Bootstrap

→ ดูรายละเอียดใน Part 3

เสาที่ 2: Step — "ห่างแค่ไหน"

กำหนดระยะห่างระหว่าง grid level — Step เล็ก = รอบเยอะ แต่ inventory risk สูง | Step ใหญ่ = รอบน้อย แต่ fill ช้า

เครื่องมือ: ATR-based step (Step ≈ k × ATR), Keltner Channel, IV-adaptive step

→ ดูรายละเอียดใน Part 3

เสาที่ 3: Capital — "ใช้เท่าไหร่ อย่างไร"

กำหนดจำนวนเงินที่ต้องใช้ต่อ level และ total capital ที่ reserve ถึง floor — Capital ไม่พอ = margin call ก่อน grid เต็มระบบ

เครื่องมือ: Close System formula, Kelly sizing, Zone Waterfall (Active/Standby/Floor)

→ ดูรายละเอียดใน Part 1A และ Part 5

0.7 ลำดับการเรียนและสิ่งที่รอคุณอยู่

Part	เนื้อหา	สิ่งที่จะทำได้หลังอ่าน
1A	Close System + Zone Tricks + Soft Martingale	ออกแบบ grid ที่ไม่ bust ไม่ว่าราคาจะลงแค่ไหน (ถ้า capital พอ)
1B	Buy-and-Sell (Bidirectional)	ทำกำไรทั้งขาขึ้นและขาลง บน perpetual futures
1C	Grid Hedge	ป้องกัน inventory risk ด้วย perp short, เก็บ funding rate ไปด้วย
2	Bloating + Step Pyramid	จัดการ "grid บวม" และเพิ่ม step แทน size ในช่วง trend
3	Volatility-Adaptive Zone+Step	ปรับ zone และ step ตาม ATR, IV, Donchian, Keltner อัตโนมัติ
3B	Indicator Filter + Order Accumulation	ใช้ MA/EMA/ADX กรอง entry, รวบ order ที่พลาดด้วย signal
4	Regime Detection	รู้ว่าตอนนี้ตลาด mean-revert หรือ trend → ตัดสินใจ grid on/off
5	Kelly + Risk Management	คำนวณขนาด position ที่ optimal, ป้องกัน ruin
6	Pair Grid	เทรด spread BTC↔ETH หรือ inverse correlation
6B	Grid Snowball + DCA	ทบกำไร 3 แบบ, โอน cashflow ไปสร้าง wealth portfolio
7+7B	Advanced + Framework+Indicator	IV-adaptive, funding-adaptive, Grid เป็น framework + indicator เป็น executor
8	5 Supplementary Strategies	Trend follow, Funding Arb, CSP, Strangle, Stat Arb เสริม Grid
9	Python Implementation	เขียน live bot บน Bybit ตั้งแต่ WebSocket ถึง state machine

0.8 แบบฝึกหัด

แบบฝึกหัด Part 0

- ▶ ข้อ 1: BTC อยู่ที่ $100k · Grid 10 levels · Step $2k · Q = 0.01 BTC/level — กำไรต่อรอบสุทธิ (fee 0.1%/side) คือเท่าไหร่?
- ▶ ข้อ 2: ทำไม Grid ถึงทำกำไรได้ในตลาด sideway แต่ Buy-and-Hold ไม่ได้?
- ▶ ข้อ 3: ถ้า Step = $1,000 (ครึ่งหนึ่ง) จะส่งผลต่อ capital และ cashflow อย่างไร?
- ▶ ข้อ 4: Hurst Exponent H = 0.45 ของ BTC/USDT บอกอะไรเราเกี่ยวกับ Grid?

GRID TRADING MASTERY · PART 1A

Close System + Zone Tricks + Soft Martingale

ระบบปิด (ไม่มี SL) · Zone Waterfall · Bell Curve Density · Capital Front-Loading · r-parameter

แนวคิดจาก: Close System (พี่ต้าน Mudley Group)

แก่นของ PART นี้

Close System หมายถึง: วาง capital ทั้งหมดที่ต้องการ *ก่อน* เข้าเล่น จนถึง "floor" หรือ give-up point — ไม่มี stop-loss นอกจากทุนหมด ข้อได้เปรียบ: ไม่โดน stop hunt, ไม่ต้อง top-up กลางทาง, รู้ maximum loss ล่วงหน้า

```
C_floor = Σ(i=1 to N) Q_i × P_i [ทุน reserve ถึง floor]
```

1A.1 Close System คืออะไร

Grid ส่วนใหญ่ที่สอนกันทั่วไปมี stop-loss — ถ้าราคาลงถึงจุดหนึ่งก็ปิด position ยอมขาดทุน

Close System คือแนวคิดตรงกันข้าม:

หลักการ Close System (ระบบปิด)

1. **กำหนด give-up point (floor)** — จุดที่ถ้าราคาลงถึง คุณยอมรับการถือ inventory ทั้งหมด (หรือ capital หมด)
2. **วาง capital ทั้งหมด ล่วงหน้า** — คำนวณว่าต้องใช้เงินเท่าไหร่ถ้าราคาลงถึง floor และวาง capital นั้นไว้ก่อน
3. **ไม่มี stop-loss ระหว่างทาง** — ยกเว้น capital หมดถึง floor เท่านั้น
4. **รู้ worst-case ล่วงหน้า** — maximum loss = capital ที่วางไว้ทั้งหมด (ไม่มีเซอร์ไพรส์)

ทำไมไม่มี SL กลาง grid?

ปัญหาของ SL กลาง grid: ถ้า BTC ลงจาก $100k → $95k → SL triggered → ขาดทุน $5k → จากนั้น BTC ฟื้นกลับ $105k — คุณเสียโอกาสทำกำไรทั้งหมด

Close System ไม่มี SL กลาง → ถ้า BTC ลง $95k ก็รอ → ถ้ากลับ $105k → กำไร | ถ้าไม่กลับเลย → ขาดทุนเท่า capital ที่ reserve ไว้

Trade-off: ยอมรับ maximum loss ที่ชัดเจน เพื่อแลกกับโอกาสไม่โดน stop hunt

1A.2 Floor Capital — การคำนวณ

Floor = ราคาต่ำสุดที่ยอมรับให้ grid ทำงานถึง (เช่น $80,000 สำหรับ BTC ที่ $100,000)

```
สูตร Capital ที่ต้องการถึง Floor (Equal Weight): C_floor = Q × Σ(P_i) สำหรับ
ทุก i ที่ P_i อยู่ระหว่าง [floor, entry] ตัวอย่าง: BTC entry = $100k, floor =
$90k, step = $2k, Q = 0.01 BTC/level Levels: $98k, $96k, $94k, $92k,
$90k (5 levels ด้านล่าง) C_floor = 0.01 × ($98k + $96k + $94k + $92k +
$90k) = 0.01 × $470k = $4,700 นี่คือทุน "Reserve" ที่ต้องมีก่อนเปิด grid ถ้าไม่มี
ครบ $4,700 → ไม่เปิด grid (ไม่ใช่ลดขนาด order) สูตร Soft Martingale (ขนาด Q
เพิ่มขึ้นตาม level): Q_i = Q_0 × r^(i-1) โดย r ∈ [1.1, 1.5], i =
1,2,3,...,N C_floor_martingale = Σ(i=1 to N) Q_i × P_i = Σ (Q_0 × r^(i-
1)) × P_i [ต้องคำนวณ case-by-case]
```

🔁 **Running Example: คำนวณ Floor Capital**

Level	ราคา	Q (BTC)	Capital ที่ใช้	Cumulative
Entry	$100,000	—	—	—
L1	$98,000	0.010	$980	$980
L2	$96,000	0.010	$960	$1,940
L3	$94,000	0.010	$940	$2,880
L4	$92,000	0.010	$920	$3,800
L5 (floor)	$90,000	0.010	$900	**$4,700**

สรุป: ต้องมีทุน $4,700 reserve ก่อนเปิด grid | ถ้า BTC ลงถึง $90k โดยไม่กลับ → unrealized loss = $4,700 – (0.05 BTC × $90k) = $4,700 – $4,500 = $200 เท่านั้น เพราะถือ BTC ไว้

1A.3 Zone Waterfall — Active / Standby / Floor

ปัญหาใหญ่ของ Close System: ต้องวาง capital ทั้งหมดไว้เฉยๆ (idle) รอราคาลงถึง — **Zone Waterfall แก้ปัญหานี้**

ZONE WATERFALL — ACTIVE / STANDBY / FLOOR

ACTIVE ZONE — 50%
Deploy เป็น Grid Orders ตอนนี้

▼

STANDBY ZONE — 30%
ฝาก lending / T-bill รับดอกเบี้ย

▼

FLOOR ZONE — 20%
สำรองฉุกเฉิน — ไม่แตะยกเว้นกรณีสุดท้าย

การไหลของทุน

Grid fill → คืนทุน + กำไร
ส่งกลับมาใช้ Active Zone ใหม่

↓

กำไรส่วนเกิน → Standby
ฝากรับดอกเบี้ย 4–8% APY

↓

Floor: ไม่แตะจนถึง
give-up point (Close System)

Active (50%) + Standby (30%) + Floor (20%) = 100% — ทุนทุกบาทมีบทบาท

ไม่ต้องใช้ทุนทั้งหมดเป็น grid orders — Standby หาดอกเบี้ยระหว่างรอ

วิธีทำงานของ Zone Waterfall

1. เริ่มต้น: Active Zone deploy grid orders ตาม price range ปัจจุบัน
2. เมื่อราคาลง → Active Zone depletes → Standby เลื่อนมาเป็น Active ใหม่
3. เมื่อ Standby depletes → Floor เลื่อนมา (นี่คือ warning signal)
4. เมื่อ Floor depletes → Close System ถึง give-up point → หยุด grid

ข้อดี: Standby Zone ฝากรับดอกเบี้ยได้ขณะรอ → ลด opportunity cost ของ capital ที่ idle

```
def zone_waterfall_capital(total_capital, zone_ratios=(0.5, 0.3, 0.2)):
"""คำนวณ capital ใน each zone""" active_ratio, standby_ratio,
floor_ratio = zone_ratios active_capital = total_capital * active_ratio
```

```
# deploy เป็น orders standby_capital = total_capital * standby_ratio #
ฝาก lending/yield floor_capital = total_capital * floor_ratio # ไม่แตะ #
Standby yield (สมมติ 5% APY) standby_yield_daily = standby_capital *
0.05 / 365 return { "active": active_capital, "standby":
standby_capital, "floor": floor_capital, "standby_daily_yield":
standby_yield_daily } # ตัวอย่าง: total = $10,000 # active=$5,000 ·
standby=$3,000 · floor=$2,000 # standby_daily_yield = $3,000 × 5%/365 =
$0.41/วัน (ลด opportunity cost)
```

⚠️ **Counterparty Risk ของ Lending Protocol**

Yield 4–8% APY จาก Aave/Compound/lending platforms **ไม่ใช่ risk-free**:

- **Smart contract risk:** bug หรือ exploit → เงินอาจหายทั้งหมด (เกิดกับ Euler Finance $197M ปี 2023)
- **Liquidity risk:** ในช่วง market crash → ถอน USDT ออกไม่ได้ทันที (utilization rate สูง)
- **เหมาะสำหรับ:** Standby capital ที่รอ > 30 วัน และยอมรับ smart contract risk ได้
- **ทางเลือกที่ safer กว่า:** CEX savings product (Binance Earn, OKX Earn) — counterparty risk ต่ำกว่าแต่ yield ต่ำกว่า

1A.4 Buy-Only Zone Asymmetry — Trick ที่ทรงพลัง

ZONE TRICK #1 — ASYMMETRIC ZONE

Grid ทั่วไป: วาง zone −2σ ถึง +2σ (symmetric)

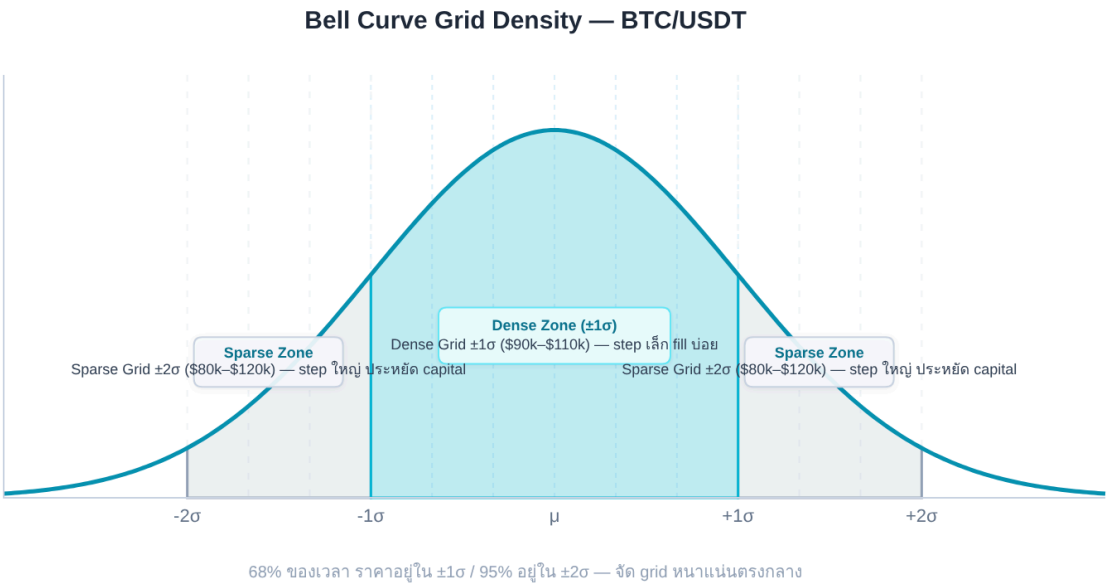
Zone Trick: วาง zone −2σ ถึง *+1σ* (ไม่ถึง +2σ) สำหรับ Buy-Only Grid

ทำไม: ราคาอยู่เหนือ +1σ น้อยมาก — วาง grid สูงเกิน +1σ เหมือน "ซื้อบนดอย" ที่ไม่ค่อย fill หรือ fill แล้วราคาไม่กลับลงมา

ข้อดีของ −2σ ถึง +1σ: ประหยัดทุนได้ ~25% (ตัดส่วนบนออก) + ลด inventory risk บนสุด

```
สมมติ BTC mean = $100k, σ = $10k (10%) Zone ทั่วไป (−2σ ถึง +2σ): $80k ถึง
$120k (range $40k) Zone Trick (−2σ ถึง +1σ): $80k ถึง $110k (range $30k)
Capital ที่ประหยัด: Grid levels $110k-$120k ไม่ต้องวาง → ถ้า step = $2k →
ประหยัด 5 levels × capital ต่อ level → ≈ 25% less capital สำหรับ upper
zone ที่ "ไม่ค่อย fill" ทดสอบ: ในช่วง 1 ปี BTC อยู่เหนือ +1σ เท่าไหร่? → ตามทฤษฎี
ปกติ: ~16% ของเวลา (1 − 84th percentile) → จริงๆ BTC: ≈ 15-20% (ใกล้เคียง)
เพราะ crypto skewed right → สรุป: 80% ของเวลาราคาอยู่ใน −2σ ถึง +1σ = Zone ที่
เลือก
```

1A.5 Bell Curve Grid Density



Grid หนาแน่นที่ mean — Step เล็กในโซนที่ราคาอยู่บ่อย ประหยัด capital ในโซนที่ราคามาน้อย

ZONE TRICK #2 — VARIABLE STEP (BELL CURVE DENSITY)

แทนที่จะใช้ step เท่ากันทุก level ลองปรับ step ตามระยะห่างจาก mean:

- ใกล้ mean (±0.5σ): step เล็ก → fill บ่อย → cashflow ดี
- ห่าง mean (1σ–2σ): step ใหญ่ขึ้น → fill น้อย → ประหยัด capital

```
def variable_step(price, mean, sigma, base_step): """คำนวณ step size ตามระยะห่างจาก mean (bell curve density)""" z = abs(price - mean) / sigma # z-score ของ price multiplier = 1 + z * 0.5 # step เพิ่ม 50% ต่อ 1σ ที่ห่างออกไป return base_step * multiplier # ตัวอย่าง: mean=$100k, σ=$10k, base_step=$2k # ที่ $100k (z=0): step = $2,000 × 1.0 = $2,000 (dense) # ที่ $95k (z=0.5): step = $2,000 × 1.25 = $2,500 # ที่ $90k (z=1.0): step = $2,000 × 1.5 = $3,000 # ที่ $80k (z=2.0): step = $2,000 × 2.0 = $4,000 (sparse) # ผลลัพธ์: cashflow จาก center สูง, capital ที่ extremes น้อย
```

ข้อดีของ Bell Curve Density

- Cashflow concentration: 70%+ ของรอบ grid อยู่ใน ±1σ จาก mean → step เล็กในโซนนี้ให้ cashflow สูง

- Capital efficiency: ไม่ต้องใส่ทุนเยอะที่ extreme level ที่ fill น้อย
- Natural risk reduction: ถ้าราคาออกไปที่ extreme, step ใหญ่ → ซื้อน้อยกว่าในวิธี equal step

1A.6 Capital Front-Loading — ยืมจาก Upper Zone

ZONE TRICK #3 — CAPITAL FRONT-LOADING

ถ้าราคาอยู่ที่ mean และ upper zone ว่างเปล่า (ไม่มี sell order fill) — ทำไมต้องปล่อยทุนนั้นนอนเฉยๆ?

Idea: ยืม capital จาก upper zone (ที่ยังไม่ถูกใช้) มาเปิด grid ในราคาปัจจุบัน — เมื่อ sell order fill ที่ upper zone ทีหลัง → คืนทุนอัตโนมัติ

```
ตัวอย่าง Capital Front-Loading: Grid zone: $90k ถึง $110k · Step $2k · Q = 0.01 BTC · ราคาปัจจุบัน = $100k ปกติ: - BUY orders ที่ $98k, $96k, $94k, $92k, $90k (ด้านล่าง) - SELL orders ที่ $102k, $104k, $106k, $108k, $110k (ด้านบน) - Capital ใน BUY side = $4,700 (ตามที่คำนวณ) - Capital ใน upper zone = 0 (ยังไม่มี inventory) Front-Loading: - เปิด BUY orders เพิ่มจาก $100k ลงมา (aggressive, เต็ม zone) - รวม BUY จาก $100k ลงถึง $90k = 5 + 1 orders เพิ่มขึ้น - ใช้ capital จาก "upper zone allocation" มา deploy เพิ่ม - เมื่อ SELL orders fill ที่ upper zone → คืน capital ข้อดี: deploy capital productive มากขึ้น | ข้อเสีย: ถ้าราคาลงก่อน = inventory เยอะกว่าปกติ
```

ความเสี่ยงของ Front-Loading

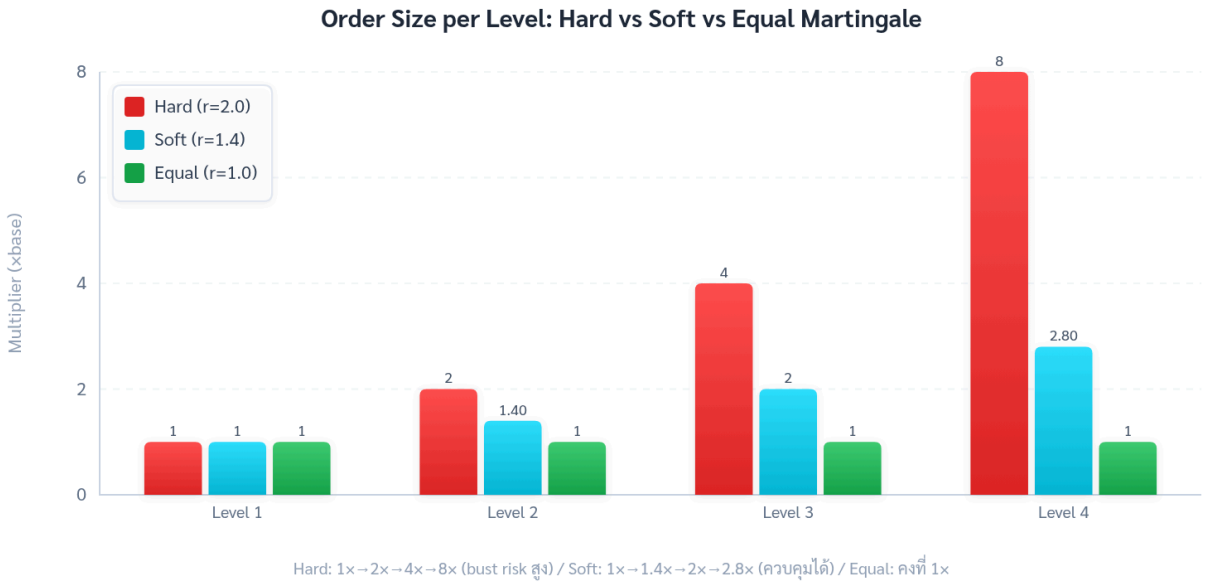
ถ้าราคาลงก่อนที่ upper zone จะ fill → front-loaded capital กลายเป็น "ขาดทุน unrealized" มากกว่าปกติ

วิธีลด risk: Front-load เฉพาะเมื่อ H < 0.45 (mean-reverting ชัดเจน) และ ATR ต่ำกว่าค่าเฉลี่ย (ตลาดเงียบ)

1A.7 Soft Martingale — r-parameter

Martingale ดั้งเดิม: เพิ่ม size เป็น 2×, 4×, 8× ทุกครั้งที่ราคาลง → capital เพิ่มแบบ exponential จนอาจ bust

Soft Martingale: ใช้ r ∈ [1.0, 2.0] (ปรับได้) แทน r = 2 ตายตัว



Soft Martingale (r=1.4) เพิ่ม size ช้ากว่า Hard Martingale (r=2) — capital requirement ต่างกันมาก

```
def soft_martingale_sizes(n_levels, base_q, r): """คำนวณขนาด order ในแต่ละ level ด้วย soft martingale""" sizes = [base_q * (r ** i) for i in range(n_levels)] return sizes # Hard Martingale r=2.0 (n=5 levels): # [1×, 2×, 4×, 8×, 16×] × base_q # Total capital ∝ 1+2+4+8+16 = 31× base_q (สูงมาก) # Soft Martingale r=1.4 (n=5 levels): # [1×, 1.4×, 2.0×, 2.7×, 3.8×] × base_q # Total capital ∝ 1+1.4+2.0+2.7+3.8 = 10.9× base_q (ต่ำกว่ามาก) # Equal Weight r=1.0 (baseline): # [1×, 1×, 1×, 1×, 1×] × base_q # Total capital ∝ 5× base_q (ต่ำสุด, safe) def calculate_capital_requirement(prices, base_q, r): """คำนวณ capital ทั้งหมดสำหรับ soft martingale grid""" total = 0 for i, price in enumerate(prices): q_i = base_q * (r ** i) total += q_i * price return total
```

r parameter	ชื่อ	Capital ที่ Level 5 (relative)	เหมาะกับ
1.0	Equal Weight	1×	Conservative, baseline
1.2	Gentle Soft	2.1×	มั่นใจ mean-reversion เล็กน้อย
1.4	Moderate Soft	3.8×	สมดุล (แนะนำ)
1.6	Aggressive Soft	6.6×	มั่นใจ mean-reversion มาก
2.0	Hard Martingale	16×	ไม่แนะนำ — bust risk สูง

Stress Test ก่อนเลือก r — BTC ลง 40% ต้องใช้ทุนเท่าไหร่?

ตัวอย่าง: Grid 5 levels · Q=0.001 BTC · r=1.4 · Price $100k

Level ที่ Fill	Q (Soft Martingale r=1.4)	Capital Deployed	Cumulative BTC
1 ($98k)	0.001	$98	0.001
2 ($96k)	0.0014	$134	0.0024
3 ($94k)	0.00196	$184	0.00436
4 ($92k)	0.00274	$252	0.00710
5 ($90k)	0.00384	$346	0.01094
รวม	—	**$1,014**	**0.01094 BTC**

ถ้า BTC ลงต่อถึง $60k (–40% จาก $100k): Inventory loss = 0.01094 × ($100k – $60k) = **– $437.6**

กฎ: ก่อนเลือก r ให้ทำ stress test: Close System capital ต้อง ≥ (cumulative Q ทุก level × worst-case price drop)

r > 1.5 เสี่ยงมากใน bear market — Close System อาจหมดก่อนถึง floor

ข้อจำกัดสำคัญของ Soft Martingale

Soft Martingale ไม่ได้ลบ risk ออก — มันแค่ *ลดความเร็ว* ที่ capital เพิ่มขึ้น

ต้องใช้ร่วมกับ Close System เสมอ: คำนวณ capital requirement ทั้งหมดถึง floor ก่อน แล้วค่อยตัดสินใจว่า r เท่าไหร่ที่ capital พอ

Rule: ถ้า capital ที่ต้องการถึง floor > 50% ของ net worth → ลด r ลงหรือลด N

1A.8 Step Pyramid — เพิ่ม Step แทน Size ในช่วง Trend

IDEA ใหม่ — STEP PYRAMID (ต่างจาก SIZE PYRAMID)

ในช่วง uptrend ชั่วคราว (H ขึ้นแต่ยังไม่ถึง 0.6): แทนที่จะเพิ่ม size ของ order (martingale) → **เพิ่ม step size** แทน

- **Size Pyramid:** step คงที่, size เพิ่ม → inventory risk เพิ่ม
- **Step Pyramid:** size คงที่, step เพิ่ม → inventory risk เท่าเดิม แต่ fill น้อยลง

ผล: cashflow คล้ายกัน (step ใหญ่ × fill น้อย ≈ step เล็ก × fill เยอะ) แต่ inventory accumulation ช้าลงในช่วง trend

```
def step_pyramid(base_step, current_h, h_normal=0.45, h_max=0.60): """ ปรับ step size ตาม Hurst Exponent: - H ต่ำ (mean-reverting): step = base_step (หรือเล็กกว่า) - H สูง (trending): step เพิ่มขึ้น เพื่อหยุดการ accumulate inventory เร็วเกินไป """ if current_h <= h_normal: return base_step elif current_h >= h_max: return base_step * 2.0 # ช่วง trending — double step (fill น้อยลง 50%) else: # interpolate ratio = (current_h - h_normal) / (h_max - h_normal) return base_step * (1.0 + ratio * 1.0) # ตัวอย่าง: base_step = $2,000 # H=0.45 → step = $2,000 (ปกติ) # H=0.52 → step = $2,467 (เริ่ม trend) # H=0.60 → step = $4,000 (trending ชัด → ไม่ค่อย fill) # H>0.60 → พิจารณา pause grid (ดู Part 4)
```

1A.9 Running Example รวม — Close System + Soft Martingale

🔁 **Running Example: BTC/USDT Close System สมบูรณ์**

Parameters: Entry = $100k · Floor = $90k · Step = $2k · r = 1.3 · Base Q = 0.01 BTC

Level	ราคา	Q (BTC)	Capital ($)	Cumulative ($)	กำไรถ้า Sell
L1	$98,000	0.010	$980	$980	+$20
L2	$96,000	0.013	$1,248	$2,228	+$26
L3	$94,000	0.017	$1,598	$3,826	+$34
L4	$92,000	0.022	$2,024	$5,850	+$44
L5 (floor)	$90,000	0.029	$2,610	**$8,460**	+$58

Capital ที่ต้องมี (Close System): $8,460

Zone Waterfall: Active $4,230 (ใช้งาน) · Standby $2,538 (lending ~5% APY → $0.35/วัน) · Floor $1,692 (สำรอง)

Max Drawdown (worst case): ถ้า BTC ลงถึง $90k ตรงๆ = unrealized ~$1,500 (เพราะถือ BTC ที่ซื้อมาทั้งหมด)

Daily cashflow (ประมาณ, 1.5 รอบ/วัน, เฉลี่ย): ≈ 1.5 × ($30 avg profit) = $45/วัน = 0.53%/วัน บน $8,460

1A.10 แบบฝึกหัด

แบบฝึกหัด Part 1A

▶ ข้อ 1: Close System BTC entry $100k, floor $85k, step $2k, Q = 0.01 BTC — ต้องการ capital เท่าไหร่?

▶ ข้อ 2: ทำไม Buy-Only zone ถึงแนะนำ −2σ ถึง +1σ แทน −2σ ถึง +2σ?

▶ ข้อ 3: Soft Martingale r=1.4, base_q=0.01 BTC, 5 levels — capital รวมมากกว่า Equal Weight กี่%?

▶ ข้อ 4: ทำไม Step Pyramid ดีกว่า Size Pyramid ในช่วง uptrend?

▶ ข้อ 5 (Challenge): ออกแบบ Close System + Soft Martingale สำหรับ $20,000 capital — BTC $100k, floor $80k, step $2k

GRID TRADING MASTERY · PART 1B

Buy-and-Sell Grid (Bidirectional)

Long Layer + Short Layer · กำไรทั้งขาขึ้นและขาลง · Capital Allocation · Funding Rate

ตัวอย่าง: BTC/USDT Perpetual บน Bybit

แก่นของ PART นี้

Buy-and-Sell Grid (Bidirectional) มีทั้ง Long Layer และ Short Layer — Long buy ต่ำ ขายสูง, Short sell สูงซื้อคืนต่ำ → ทำกำไรได้ทั้ง sideway ขาขึ้นและขาลง แต่ต้องการ **Perpetual Futures** สำหรับ Short side

```
กำไรรวม = กำไร Long Layer + กำไร Short Layer − Funding Rate ที่จ่าย (net)
```

1B.1 Buy-Only vs Buy-and-Sell — เปรียบเทียบ

Buy-Only Grid (Part 1A)

- ซื้อเมื่อราคาลง, ขาย (TP) เมื่อขึ้น
- ต้องการ spot หรือ perp long เท่านั้น
- กำไรเมื่อราคา sideway → สูงขึ้น
- ขาดทุน unrealized เมื่อราคาดิ่งต่ำ
- เหมาะ: long-term holder + passive income

Buy-and-Sell Grid (Bidirectional)

- Long layer: ซื้อต่ำขายสูง (ด้านล่าง)
- Short layer: ขายสูงซื้อคืนต่ำ (ด้านบน)
- ต้องการ perpetual futures (margin)
- กำไรทั้งขาขึ้น ขาลง และ sideway
- เหมาะ: choppy market, futures trader

ข้อสำคัญ: Buy-and-Sell Grid **ไม่ได้** "ดีกว่า" Buy-Only เสมอ — ใน bull market ที่ราคาขึ้นตลอด Short layer จะขาดทุนต่อเนื่อง ต้องเลือกให้เหมาะกับ regime ตลาด

1B.2 โครงสร้าง Long Layer + Short Layer

ในระบบ Bidirectional Grid เราแบ่งโซนออกเป็น 2 ส่วน:

```
ตัวอย่าง: BTC = $100,000 · Zone $90k–$110k · Step $2k Long Layer (ด้านล่าง current price): L1: BUY $98k → SELL $100k (profit $2k × Q) L2: BUY $96k → SELL $98k L3: BUY $94k → SELL $96k L4: BUY $92k → SELL $94k L5: BUY $90k → SELL $92k Short Layer (ด้านบน current price): S1: SELL $102k → BUY $100k (profit $2k × Q) S2: SELL $104k → BUY $102k S3: SELL $106k → BUY $104k S4: SELL $108k → BUY $106k S5: SELL $110k → BUY $108k กำไรต่อรอบ (แต่ละ layer) = Q × Step − Fees = Q × $2,000 − (Q × P × 0.1% × 2)
```

กลไก: ราคาแกว่งผ่าน current price ซ้ำๆ

เมื่อราคาแกว่งขึ้นจาก $98k → $102k: Long L1 fill (ซื้อ $98k ขาย $100k) + Short S1 fill (ขาย $102k ซื้อ $100k) = 2 รอบจากการแกว่งครั้งเดียว

นี่คือ "double capture" ของ bidirectional — แต่ต้องการ margin สำหรับทั้ง 2 layer

1B.3 Capital Allocation ทั้งสองฝั่ง

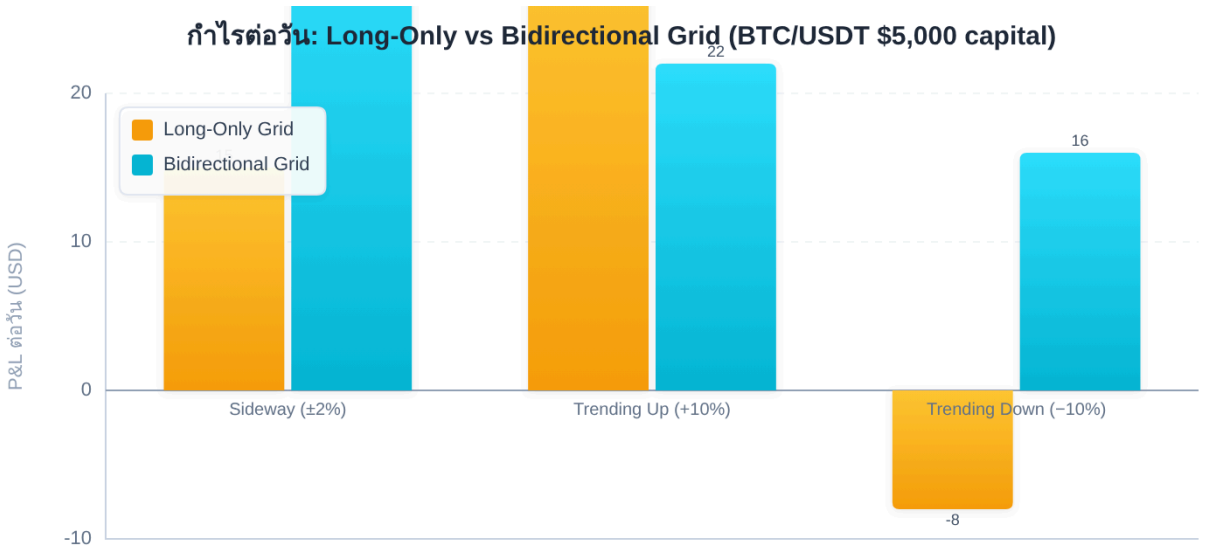
```
Capital ที่ต้องการ (ไม่ใช้ Leverage): Long Layer (ซื้อ BTC ด้วยเงิน):
Capital_long = Q × Σ(P_i) สำหรับทุก Long level = 0.01 × ($98k + $96k +
$94k + $92k + $90k) = 0.01 × $470k = $4,700 Short Layer (ต้องการ margin
สำหรับ short position): Capital_short = Q × Σ(P_i) × margin_rate (เช่น 5%
= 20× leverage) = 0.01 × ($102k + $104k + $106k + $108k + $110k) × 5% =
0.01 × $530k × 5% = $265 (ถ้าใช้ 20× leverage) = $530 (ถ้าใช้ 10× leverage)
Capital รวม (20× leverage บน short): Total = $4,700 (long) + $265
(short) = $4,965 เปรียบกับ Buy-Only: $4,700 + 6% เท่านั้น แต่ได้รอบเพิ่มอีก 2× ใน
ช่วง sideway
```

Leverage Warning — Short Layer

Short layer ใช้ leverage — ถ้าราคาพุ่งขึ้นแรง (short squeeze) short position มี unrealized loss สูง

กฎ: Short layer leverage ≤ 10× เสมอ | ตั้ง stop-loss สำหรับ short layer (ต่างจาก long layer ที่ไม่มี SL ใน Close System)

ทำไม short ถึงมี SL แต่ long ไม่มี? เพราะ long สามารถถือ BTC ได้โดยไม่มีต้นทุนเพิ่ม แต่ short position มี funding cost ต่อเนื่องและ unlimited loss ถ้าราคาพุ่ง



Sideway: Bidirectional ดีกว่า / Up Trend: Long-only ได้กำไรเพิ่มจาก inventory / Down: Bidirectional ป้องกันได้ด้วย Short Layer

Bidirectional Grid ทำกำไรได้ทั้ง 3 สถานการณ์ — Long-only ขาดทุนเมื่อราคาลง แต่ Bidirectional ยังได้กำไรจาก Short Layer

1B.4 Funding Rate — ต้นทุนซ่อนของ Short Layer

Perpetual futures มี funding rate ที่จ่ายทุก 8 ชั่วโมง — ถ้า funding rate เป็นบวก ฝั่ง short ได้รับเงิน ถ้าเป็นลบ ฝั่ง short จ่ายเงิน

```
Funding Cost/Income (Short Layer): funding_pnl = short_notional × funding_rate × (24/8) per day = (0.01 BTC × $106k avg) × rate × 3 # avg($102k...$110k) = $106k = $1,060 × rate_per_8h × 3 ตัวอย่าง: Funding rate = +0.05%/8h (บวก → short ได้รับเงิน) Funding income = $1,060 × 0.05% × 3 = $1.59/วัน ✓ Funding rate = −0.02%/8h (ลบ → short จ่ายเงิน) Funding cost = $1,060 × 0.02% × 3 = $0.636/วัน ✗ กำไร grid short layer = (รอบ × profit/รอบ) − funding_cost ≈ 1 รอบ × ($20 − fee) − $0.636/วัน = $18 − $0.64 ≈ $17.36/วัน (ยังกำไรอยู่)
```

Funding Rate Adaptation (จาก Part 7)

ถ้า funding rate สูงมากเป็นบวก (เช่น 0.3%/8h) → Short layer ทำกำไรจาก funding เพิ่มนอกจาก grid cycles → เพิ่ม size short layer

ถ้า funding rate ลบมาก → Short layer เสีย → ลด size short layer หรือปิดชั่วคราว

ดูรายละเอียดใน [Part 7: Funding Rate Adaptive Grid]

1B.5 เมื่อไหร่ Bidirectional ดีกว่า Buy-Only

สถานการณ์	Buy-Only	Bidirectional	แนะนำ
Sideway choppy (H < 0.45)	ดี	ดีกว่า (2× cycles)	Bidirectional
Bull trend ช้า (H ≈ 0.52)	ดี (inventory มีมูลค่า)	กลาง (short layer drag)	Buy-Only
Bull trend แรง (H > 0.58)	กลาง (grid หยุด)	แย่ (short layer ขาดทุน)	ทั้งคู่ไม่ดี → หยุด grid
Bear trend (ราคาลงต่อเนื่อง)	แย่ (inventory ขาดทุน)	กลาง (short layer ได้)	Bidirectional (short captures)
Funding rate +สูง (>0.1%)	ไม่ได้ประโยชน์	ดีมาก (short gets funding)	Bidirectional

1B.6 Close System สำหรับ Bidirectional

```
Close System Bidirectional: Long side: วาง capital ถึง zone_bottom
(เหมือน Part 1A) C_long_floor = Σ(Q × P_i) สำหรับ Long levels ถึง floor
Short side: ใช้ different approach — short layer มี SL SL ของ short layer
= zone_top + buffer (เช่น zone_top + 5%) Capital_short = Q × N_short ×
P_avg × margin_rate Maximum loss (short side) = (SL_price − P_entry) ×
Q × N_short กฎ Close System Bidirectional: 1. Long layer: ไม่มี SL (Close
System ปกติ) 2. Short layer: มี SL ที่ zone_top + 5% (เพื่อป้องกัน short
squeeze) 3. Short capital ≤ 20% ของ long capital เสมอ (เพื่อให้ long side
dominate ใน bull scenario)
```

1B.7 Running Example — Bidirectional BTC/USDT

🔁 **Running Example: BTC Bidirectional Grid**

Setup: BTC = $100k · Zone $92k–$108k · Step $2k · Q = 0.01 BTC · Leverage short = 10×

Layer	ระดับ	ราคา	Action	Capital
Long	L1	$98k–$100k	BUY $98k / SELL $100k	$980
	L2	$96k–$98k	BUY $96k / SELL $98k	$960
	L3	$94k–$96k	BUY $94k / SELL $96k	$940
	L4	$92k–$94k	BUY $92k / SELL $94k	$920
Short	S1	$100k–$102k	SELL $102k / BUY $100k	$102 (10× margin)
	S2	$102k–$104k	SELL $104k / BUY $102k	$104
	S3	$104k–$106k	SELL $106k / BUY $104k	$106
	S4	$106k–$108k	SELL $108k / BUY $106k	$108

Capital รวม: Long $3,800 + Short $420 = **$4,220**

SL ของ Short Layer: $108k × 1.05 = $113,400 (ถ้า BTC พุ่งเกินนี้ → ปิด short ทั้งหมด)

Daily estimate: 3 รอบ (2 long + 1 short) × $18 avg = $54/วัน = 1.28%/วัน บน $4,220

เปรียบ Buy-Only เดิม (4 levels): 1.5 รอบ × $18 = $27/วัน = 0.72%/วัน บน $3,800

Bidirectional: +78% cashflow ด้วย capital +11% เพิ่ม — แต่ขึ้นกับ market regime

1B.8 Pseudocode — Bidirectional Grid Manager

```
class BidirectionalGrid: def __init__(self, symbol, zone_low,
zone_high, step, q_long, q_short, leverage_short=10,
short_sl_buffer=0.05): self.zone_low = zone_low self.zone_high =
zone_high self.step = step self.q_long = q_long # BTC per long level
self.q_short = q_short # BTC per short level self.short_sl = zone_high
* (1 + short_sl_buffer) def setup_orders(self, current_price): orders =
[] # Long Layer — levels below current price level = current_price -
self.step while level >= self.zone_low: orders.append({"side": "BUY",
"price": level, "qty": self.q_long, "tp": level + self.step, "sl":
None}) # no SL (Close System) level -= self.step # Short Layer — levels
above current price level = current_price + self.step while level <=
self.zone_high: orders.append({"side": "SELL", "price": level, "qty":
self.q_short, "tp": level - self.step, "sl": self.short_sl}) # SL
required level += self.step return orders def check_short_sl(self,
current_price): """ตรวจ short squeeze — ถ้าราคาพุ่งเกิน SL ปิดทุก short""" if
current_price > self.short_sl: return "CLOSE_ALL_SHORTS" return "OK"
```

1B.9 แบบฝึกหัด

แบบฝึกหัด Part 1B

- ▶ ข้อ 1: Bidirectional BTC $100k, Zone $95k–$105k, Step $2k, Q=0.01 BTC — กี่รอบ/วันถ้า ATR = $3k?
- ▶ ข้อ 2: ทำไม Short Layer ถึงต้องมี SL แต่ Long Layer ไม่ต้อง?
- ▶ ข้อ 3: Funding rate = –0.05%/8h ส่งผลต่อ Short Layer อย่างไร? ควรทำอย่างไร?
- ▶ ข้อ 4: ตลาด Bull Run ชัด (H=0.65, ADX=40) — ควรทำ Bidirectional หรือ Buy-Only?

GRID TRADING MASTERY · PART 1C

Grid Hedge — Spot + Perp Delta-Neutral

ป้องกัน Inventory Risk · Funding Rate Income · Cash & Carry + Grid · Hedge Ratio

อ้างอิง: Arbitrage Series (Funding Rate, Spot-Perp Basis)

แก่นของ PART นี้

Grid Hedge คือการใช้ Perp Short ป้องกัน inventory risk ของ Spot Grid — ถือ spot BTC เพื่อ grid trading แต่ short perp เท่าๆ กัน → net delta ≈ 0 → กำไรจาก grid cashflow + funding rate เท่านั้น (ไม่สนใจ BTC price direction)

```
Net P&L = Grid Cashflow + Funding Income − (Spot P&L +
Perp P&L) ≈ Grid + Funding [ถ้า hedge สมบูรณ์]
```

1C.1 ทำไมต้อง Hedge Spot Grid

ปัญหาของ Buy-Only Spot Grid: เมื่อราคาลงมาก grid ซื้อ BTC ทั้งหมด → ถือ BTC ที่มีมูลค่าน้อยกว่าที่ซื้อ → unrealized loss สูง

Spot Grid (ไม่ Hedge)

BTC ลง 20%

→ Inventory loss: −20% × Capital

→ Grid cashflow: +$450

Net: ขาดทุน

+

Perp Short Hedge

BTC ลง 20%

→ Short profit: +20% × Notional

→ Funding received: +$XX

Net: กำไร

=

Hedged Grid

BTC ลง 20%

→ Spot + Short = ~0

→ Grid cashflow: +$450

Net: +$450 + Funding

1C.2 โครงสร้าง Spot Grid + Perp Short

```
Setup Delta-Neutral Grid: Spot Leg (Buy-Only Grid): ซื้อ BTC ที่ $98k, $96k, $94k ... (Close System) Capital: $4,700 (5 levels × avg $940) Inventory หลัง fill ทั้งหมด: 0.05 BTC Perp Leg (Short Hedge): Short 0.05 BTC perpetual ที่ current price $100k Notional: 0.05 × $100k = $5,000 Margin (10×): $500 → Net delta = +0.05 BTC (spot) − 0.05 BTC (perp short) = 0 กำไรจาก Price Movement: BTC ลง $20k: Spot loss = 0.05 × −$20k = −$1,000 Perp gain = 0.05 × +$20k = +$1,000 Net price P&L = $0 ✓ กำไรที่เหลือ = Grid cashflow + Funding received
```

ปัญหา: Inventory เปลี่ยนตลอดเวลา

Spot grid ไม่ได้ถือ BTC คงที่ — เมื่อ buy order fill, inventory เพิ่ม; เมื่อ sell order fill, inventory ลด

ดังนั้น hedge perp ต้องปรับตาม inventory: ถ้า inventory = 0.03 BTC → short 0.03 BTC; ถ้า inventory = 0.08 BTC → short 0.08 BTC

Dynamic Hedge Rebalancing: ปรับ perp short ทุกครั้งที่ grid fill (หรือ rebalance ทุก 1 ชั่วโมง)

1C.3 Funding Rate Income จาก Perp Short

เมื่อ funding rate เป็นบวก (ซึ่งเป็นกรณีปกติใน bull market): ฝั่ง short ได้รับ funding จากฝั่ง long

```
Funding Rate Income: funding_income = short_notional × funding_rate_per_8h × 3/วัน = (0.05 BTC × $100k) × 0.05% × 3 = $5,000 × 0.05% × 3 = $7.50/วัน Grid cashflow: 1.5 รอบ/วัน × $18/รอบ = $27/วัน Total daily income: $27 (grid) + $7.50 (funding) = $34.50/วัน = 34.50 / (4,700 + 500) × 100% = 0.66%/วัน บน total capital $5,200 เทียบกับ Buy-Only (ไม่ hedge): $27/วัน = 0.57%/วัน บน $4,700 Grid Hedge เพิ่ม income 28% โดยใช้ capital เพิ่มแค่ 11% (margin)
```

1C.4 Hedge Ratio — ไม่ต้อง 1:1 เสมอ

Hedge ratio ≠ 1 เสมอ — ขึ้นอยู่กับ view และ risk tolerance:

Hedge Ratio (Short/Spot)	Net Delta	เหมาะกับ
0% (ไม่ hedge)	+1.0 (full long)	Bull view แรง — รับ inventory risk ทั้งหมด
50%	+0.5 (half long)	Mild bull view — ลด risk ครึ่งหนึ่ง
100% (full hedge)	0 (delta-neutral)	ไม่มี view — grid + funding เท่านั้น
150%	−0.5 (net short)	Bear view — ได้ทั้ง grid + directional short gain

Adaptive Hedge Ratio

ปรับ hedge ratio ตาม signal:

- Funding rate > 0.1%/8h → เพิ่ม hedge ratio (short มากขึ้น → เก็บ funding มากขึ้น)
- H > 0.55 (trending up) → ลด hedge ratio (ถือ long มากขึ้น เพื่อ capture trend)
- Crash signal → เพิ่ม hedge ratio เป็น 100% หรือมากกว่า

HEDGE RATIO SPECTRUM — DELTA EXPOSURE

0%
No Hedge
Delta = +1.0
Full Inventory Risk
Bull View เต็มๆ
Grid Cashflow Only

→

50%
Partial Hedge
Delta = +0.5
Risk ลดครึ่งหนึ่ง
Mild Bull View
Grid + ½ Funding

→

100%
Full Hedge
Delta = 0
Delta-Neutral
No Directional View
Grid + Full Funding

→

150%
Over-Hedge
Delta = −0.5
Net Short
Bear View
Grid + Extra Short Gain

ปรับ Hedge Ratio แบบ adaptive ตาม Funding Rate, Hurst, และ Market Signal

1C.5 Cash & Carry + Grid — Combination

Cash & Carry Arbitrage (จาก Arbitrage Series): ซื้อ Spot + Short Futures Delivery ต่างเดือน → lock in basis spread

Grid Hedge เป็นรูปแบบหนึ่งของ Cash & Carry แต่ใช้ Perpetual แทน Fixed-Term Futures:

```
Classical Cash & Carry: ซื้อ 1 BTC Spot @ $100k Short 1 BTC Futures Dec @ $102k (basis = $2k) → ได้ $2k ที่ expiry ไม่ว่า BTC จะเป็นเท่าไหร่ Grid Hedge (Perpetual Variation): ซื้อ BTC Spot (ทยอยผ่าน grid) Short 1 BTC Perp (dynamic, ตาม inventory) → ได้ grid cashflow + funding แทน basis ที่ fixed ข้อดีของ Perpetual vs Futures: + ไม่มีวันหมดอายุ (ไม่ต้อง rollover) + Funding rate ปรับตลาด (บางทีได้มากกว่า basis) − Funding rate ลบได้ (ถ้า bear sentiment) − ต้องปรับ hedge ratio อยู่เสมอ
```

1C.6 Running Example — Complete Grid Hedge

🔁 **Running Example: BTC Spot Grid + Perp Short Hedge**

Setup: Capital $5,700 · Spot grid $4,700 (5 levels $98k–$90k, Q=0.01 BTC) · Perp margin $1,000 (10× = $10k notional)

สถานการณ์	Spot P&L	Perp P&L	Grid Cashflow	Funding	Net
BTC ลง $5k (sideway)	–$250	+$250	+$40	+$7.5	**+$47.5**
BTC ขึ้น $5k (sideway)	+$250	–$250	+$35	+$7.5	**+$42.5**
BTC ลง $20k (crash)	–$1,000	+$1,000	+$80	+$7.5	**+$87.5**
BTC ขึ้น $20k (pump)	+$1,000	–$1,000	+$25	+$7.5	**+$32.5**

สังเกต: ทุกสถานการณ์กำไร — เพราะ hedge ป้องกัน directional risk สมบูรณ์ กำไรมาจาก grid + funding เท่านั้น

Return: ~$40–$88/วัน บน $6,000 = 0.66–1.47%/วัน (ขึ้นอยู่กับ volatility)

1C.7 แบบฝึกหัด

แบบฝึกหัด Part 1C

▶ ข้อ 1: Spot grid inventory = 0.07 BTC · BTC = $100k · Hedge ratio 80% — Perp short notional เท่าไหร่?

▶ ข้อ 2: Funding rate –0.05%/8h · Short notional $5,000 · Grid profit $30/วัน — Net P&L/วัน?

▶ ข้อ 3: ทำไม Grid Hedge ถึงดีกว่า Buy-Only ในตลาด Bear?

▶ ข้อ 4: เมื่อไหร่ควรเพิ่ม Hedge Ratio เกิน 100%?

GRID TRADING MASTERY · PART 2

Bloating Patterns + Step Pyramid

บวมบน · บวมล่าง · บวมกลาง · Dynamic Sizing · Step Pyramid vs Size Pyramid

จุดหักล้างความเชื่อเดิม: Martingale ทำได้ใน Grid เพราะรู้ทุนล่วงหน้า

แก่นของ PART นี้

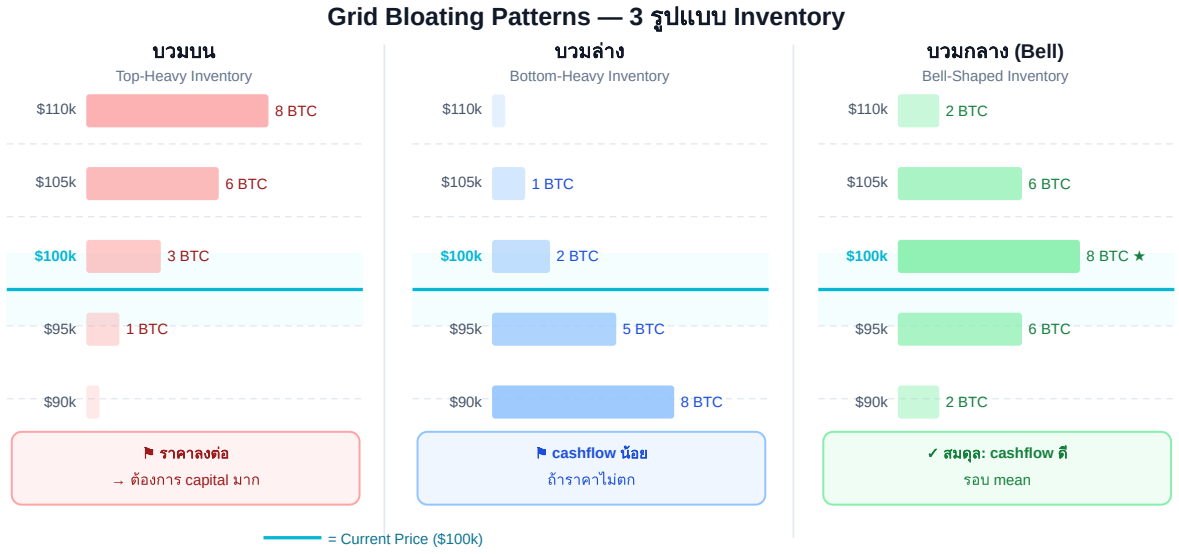
Grid **บวม** หมายถึง inventory สะสมหนาแน่นในโซนใดโซนหนึ่ง — ไม่ใช่สิ่งผิดปกติ แต่ต้องเข้าใจ pattern และออกแบบล่วงหน้า Close System ทำให้เราสามารถ **ใช้ Martingale/Dynamic Sizing** ได้อย่างปลอดภัย เพราะรู้ worst-case capital requirement แล้ว

2.1 Grid Bloating คืออะไร

เมื่อราคาอยู่ในโซนหนึ่งนานๆ grid ซื้อสะสม inventory ในโซนนั้น — เรียกว่า **grid บวม** ในโซนนั้น

ตัวอย่าง: BTC ลงจาก $100k → $90k แล้วแกว่งที่ $90k–$95k นาน 2 สัปดาห์

- Levels $98k–$92k ซื้อสะสม แต่ยังไม่ sell (ราคาไม่กลับมา)
- Levels $92k–$90k fill และ sell ซ้ำๆ (cashflow ดี)
- ผลลัพธ์: inventory "บวม" ที่ระดับ $90k–$98k มี BTC ค้างอยู่มาก



ทั้ง 3 แบบมี P&L ใกล้เคียงกัน — ความต่างคือ risk distribution และ smoothness ของ cashflow

3 รูปแบบ bloating — P&L รวมใกล้เคียงกัน แต่ risk distribution และ cashflow timing ต่างกัน

2.2 บวมบน (Top-Heavy) — Inventory สะสมสูง

บวมบน (Top-Heavy)

inventory มากในโซนบน

ราคาลง → ซื้อสูง รอขาย

เกิดเมื่อ: ราคา *เคยสูง* แล้วลงมา — grid ซื้อระหว่างทางลง แต่ราคาไม่กลับขึ้นไปขาย

```
สาเหตุ Top-Heavy: 1. เปิด grid ที่ราคาสูงกว่า mean ปัจจุบัน 2. BTC ดิ่งลง → grid ซื้อทุก level 3. ราคาหยุดแกว่งที่ต่ำกว่า zone กลาง ผลกระทบ: + Grid levels ต่ำยังทำ cashflow ได้ปกติ − Upper inventory ขาดทุน unrealized สูง − ถ้าราคาพุ่งกลับ → ขายได้ทันที + กำไรมาก − ถ้าราคาลงต่อ → inventory เพิ่มต่อ แนวทางจัดการ: Option A: รอ (Close System — ถือไว้) Option B: ลด Q ของ upper levels ลง 50% (ใช้ทุนน้อยลงในส่วนที่ "ดอย") Option C: เพิ่ม hedge (short perp ส่วนที่ upper inventory)
```

2.3 บวมล่าง (Bottom-Heavy) — Inventory สะสมต่ำ

บวมล่าง (Bottom-Heavy)

inventory มากในโซนล่าง

ราคาลงมากก่อน กลับมา

เกิดเมื่อ: ราคา *ดิ่งลงไกล* สะสม inventory มาก แล้วค่อยๆ ฟื้น — levels ล่างๆ fill ซ้ำๆ บ่อย

```
สาเหตุ Bottom-Heavy: 1. BTC crash ลง 30%+ จาก entry 2. Grid ซื้อ level ล่างสุดทั้งหมด (Close System สำเร็จ) 3. ราคาฟื้นช้า แกว่งที่ bottom zone ผลกระทบ: + Cashflow จาก lower zone ดี (รอบเยอะ) + ถ้าราคาพุ่งกลับ → ขายได้ทั้งหมด กำไรใหญ่ − Capital lock ในโซนล่างมาก − Upper zone ว่าง → ไม่ได้ใช้ชั้นบน แนวทางจัดการ: Option A: เพิ่ม grid levels ใหม่ที่โซนบน (Front-load capital จาก Zone ที่ว่าง) Option B: นำ inventory ล่างขาย แล้วซื้อใหม่สูงกว่า (Grid Reset — ดูใน Part 4)
```

2.4 บวมกลาง (Center-Heavy / Bell) — Ideal Pattern

บวมกลาง (Bell)

inventory หนาแน่น ±1σ

cashflow ดีที่สุด

เกิดเมื่อ: ราคาแกว่งรอบ mean อย่างสมดุล — Bell Curve Density (Part 1A) ออกแบบมาให้ได้ pattern นี้

```
ทำไม Bell Curve Pattern ดีที่สุด: 1. Inventory กระจายรอบ mean 2. Fill บ่อยสุด
ในโซนที่ราคาอยู่นานสุด (±1σ) 3. Capital ไม่กระจุกตัวที่ extreme 4. ถ้าราคาออกนอก
zone → inventory น้อย (step ใหญ่ไม่ fill บ่อย) วิธีรักษา Bell Pattern: - ใช้
Variable Step (bell curve density จาก Part 1A) - Rebalance ทุกสัปดาห์ถ้า
inventory เบี้ยวมาก - ดู Hurst — ถ้า H ต่ำ (mean-reverting) pattern นี้ stable
```

2.5 Dynamic Sizing — จุดหักล้างความเชื่อเดิม

ทำไม Martingale ปลอดภัยใน Close System

ความเชื่อเดิม: Martingale อันตราย — เพิ่ม size เรื่อยๆ จนทุนหมด

ใน Close System: เราคำนวณ capital ทั้งหมดที่ต้องใช้ถึง floor *ก่อน* เปิด grid — Martingale แค่กำหนดว่าจะ allocate capital อย่างไรใน range ที่คำนวณไว้แล้ว

ตัวอย่าง: มี $10,000 · Close System บอกว่า worst case ใช้ $8,000 | Soft Martingale กำหนดว่าจาก $8,000 นั้นจะใช้มากขึ้นที่ level ล่าง — แต่ไม่เกิน $8,000 รวม

```
Dynamic Sizing Pseudocode: def allocate_grid_capital(total_capital, n_levels, style="bell"): """ Allocate capital across grid levels (all styles stay within total_capital) """ if style == "equal": weights = [1.0 / n_levels] * n_levels elif style == "top_heavy": # บวมบน: น้ำหนักมากที่บน weights = [1.0 / (n_levels - i) for i in range(n_levels)] elif style == "bottom_heavy": # บวมล่าง: น้ำหนักมากที่ล่าง weights = [1.0 / (i + 1) for i in range(n_levels)] elif style == "bell": # บวมกลาง: Gaussian weights center = n_levels // 2 weights = [math.exp(-0.5 * ((i - center) / (n_levels/4))**2) for i in range(n_levels)] # Normalize weights to sum to 1 total = sum(weights) weights = [w / total for w in weights] # Allocate capital capital_per_level = [total_capital * w for w in weights] return capital_per_level # ตัวอย่าง: $10,000 · 5 levels · bell style # → [$1,500, $2,500, $3,000, $2,500, $1,500] ← หนาแน่นกลาง (weights ก่อน normalize) # หลัง normalize: ทั้งหมด = $10,000 ✓ (normalize step บรรทัด 192–193 ด้านบน)
```

2.6 Step Pyramid vs Size Pyramid

เมื่อตลาดมีแนวโน้ม (H เพิ่มขึ้น) — มีทางเลือก 2 แบบในการ "respond":

Strategy	วิธี	ผลต่อ Cashflow	ผลต่อ Inventory Risk
Size Pyramid	เพิ่ม Q ที่ level ลึก (martingale)	คงที่	สูงขึ้น (ซื้อมากถ้าลง)
Step Pyramid	เพิ่ม Step size แทน	ลดลงเล็กน้อย	ต่ำกว่า (fill น้อยกว่า)
Equal Weight	ไม่เปลี่ยนอะไร (baseline)	ลดลงตาม trend	สะสมเร็ว

```
Step Pyramid Implementation: def compute_step(base_step, level_depth, hurst, trend_factor=0.5): """ เพิ่ม step ตามความลึกของ level (ห่างจาก current price) และ Hurst exponent """ # ถ้า Hurst สูง (trending) → เพิ่ม step มากขึ้น hurst_multiplier = 1 + max(0, (hurst - 0.45) / 0.15) * trend_factor # ถ้า level ลึก (ห่างจาก mean มาก) → เพิ่ม step depth_multiplier = 1 + level_depth * 0.1 # 10% ต่อ 1 level ลึก return base_step * hurst_multiplier * depth_multiplier # ตัวอย่าง: base_step = $2,000 · Hurst = 0.55 · level depth = 3 # hurst_multiplier = 1 + (0.55-0.45)/0.15 * 0.5 = 1 + 0.333 = 1.333 # depth_multiplier = 1 + 3 × 0.10 = 1.30 # step = $2,000 × 1.333 × 1.30 = $3,467 (ใหญ่ขึ้น → fill น้อยลง)
```

Step Pyramid: กำไรเท่าเดิม สบายใจกว่า

ทำไม Step Pyramid ดี: ในช่วง trend

- Size Pyramid: ซื้อมากขึ้นที่ราคาต่ำ → ถ้า trend กลับ กำไรมาก แต่ถ้า trend ต่อ → inventory massive
- Step Pyramid: ซื้อช้าลง (fill น้อยกว่า) → inventory สะสมช้า → ถ้า trend ต่อ ไม่ติดสต็อกมาก
- P&L รวมใกล้เคียงกัน แต่ **drawdown ต่างกันมาก**

2.7 Running Example — Comparing 3 Bloating Patterns

🔁 **Running Example: BTC ลงจาก $100k → $85k แล้วแกว่ง**

Setup: Zone $80k–$110k · Step $2k · Capital $15,000 · 3 strategies เปรียบกัน

Strategy	Inventory ที่ $85k	Capital ที่ใช้	Cashflow/วัน (ที่ $83k–$88k)	Profit ถ้า BTC กลับ $100k
Equal Weight	0.08 BTC	$7,200	$36	+$1,200
Bottom-Heavy (บวมล่าง)	0.14 BTC	$9,800	$54	+$2,100
Bell Curve	0.06 BTC	$5,400	$28	+$900

สังเกต: Bottom-Heavy ได้ทั้ง cashflow สูงสุดและ profit กลับมากสุด แต่ใช้ capital มากสุดด้วย

ถ้า BTC ลงต่อไป $70k: Bottom-Heavy มี max loss สูงสุด | Bell Curve มี max loss ต่ำสุด

สรุป: เลือก style ตาม *conviction* — ถ้ามั่นใจ BTC จะกลับ → Bottom-Heavy | ถ้าไม่แน่ใจ → Bell Curve

2.8 แบบฝึกหัด

แบบฝึกหัด Part 2

▶ ข้อ 1: Grid บวมบน (Top-Heavy) มักเกิดใน market condition แบบไหน?

▶ ข้อ 2: ทำไม Size Pyramid ถึงเสี่ยงกว่า Step Pyramid ใน trending market?

▶ ข้อ 3: Bell Curve Grid density ให้ Q_center = 0.02 BTC · Q_extreme = 0.005 BTC · Step เท่ากัน $2k — เปรียบ cashflow vs Equal Weight Q=0.01 BTC ทุก level

▶ ข้อ 4: ถ้า inventory ค้าง "บวมล่าง" มาก ควร Reset Grid หรือรอ?

GRID TRADING MASTERY · PART 3

Zone Sizing — ATR, Donchian, Keltner & Bootstrap

วิธีกำหนด Zone และ Step จากข้อมูลตลาด · Donchian Channel · Keltner Channel

Block Bootstrap Monte Carlo · GARCH Volatility Forecast

อ้างอิง: Vol Series Part 2 (ATR/Volatility), Statarb Ch.3 (OU Process)

แก่นของ PART นี้

Zone และ Step ไม่ควรตั้งแบบ "ลองดู" — ควรมาจากข้อมูล volatility ของตลาดจริง ATR บอก "ราคาเดินกี่ USD/วัน" → ใช้ตั้ง Step | Donchian Channel บอก "range ของตลาดในช่วงนี้" → ใช้ตั้ง Zone

```
Step ≈ 0.5 × ATR(14) · Zone = Donchian(20) · Keltner = EMA ± ATR × k
```

3.1 ATR — ตัววัด Volatility สำหรับ Step

ATR (Average True Range) วัด "ขนาดเฉลี่ยของการเคลื่อนไหวต่อวัน" — เหมาะมากสำหรับกำหนด Step ของ grid:

$$\mathrm{ATR}(n) = \frac{1}{n}\sum_{i=1}^{n} \mathrm{TR}_i \quad \text{โดย} \quad \mathrm{TR}_i = \max(H_i - L_i,\ |H_i - C_{i-1}|,\ |L_i - C_{i-1}|)$$

```
หลักการ: Step ควรเป็น "หนึ่งหน่วยของความผันผวน" BTC/USDT — ATR(14) = $2,500/วัน Step ที่แนะนำ: Step_tight = 0.25 × ATR = $625 → fill บ่อย แต่ fee สูง Step_moderate = 0.50 × ATR = $1,250 → สมดุล (แนะนำ) Step_wide = 1.00 × ATR = $2,500 → fill น้อย cashflow ต่ำ แต่ตาม volatility พอดี Rule of Thumb: ตั้ง Step ≈ 0.5 × ATR(14) → ราคาต้อง "เดินครึ่งวัน" เพื่อ complete 1 grid cycle → ไม่ fill ถี่เกินไป (fee ไม่กิน profit) → ไม่ช้าเกินไปจนไม่ได้ทำเงิน ตรวจสอบ: กำไรต่อรอบ > 3 × ค่า Fee ต่อรอบ Q × Step > 3 × (Q × P × 0.1% × 2) Step > 3 × P × 0.002 = 3 × $100k × 0.002 = $600 ∴ Step ≥ $600 สำหรับ BTC ที่ $100k
```

ทำไม ATR(14) ไม่ใช่ ATR(7) หรือ ATR(30)?

ATR(7) = ตอบสนองเร็วเกินไป — volatility spike 2 วันทำให้ Step พุ่งแล้วหด grid ปรับถี่เกิน

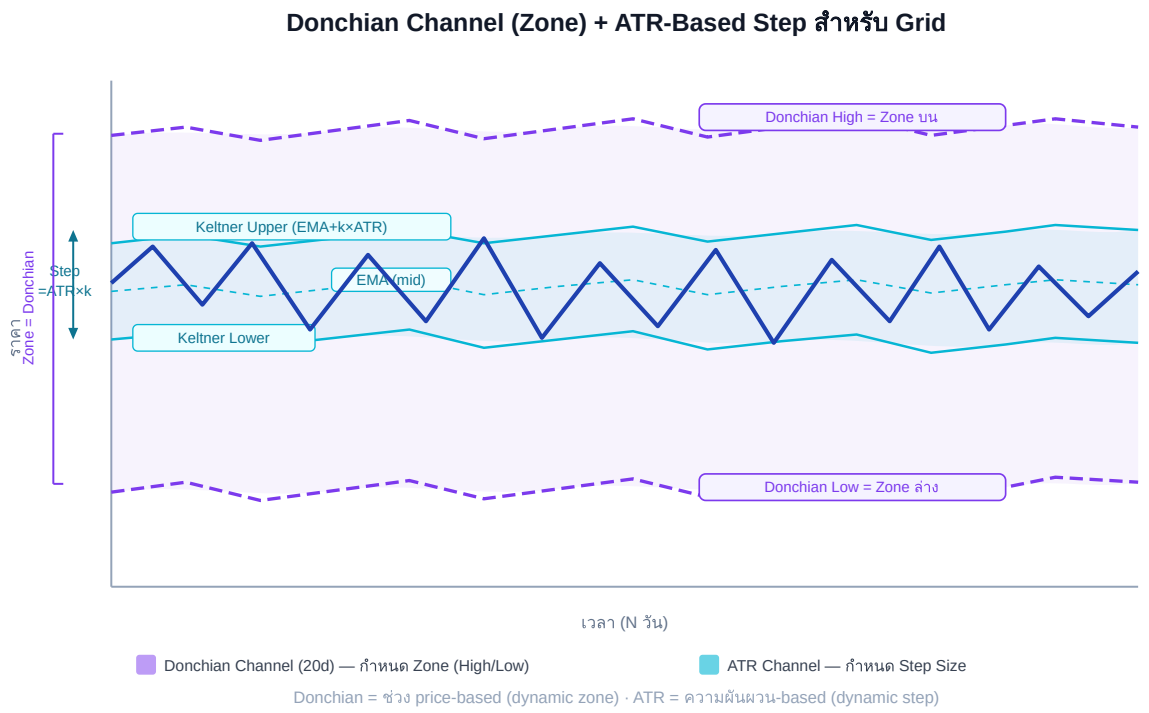
ATR(14) = 2 สัปดาห์ = ค่ากลางที่ดี: ตาม regime ปัจจุบัน แต่ไม่ noisy

ATR(30) = ตามช้าเกินไป — ใน volatility cluster ที่ ATR พุ่งแล้ว ATR(30) ยังต่ำ = step เล็กเกิน

3.2 Donchian Channel — กำหนด Zone จาก Price Range

Donchian Channel(N) = highest high – lowest low ของ N periods ที่ผ่านมา:

$$\text{Donchian}_{\text{high}}(N) = \max_{i=0}^{N-1}(H_{t-i}) \qquad \text{Donchian}_{\text{low}}(N) = \min_{i=0}^{N-1}(L_{t-i})$$



Donchian Channel (กว้าง, สีม่วง) กำหนด Zone · Keltner Channel (แคบ, สีฟ้า) กำหนด Step · ราคาแกว่งภายใน Keltner มากกว่า 68% ของเวลา

```
Donchian Zone Sizing: # Zone = Donchian(20) — range ของ 20 วันที่ผ่านมา
zone_high = max(high[-20:]) # highest high 20 วัน zone_low =
min(low[-20:]) # lowest low 20 วัน ตัวอย่าง BTC: 20-day high = $108,000
20-day low = $91,000 Zone width = $17,000 # เพิ่ม buffer 10% เพื่อรองรับ
breakout zone_high_buffered = zone_high * 1.05 # $113,400
zone_low_buffered = zone_low * 0.95 # $86,450 # ทดสอบ: zone กว้างพอไหม?
min_zone_width = ATR(14) * 6 # อย่างน้อย 6 ATR = 6 ×$2,500 = $15,000 if
zone_width < min_zone_width: zone_high = current_price * 1.08 # ขยายเป็น
±8% zone_low = current_price * 0.92
```

Donchian(20) vs Donchian(30) vs Donchian(60)

- **Donchian(20)** — ≈ 1 เดือน trading: ตาม regime ปัจจุบัน ปรับตัวเร็ว เหมาะ swing grid

- **Donchian(30)** — ≈ 6 สัปดาห์: สมดุล หน้าต่างยอดนิยม (แนะนำ)
- **Donchian(60)** — ≈ 3 เดือน: zone กว้าง grid อยู่ได้นาน แต่ capital ใช้มาก

เลือกตาม holding period ที่วางแผน: grid 1 เดือน → Donchian(20); grid 3 เดือน → Donchian(60)

3.3 Keltner Channel — Step จาก ATR Bands

Keltner Channel ใช้ EMA เป็น midline + ATR เป็น band width — เหมาะสำหรับกำหนด Step ที่ "ตาม volatility อย่าง adaptive":

$$\mathrm{KC}_{\mathrm{upper}} = \mathrm{EMA}(20) + k \times \mathrm{ATR}(14) \qquad \mathrm{KC}_{\mathrm{lower}} = \mathrm{EMA}(20) - k \times \mathrm{ATR}(14)$$

```
Keltner-Based Step Sizing: k = 1.0 (standard Keltner bandwidth = 1 ATR each side) KC_upper = EMA(20) + 1.0 × ATR(14) = $100k + $2,500 = $102,500 KC_lower = EMA(20) − 1.0 × ATR(14) = $100k − $2,500 = $97,500 KC_width = $5,000 (total 2 ATR range) # Step = Keltner half-width / desired levels per ATR desired_levels_per_atr = 2 # 2 grid levels ต่อ 1 ATR ฝั่ง step = ATR(14) / desired_levels_per_atr = $2,500 / 2 = $1,250 หรือ: step = KC_width / total_levels_in_keltner_zone = $5,000 / 4 levels = $1,250 ข้อดี Keltner Step: + Step ปรับตาม volatility อัตโนมัติ + ช่วง volatility สูง: ATR สูง → step ใหญ่ → fill ช้าลง (ป้องกัน overtrading) + ช่วง volatility ต่ำ: ATR ต่ำ → step เล็ก → fill บ่อยขึ้น (เก็บ cashflow)
```

3.3.1 Adaptive Step Implementation

```
def adaptive_step(atr14, current_price, fee_rate=0.001): """ คำนวณ step ที่ปรับตาม ATR โดยรับประกันว่ากำไรต่อรอบ > 3× fee """ # Step พื้นฐาน = 0.5 × ATR step_base = 0.5 * atr14 # ตรวจสอบ fee threshold fee_per_cycle = current_price * fee_rate * 2 # buy + sell min_step = 3 * fee_per_cycle # กำไร ≥ 3× fee step = max(step_base, min_step) # Round ให้สวยงาม (nearest 100 USD สำหรับ BTC) step = round(step / 100) * 100 return step # ตัวอย่าง: # ATR(14) = $2,500, P = $100k, fee = 0.1% # step_base = $1,250 # fee_per_cycle = $100k × 0.001 × 2 = $200 # min_step = $600 # step = max($1,250, $600) = $1,250 → round → $1,300
```

3.4 Combining Donchian + Keltner — Dual Channel Method

วิธีที่ดีที่สุด: ใช้ทั้งสองร่วมกัน — Donchian กำหนด Zone (wide), Keltner กำหนด Step (inner)

Parameter	วิธีคำนวณ	ตัวอย่าง BTC
Zone High	Donchian(30) high × 1.05	$108k × 1.05 = $113,400
Zone Low	Donchian(30) low × 0.95	$91k × 0.95 = $86,450
Step	ATR(14) × 0.5 (Keltner-based)	$2,500 × 0.5 = $1,250
N (levels)	(Zone_high – Zone_low) / Step	($113,400 – $86,450) / $1,250 ≈ 22 levels
Q (per level)	grid_capital / (N × avg_price)	$20k / (22 × $100k) = 0.0091 BTC ≈ 0.009

```
def dual_channel_sizing(prices, highs, lows, current_price, dc_period=30, atr_period=14, grid_capital=20000): """ Zone จาก Donchian(30), Step จาก ATR(14) """ # Donchian Zone dc_high = max(highs[-dc_period:]) * 1.05 dc_low = min(lows[-dc_period:]) * 0.95 # ATR Step true_ranges = [] for i in range(1, atr_period + 1): tr = max(highs[-i] - lows[-i], abs(highs[-i] - prices[-i-1]), abs(lows[-i] - prices[-i-1])) true_ranges.append(tr) atr = sum(true_ranges) / len(true_ranges) step = round(0.5 * atr / 100) * 100 # round to $100 # Grid Parameters n_levels = int((dc_high - dc_low) / step) avg_price = (dc_high + dc_low) / 2 q_per_level = grid_capital / (n_levels * avg_price) return { "zone_high": dc_high, "zone_low": dc_low, "step": step, "n_levels": n_levels, "q_per_level": round(q_per_level, 4) }
```

3.5 Block Bootstrap — Zone Sizing ที่แข็งแกร่ง

Donchian ดูแค่ high/low ในช่วงเวลา แต่ไม่บอกว่า "ความน่าจะเป็นที่ราคาจะอยู่ในช่วงนี้" คือเท่าไหร่ Block Bootstrap ช่วยตอบคำถามนี้

> ⚠️ **Bootstrap วัด Path Excursion ไม่ใช่ Final Price**
>
> สำหรับ Grid Trading: สิ่งที่สำคัญคือ **ราคาต่ำสุดระหว่างทาง** (max path excursion) ไม่ใช่ราคาสุดท้าย — grid ต้องอยู่รอดตลอด path ไม่ใช่แค่ตอนสิ้นสุด
>
> ฟังก์ชันด้านล่างคำนวณทั้ง final price และ `min_price_in_path` — ใช้ `p5_min_path` (5th percentile ของราคาต่ำสุด) เป็นตัวกำหนด zone_low

```
แนวคิด Block Bootstrap Monte Carlo: 1. เก็บ daily return ของ BTC ย้อนหลัง 90 วัน 2. สุ่ม "blocks" ของ return (ขนาด block = 5 วัน) → เพื่อรักษา autocorrelation 3. ประกอบ synthetic price path จาก blocks ที่สุ่มได้ 4. ทำซ้ำ 1,000 ครั้ง → ได้ distribution ของ "ราคาจะไปถึงไหนใน 30 วัน" 5. ใช้ percentile 5 และ 95 เป็น zone boundaries (ใช้ min_price_in_path สำหรับ zone_low — grid ต้องอยู่รอดตลอด path) def block_bootstrap_zone(daily_returns, n_sim=1000, horizon=30, block_size=5, start_price=100000): """ Bootstrap zone boundaries ด้วย 95% confidence interval คำนวณทั้ง final price และ min_price_in_path (path excursion metric) สำหรับ grid trading: zone_low ควรใช้ p5_min_path ไม่ใช่ p5_final เพราะ grid ต้องอยู่รอดตลอด path — ไม่ใช่แค่ตอนสิ้นสุด """ import random final_prices = [] min_prices_in_path = [] # path excursion metric: ราคาต่ำสุดระหว่างทาง for _ in range(n_sim): price = start_price min_price = start_price # ติดตามราคาต่ำสุดตลอด path days = 0 while days < horizon: # สุ่ม block ของ returns start_idx = random.randint(0, len(daily_returns) - block_size) block = daily_returns[start_idx : start_idx + block_size] for r in block: price *= (1 + r) min_price = min(min_price, price) # บันทึก worst excursion days += 1 if days >= horizon: break final_prices.append(price) min_prices_in_path.append(min_price) # max drawdown from start # Zone จาก final price (95% CI ของ final distribution) final_prices.sort() p5_final = final_prices[int(0.05 * n_sim)] # 5th percentile final p95_final = final_prices[int(0.95 * n_sim)] # 95th percentile final # Path excursion metric: zone_low ควรใช้ค่านี้แทน p5_final # เพราะ grid ต้องอยู่รอดตลอด path ไม่ใช่แค่ตอนสิ้นสุด min_prices_in_path.sort() p5_min_path = min_prices_in_path[int(0.05 * n_sim)] # worst 5% path low return {
```

```
"zone_low": p5_min_path, # ใช้ path excursion เป็น zone_low "zone_high": p95_final, # 95th percentile final price "p5_final": p5_final, # final price 5th pctile (reference) "p5_min_path": p5_min_path, # path excursion 5th pctile (primary) } ตัวอย่าง BTC: 1,000 simulations, 30-day horizon p5_final = $87,500 (ราคาสุดท้าย 5th pctile — ไม่พอสำหรับ grid) p5_min_path = $84,200 (ราคาต่ำสุดระหว่างทาง 5th pctile — ใช้เป็น zone_low) p95_final = $119,800 (95th pctile) → Zone: $84k–$120k (grid ต้องอยู่รอดถึง $84k ระหว่างทาง)
```

การเลือก block_size

block_size=5 (วัน) เป็นค่าเริ่มต้นที่สมเหตุสมผลสำหรับ BTC แต่ควร test หลายค่า:

- block_size เล็ก (3–5): ล้อมรับ autocorrelation สั้น (ความผันผวนรายวัน)
- block_size ใหญ่ (10–20): ล้อมรับ momentum / trend ยาวกว่า
- วิธีที่ดีกว่า: Politis-Romano Stationary Bootstrap (ไม่ต้องกำหนด block_size ตายตัว) — ใช้ `arch` library: `from arch.bootstrap import StationaryBootstrap`

ทำไม Block Bootstrap ดีกว่า Simple Historical Range?

- **Simple range**: max – min ของ 30 วัน — อาจกว้างเกิน (ถ้ามี spike) หรือแคบเกิน (ถ้าช่วง low-vol)
- **Block Bootstrap**: preserve autocorrelation (volatility cluster), ให้ probabilistic range ที่ถูกกว่า
- Zone จาก 95% CI หมายความว่า: "90% ของ scenarios ราคาอยู่ใน zone นี้" — เข้าใจง่าย จัดการ capital ได้ตาม probability

3.6 GARCH(1,1) สำหรับ Volatility-Adaptive Step

GARCH(1,1) จาก Vol Series ให้ค่า volatility forecast ที่ดีกว่า ATR ธรรมดาในช่วง volatility cluster:

$$\sigma_t^2 = \omega + \alpha \cdot \epsilon_{t-1}^2 + \beta \cdot \sigma_{t-1}^2$$

```
GARCH Step Sizing: # สมมติ GARCH fitted: ω=1e-6, α=0.08, β=0.91 #
σ_today = 0.025 (2.5% daily vol) sigma_daily = 0.025 # จาก GARCH
forecast price = 100000 # 1-day VaR = σ × price atr_garch = sigma_daily
* price # = $2,500 # Step = 0.5 × GARCH-ATR (เหมือน ATR method แต่
forward-looking) step = 0.5 * atr_garch # = $1,250 # ข้อดีของ GARCH vs
ATR: # ATR: backward-looking (เฉลี่ย 14 วันที่แล้ว) # GARCH: forward-looking
(forecast vol วันพรุ่งนี้จาก cluster ปัจจุบัน) # ช่วง volatility เพิ่ง spike →
GARCH ตอบสนองเร็วกว่า ATR(14) # Practical: ถ้าไม่ fit GARCH เอง → ใช้ implied
vol จาก options # IV ของ BTC 7-day ATM option = 65% annualized #
σ_daily = 65% / sqrt(365) = 3.4%/day → ATR_implied = $3,400
```

3.7 Volatility Regime — ปรับ Zone/Step ตาม Regime

Zone และ Step ไม่ใช่ fixed — ควรปรับตาม volatility regime ปัจจุบัน:

Regime	ATR (BTC ตัวอย่าง)	Step แนะนำ	Zone Width	N Levels
Low Vol (Consolidation)	$1,000–$1,500	$500–$700	$8k–$12k	16–24
Normal Vol	$1,500–$3,000	$750–$1,500	$15k–$25k	16–22
High Vol	$3,000–$5,000	$1,500–$2,500	$25k–$40k	16–22
Extreme Vol (Crash/Pump)	>$5,000	$2,500+	$40k+	≤ 16

```
def vol_regime_step(atr14, current_price): """ กำหนด step ตาม vol regime โดยรักษา N ≈ 16–22 levels """ atr_pct = atr14 / current_price if atr_pct < 0.015: # Low vol: ATR < 1.5% regime = "low" step_factor = 0.35 elif atr_pct < 0.030: # Normal: ATR 1.5–3% regime = "normal" step_factor = 0.50 elif atr_pct < 0.050: # High: ATR 3–5% regime = "high" step_factor = 0.60 else: # Extreme: ATR > 5% regime = "extreme" step_factor = 0.70 step = round(step_factor * atr14 / 100) * 100 return regime, step
```

3.8 Practical Zone Validation Checklist

ก่อน deploy grid ทุกครั้ง ตรวจสอบ 5 ข้อนี้:

```
def validate_zone(zone_high, zone_low, step, current_price, atr14,
grid_capital, q_per_level, fee_rate=0.001): """ ตรวจสอบ zone ก่อน deploy
Returns: list of (check_name, passed, message) """ checks = [] # 1.
Current price อยู่ใน zone in_zone = zone_low < current_price < zone_high
checks.append(("Price in Zone", in_zone, f"P={current_price:,} Zone=
[{zone_low:,},{zone_high:,}]")) # 2. Zone กว้างพอ (≥ 6 ATR) zone_width =
zone_high - zone_low wide_enough = zone_width >= 6 * atr14
checks.append(("Zone Width", wide_enough, f"Width={zone_width:,.0f}
Min={6*atr14:,.0f} (6×ATR)")) # 3. Step ≥ 3× fee per cycle
profit_per_cycle = q_per_level * step fee_per_cycle = q_per_level *
current_price * fee_rate * 2 profitable = profit_per_cycle > 3 *
fee_per_cycle checks.append(("Profitability", profitable,
f"Profit/cycle=${profit_per_cycle:.1f} Fee=${fee_per_cycle:.1f}")) # 4.
Capital พอสำหรับ worst case n_buy_levels = (current_price - zone_low) /
step worst_case_capital = n_buy_levels * q_per_level * current_price *
0.9 capital_ok = grid_capital >= worst_case_capital
checks.append(("Capital Sufficient", capital_ok,
f"Grid=${grid_capital:,} WorstCase=${worst_case_capital:,.0f}")) # 5. N
levels ≤ 30 (manageable) n_total = int(zone_width / step) manageable =
n_total <= 30 checks.append(("N Levels OK", manageable, f"N=
{n_total}")) return checks
```

3.9 Running Example — BTC Zone Sizing จากศูนย์

🔁 **Running Example: คำนวณ Zone + Step สำหรับ BTC/USDT**

ข้อมูล: BTC = $102,000 · ATR(14) = $2,800 · Donchian(30): high=$112,000 low=$89,000

```
Step 1: Zone จาก Donchian(30) + Buffer zone_high = $112,000 × 1.05
= $117,600 → ปัดเป็น $117,000 (round) zone_low = $89,000 × 0.95 =
$84,550 → ปัดเป็น $85,000 zone_width = $117,000 − $85,000 = $32,000
Step 2: ตรวจสอบ minimum zone width min_width = 6 × ATR(14) = 6 ×
$2,800 = $16,800 $32,000 > $16,800 ✓ Step 3: Step จาก ATR
step_base = 0.5 × $2,800 = $1,400 fee_min = 3 × ($102k × 0.001 ×
2) = $612 step = max($1,400, $612) = $1,400 → round to nearest
$500 = $1,500 Step 4: N และ Q N = $32,000 / $1,500 ≈ 21 levels
grid_capital = $20,000 (สมมติ) avg_price = ($117k + $85k) / 2 =
$101,000 Q = $20,000 / (21 × $101,000) = 0.00943 BTC ≈ 0.009 BTC
Step 5: Validation ✓ Price in zone: $102k ∈ [$85k, $117k] ✓ Zone
width: $32k > $16.8k ✓ Profitability: 0.009 × $1,500 = $13.5 vs
fee $1.84/cycle → 7.3× ✓ ✓ N levels: 21 ≤ 30 ✓ Step 6:
Distribution ของ BUY orders (Asymmetric Zone Part 1A) ราคาปัจจุบัน
$102k → ใกล้กลาง zone Upper zone ($102k → $117k): ใส่ SELL รอ หรือไม่
วาง order Lower zone ($102k → $85k): วาง BUY orders 11 levels ×
0.009 BTC Capital ที่ต้องใช้ worst-case: 11 × 0.009 × $93k (avg) =
$9,207
```

3.10 แบบฝึกหัด

แบบฝึกหัด Part 3

▶ ข้อ 1: BTC = $95,000 · ATR(14) = $3,200 · Fee = 0.1% — Step ที่แนะนำคือเท่าไหร่?

▶ ข้อ 2: Donchian(30) high=$108k, low=$88k · Buffer=5% · Current=$100k — Zone boundaries คืออะไร? ราคา in-zone ไหม?

▶ ข้อ 3: Grid capital $15,000 · N=20 levels · avg_price=$100k — Q per level คือเท่าไหร่? กำไรต่อรอบถ้า Step=$1,500?

▶ ข้อ 4 (Challenge): ATR ของ BTC ขึ้นจาก $2,000 → $5,000 ใน 1 สัปดาห์ (volatility spike) — ควรปรับ grid อย่างไร?

GRID TRADING MASTERY · PART 3B

Indicator Filter + Order Accumulation

RSI · Bollinger Band · Volume Filter · Composite Score 0–100

Order Accumulation Strategy · TWAP Grid · Partial Deploy

อ้างอิง: Vol Part 2 (BB/ATR), Statarb Ch.11 (State Machine)

แก่นของ PART นี้

Grid math กำหนด "ขนาด" — Indicator filter กำหนด "เวลา" ไม่ใช่ทุก indicator ที่เหมาะ ต้องเลือกเฉพาะ indicator ที่บอก "mean-reversion opportunity" ไม่ใช่ trend-following | Order Accumulation ช่วยลด timing risk เมื่อ entry ไม่แน่ใจ

```
Composite Score = 30×RSI_score + 30×BB_score +
20×Vol_score + 20×Hurst_score ∈ [0, 100]
```

3B.1 ทำไมต้อง Indicator Filter

Grid trading ทำงานได้ดีใน "mean-reverting" market — แต่ไม่ใช่ทุกเวลาที่ตลาดเป็น mean-reverting Indicator filter ช่วย:

- **Gate keeper:** บอกว่า "ตอนนี้ควรเปิด grid หรือไม่"
- **Size adjuster:** บอกว่า "ควร deploy เต็มหรือ partial"
- **Entry timing:** บอกว่า "เปิด grid ตอนนี้หรือรอ dip ก่อน"

> **สิ่งที่ Indicator Filter ไม่ควรทำ**
>
> ไม่ใช้ indicator เพื่อ "predict direction" — grid ไม่ต้องรู้ว่าราคาจะขึ้นหรือลง
>
> ใช้ indicator เพื่อตอบคำถาม: **"ตลาดเป็น mean-reverting ระดับไหน?"** เท่านั้น
>
> ถ้าใช้ MACD crossover หรือ Moving Average crossover เพื่อ "signal" — คุณกำลัง overlay trend-following บน grid ซึ่งขัดกับ grid philosophy

3B.2 RSI Filter — Mean-Reversion Opportunity

RSI (Relative Strength Index) วัด "overbought/oversold" — สำหรับ grid เราต้องการรู้ว่า "ราคาเบี่ยงจาก mean มากแค่ไหน":

$$\text{RSI} = 100 - \frac{100}{1 + \text{RS}} \qquad \text{RS} = \frac{\text{avg gain (14 periods)}}{\text{avg loss (14 periods)}}$$

```
RSI Score สำหรับ Grid (ใช้ RSI ใน Mean-Reversion Context): def rsi_score_for_grid(rsi_value): """ Score สูง = โอกาส mean-reversion สูง (ดีสำหรับ grid entry) Score ต่ำ = ตลาด momentum สูง (ไม่เหมาะ grid) """ # RSI extremes (< 30 หรือ > 70) = mean-reversion opportunity if rsi_value < 30: score = 70 + 100 * (30 - rsi_value) / 30 # ยิ่ง oversold ยิ่งสูง (70–100) elif rsi_value > 70: score = 70 + 100 * (rsi_value - 70) / 30 # ยิ่ง overbought ยิ่งสูง (70–100) else: # RSI 30–70 = middle zone # 50 = neutral (score ปานกลาง), ยิ่งห่างจาก 50 ยิ่งสูง distance_from_50 = abs(rsi_value - 50) score = 50 + distance_from_50 # 50 → score=50, 30→score=70, 70→score=70 return min(100, max(0, score)) # ตัวอย่าง: # RSI=25 → score = 70 + 100×(30-25)/30 = 86.7 (oversold มาก → grid entry ดี) # RSI=31 → score = 50 + |31-50| = 69 ≈ 70 (ใกล้ oversold → โอกาสดี) # RSI=50 → score = 50 (neutral → ปานกลาง) # RSI=72 → score = 70 + 100×(72-70)/30 = 76.7 (overbought → grid sell ดี)
```

RSI สำหรับ Entry Timing

นอกจาก filter แล้ว RSI ยังใช้กำหนด "เปิด grid ที่ชั้นไหนก่อน":

- RSI < 35 (oversold): เปิด lower zone ก่อน (ราคาน่าจะฟื้น) → deploy 70% ของ capital ที่ lower levels
- RSI 35–65 (neutral): deploy equal weight ตามปกติ
- RSI > 65 (overbought): เน้น upper zone / รอ pull-back ก่อน deploy lower zone

3B.3 Bollinger Band Score — Volatility Gate

Bollinger Band (BB) ให้ข้อมูล 2 อย่าง: ตำแหน่งของราคาเทียบ band และ width ของ band (BB Width):

$$\mathrm{BB}_{\mathrm{upper}} = \mathrm{SMA}(20) + 2\sigma_{20} \qquad \mathrm{BB}_{\mathrm{lower}} = \mathrm{SMA}(20) - 2\sigma_{20}$$

$$\%B = \frac{P - \mathrm{BB}_{\mathrm{lower}}}{\mathrm{BB}_{\mathrm{upper}} - \mathrm{BB}_{\mathrm{lower}}} \qquad \text{BB Width} = \frac{\mathrm{BB}_{\mathrm{upper}} - \mathrm{BB}_{\mathrm{lower}}}{\mathrm{SMA}(20)}$$

```
def bb_score_for_grid(price, sma20, upper, lower): """ Score สูง = ราคา
อยู่ใกล้ band (ใกล้ extreme = mean-reversion โอกาส) Score ต่ำ = ราคาอยู่กลาง
(ไม่มี extreme signal) BB Width ต่ำ (squeeze) = ดีสำหรับ grid เพราะ vol ต่ำ
fill บ่อย """ # %B: ตำแหน่งของราคาใน band if upper == lower: return 50
pct_b = (price - lower) / (upper - lower) # ถ้าอยู่นอก band (%B < 0 หรือ >
1) = extreme = โอกาสสูง if pct_b < 0: position_score = min(100, 60 + (0
- pct_b) * 200) elif pct_b > 1: position_score = min(100, 60 + (pct_b -
1) * 200) else: # ภายใน band: ยิ่งใกล้ขอบ ยิ่งดี distance_from_edge =
min(pct_b, 1 - pct_b) # 0 = ที่ band edge, 0.5 = กลาง position_score = 60
- distance_from_edge * 80 # 60 at edge, 20 at center # BB Width score:
narrow band = low vol = grid-friendly bb_width = (upper - lower) /
sma20 if bb_width < 0.02: # squeeze (<2%) width_score = 100 elif
bb_width < 0.04: # narrow width_score = 75 elif bb_width < 0.08: #
normal width_score = 50 else: # wide (high vol) width_score = 25 return
0.6 * position_score + 0.4 * width_score
```

3B.4 Volume Filter — ยืนยัน Mean-Reversion

Volume สูงผิดปกติมักเกิดพร้อม trend / breakout — volume ต่ำปกติเหมาะกับ grid มากกว่า:

```
def volume_score_for_grid(current_volume, volume_sma20): """ Volume ratio = current_volume / sma_volume_20 Score สูง = volume ปกติหรือต่ำ (grid-friendly) Score ต่ำ = volume spike (breakout risk) """ vol_ratio = current_volume / volume_sma20 if vol_ratio < 0.7: # volume ต่ำผิดปกติ (อาจไม่มี liquidity) return 40 elif vol_ratio < 1.0: # volume ปกติต่ำ = ideal return 85 elif vol_ratio < 1.5: # volume ปกติสูง return 70 elif vol_ratio < 2.0: # volume สูง = ระวัง return 45 elif vol_ratio < 3.0: # volume spike = breakout warning return 20 else: # extreme volume spike = หยุด grid return 0 # ตัวอย่าง: # Daily volume ปกติ = 50,000 BTC # วันนี้ volume = 75,000 BTC (vol_ratio = 1.5) # → score = 70 (ระวังเล็กน้อย แต่ยังเปิด grid ได้)
```

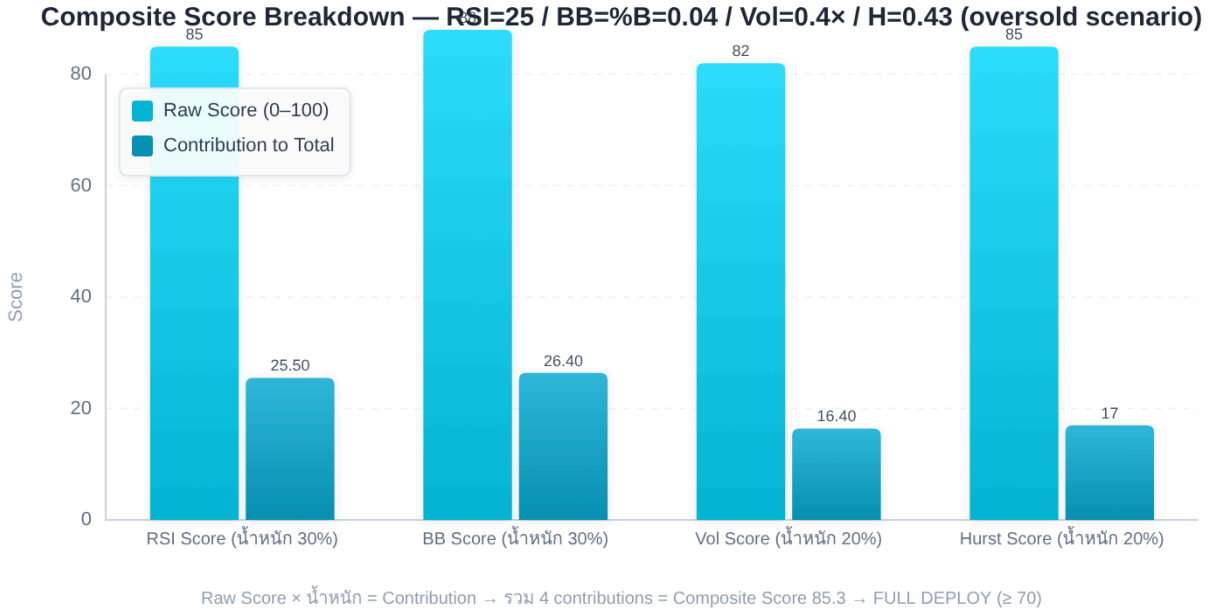
3B.5 Hurst Score — Mean-Reversion Regime Check

Hurst Exponent จาก Statarb คือ indicator หลักของ grid — ควรเป็น component สำคัญใน composite score:

```
def hurst_score_for_grid(hurst_value): """ Hurst < 0.5 = mean-reverting = grid ดี → score สูง Hurst > 0.5 = trending = grid ไม่ดี → score ต่ำ """ if hurst_value < 0.40: # strongly mean-reverting return 100 elif hurst_value < 0.45: # mean-reverting return 85 elif hurst_value < 0.50: # mildly mean-reverting return 65 elif hurst_value < 0.55: # borderline return 40 elif hurst_value < 0.60: # mildly trending return 20 else: # trending / breakout return 0 # Hurst = 0.47 → score = 85 (เปิด grid ได้ดี) # Hurst = 0.52 → score = 40 (เปิดได้ แต่ลด size) # Hurst = 0.62 → score = 0 (หยุด grid)
```

3B.6 Composite Score 0–100 — Grid Readiness Index

รวม 4 indicators เป็น Composite Score เพื่อตัดสินใจ:



Running Example: RSI=25 (oversold), BB=%B=0.04 (lower band), Vol=0.4× (quiet), Hurst=0.43 → Composite Score = 85 → FULL DEPLOY

```
WEIGHTS = { "rsi": 0.30, # 30% — mean-reversion extremity "bb": 0.30, # 30% — position + volatility "vol": 0.20, # 20% — liquidity/breakout check "hurst": 0.20, # 20% — regime check } def composite_grid_score(price, rsi, bb_upper, bb_lower, sma20, volume, volume_sma20, hurst): scores = { "rsi": rsi_score_for_grid(rsi), "bb": bb_score_for_grid(price, sma20, bb_upper, bb_lower), "vol": volume_score_for_grid(volume, volume_sma20), "hurst": hurst_score_for_grid(hurst), } composite = sum(scores[k] * WEIGHTS[k] for k in WEIGHTS) return composite, scores ACTION_TABLE = { (75, 100): ("FULL_DEPLOY", "deploy 100% grid capital"), (55, 75): ("PARTIAL_DEPLOY","deploy 60% grid capital, hold 40% standby"), (35, 55): ("MINIMAL_DEPLOY","deploy 30% grid capital, observe"), (0, 35): ("WAIT", "รอ score ดีขึ้น ไม่เปิด grid"), } ตัวอย่างผล: RSI=32 (oversold) → rsi_score = 68 %B=0.1 (near lower) → bb_score = 72 Vol ratio=0.9 → vol_score = 85 Hurst=0.46 → hurst_score = 85 Composite = 0.30×68 + 0.30×72 + 0.20×85 + 0.20×85 = 20.4 + 21.6 + 17 + 17 = 76.0 Action: FULL_DEPLOY (deploy 100%)
```

3B.7 Order Accumulation — เปิด Grid แบบ TWAP

แทนที่จะเปิด grid order ทุก level ทันที ("all-in") — Order Accumulation ค่อยๆ วาง order ทีละชั้นตามเวลา (TWAP-like)

```
กลยุทธ์ Order Accumulation: แบบ 1: Time-Spread (TWAP Grid) วาง 20% ของ orders ทุก 1 ชั่วโมง → ใช้เวลา 5 ชั่วโมงกว่าจะ full deploy ข้อดี: ถ้าราคาลงต่อหลัง open → ชั้นล่างซื้อได้ราคาดีกว่า ข้อเสีย: เสียโอกาสถ้าราคากลับขึ้นทันที แบบ 2: Price-Level Triggered เปิด order ที่ level ถัดไปก็ต่อเมื่อ level ปัจจุบัน fill แล้ว ข้อดี: ไม่ lock capital ใน pending orders ข้อเสีย: ถ้าราคากระโดด ข้าม levels → พลาด fill แบบ 3: Score-Triggered Deployment ตรวจ composite score ทุก 1 ชั่วโมง: Score > 75: deploy 20% เพิ่ม (จน 100%) Score 55–75: คง size ปัจจุบัน Score < 35: หยุด deploy เพิ่ม def order_accumulation_plan(total_capital, composite_score, current_deployed): """ คำนวณว่าควร deploy เพิ่มเท่าไหร่ """ if composite_score > 75: target_deploy = 1.00 # 100% elif composite_score > 55: target_deploy = 0.60 # 60% elif composite_score > 35: target_deploy = 0.30 # 30% else: target_deploy = 0.0 # หยุด additional = max(0, target_deploy - current_deployed) return additional * total_capital # จำนวน capital ที่ deploy เพิ่ม
```

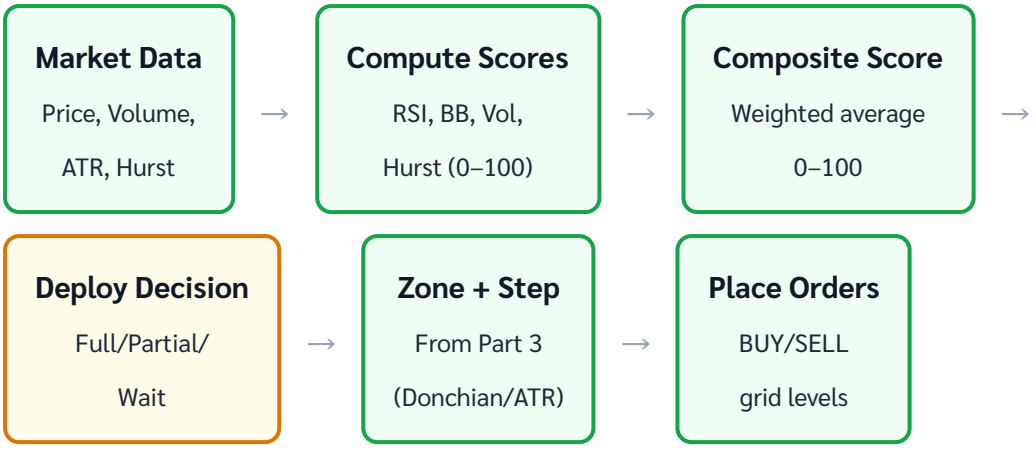
Order Accumulation ช่วยลด "Timing Risk"

ปัญหา: เปิด grid ทั้งหมดวันเดียว แต่ราคาลงต่ออีก 10% วันถัดไป → ทุก BUY order fill ที่ราคาสูงกว่าที่ควร

Order Accumulation: วาง 30% แรก → ถ้าราคาลง → วางอีก 30% ที่ระดับต่ำกว่า → average entry ดีกว่า

คล้ายกับ DCA แต่ structured: มี systematic trigger (score/time/price level) แทนที่จะเป็น random

3B.8 Filter Pipeline — Flow สมบูรณ์




```
class GridFilterPipeline: def __init__(self, zone_high, zone_low, step,
grid_capital): self.zone_high = zone_high self.zone_low = zone_low
self.step = step self.grid_capital = grid_capital self.deployed_pct =
0.0 def evaluate(self, market_data): """ market_data: dict ของ
indicators Returns: action + capital_to_deploy """ score, sub_scores =
composite_grid_score( price=market_data["price"],
rsi=market_data["rsi14"], bb_upper=market_data["bb_upper"],
bb_lower=market_data["bb_lower"], sma20=market_data["sma20"],
volume=market_data["volume"], volume_sma20=market_data["volume_sma20"],
hurst=market_data["hurst30"] ) additional = order_accumulation_plan(
self.grid_capital, score, self.deployed_pct ) action = "DEPLOY" if
additional > 0 else "HOLD" self.deployed_pct += additional /
self.grid_capital return { "score": round(score, 1), "sub_scores":
sub_scores, "action": action, "deploy_amount": additional,
"total_deployed_pct": self.deployed_pct }
```

3B.9 Running Example — Full Pipeline

🔁 **Running Example: Filter Pipeline ตัดสินใจ Deploy Grid**

Setup: BTC = $98,500 · Grid capital = $20,000 · Zone $85k–$113k · Step $1,500

Indicator	ค่า	Score (0–100)	น้ำหนัก	Contribution
RSI(14)	31 (oversold)	70	30%	21.0
BB Position + Width	%B=0.08, Width=3.2%	78	30%	23.4
Volume Ratio	0.85× (ต่ำกว่าปกติ)	85	20%	17.0
Hurst(30)	0.44 (mean-reverting)	90	20%	18.0
Composite Score			**79.4 → FULL DEPLOY**	

```
Action: FULL_DEPLOY (score 79.4 > 75) Order Accumulation Plan (ยังไม่ได้ deploy อะไรเลย): ชั่วโมงที่ 0: deploy 20% = $4,000 ชั่วโมงที่ 1: re-evaluate score → ยังสูง → deploy อีก 20% = $4,000 ชั่วโมงที่ 2–4: ทำซ้ำ → full deploy ภายใน 5 ชั่วโมง หรือถ้าใช้ One-shot (score > 75): Deploy 100% ทันที = $20,000 Entry Zone Analysis: ราคาปัจจุบัน $98,500 (ใกล้ lower half ของ zone) BUY orders: $97k, $95.5k, $94k, ..., $85k (9 levels) Capital ที่ต้องใช้ (worst case all fill): 9 × 0.009 × $91k avg = $7,371 Capital remaining ใน Standby: $20,000 − $7,371 = $12,629 (63%) Score Report: ✓ RSI oversold → BTC น่าจะ rebound → lower zone เปิดก่อน ✓ BB near lower band → mean-reversion signal ชัด ✓ Volume ต่ำ → ไม่มี breakout risk ✓ Hurst 0.44 → strongly mean-reverting → เหมาะมากสำหรับ grid entry
```

3B.10 แบบฝึกหัด

แบบฝึกหัด Part 3B

▶ ข้อ 1: RSI=72, BB %B=0.95, Volume ratio=2.5×, Hurst=0.56 — Composite Score และ Action คืออะไร?

▶ ข้อ 2: ทำไม Volume spike ถึงเป็น signal ลบสำหรับ grid (ทั้งที่ volume สูงแสดง "activity")?

▶ ข้อ 3: Composite Score = 62 · Grid capital = $10,000 · ปัจจุบัน deployed = 0% — ควร deploy เท่าไหร่?

▶ ข้อ 4 (Challenge): Order Accumulation แบบ Price-Level Triggered เหมาะกับ market condition แบบไหนที่สุด?

GRID TRADING MASTERY · PART 4

Regime Detection — เมื่อไหร่เปิด/หยุด Grid

Hurst Exponent · ADX · Bollinger Band Width · State Machine (GRID ON / CAUTION / GRID OFF)

Grid Reset + Zone Migration · Practical Watchdog System

อ้างอิง: Statarb Ch.3 (Hurst, OU Process), Statarb Ch.11 (State Machine)

แก่นของ PART นี้

Grid ทำงานได้ดีใน **mean-reverting regime** (H < 0.5) เท่านั้น — การรู้ว่า regime เปลี่ยนแปลงและตอบสนองทันทีคือความแตกต่างระหว่าง "grid ที่กำไรต่อเนื่อง" กับ "grid ที่ติดหล่ม" State Machine ช่วยเปลี่ยน grid behavior อัตโนมัติตาม regime

```
GRID ON (H < 0.50, ADX < 40 สำหรับ BTC) → CAUTION (H 0.50–0.58, ADX 40–50) → GRID OFF (H > 0.58, ADX > 50) — BTC-calibrated (Forex threshold 25/35 สูงเกินไปสำหรับ BTC)
```

4.1 Hurst Exponent — ตัววัด Mean-Reversion

Hurst Exponent H วัดว่าราคาเป็น mean-reverting (H < 0.5), random walk (H = 0.5), หรือ trending (H > 0.5):

$$H = \frac{\log(R/S)}{\log(n)} \qquad \text{โดย } R/S = \text{Rescaled Range}$$

```
Hurst Exponent สำหรับ Grid: คำนวณด้วย R/S Analysis (rolling 30 วัน): 1. เก็บ daily return r_i = log(P_i / P_{i-1}) 2. R/S ของ n วัน = (max cumulative dev − min cumulative dev) / std(returns) 3. Plot log(R/S) vs log(n) → slope = H def hurst_exponent(prices, min_lag=10, max_lag=100): """ คำนวณ Hurst จาก price series (R/S method) """ import numpy as np returns = np.diff(np.log(prices)) lags = range(min_lag, min(max_lag, len(returns) // 2)) rs_values = [] for lag in lags: # คำนวณ R/S สำหรับแต่ละ lag sub_series = [returns[i:i+lag] for i in range(0, len(returns)-lag, lag)] rs_per_sub = [] for sub in sub_series: mean_sub = np.mean(sub) dev = np.cumsum(sub - mean_sub) R = np.max(dev) - np.min(dev) S = np.std(sub, ddof=1) if S > 0: rs_per_sub.append(R / S) if rs_per_sub: rs_values.append((lag, np.mean(rs_per_sub))) if len(rs_values) < 2: return 0.5 # default random walk log_lags = np.log([x[0] for x in rs_values]) log_rs = np.log([x[1] for x in rs_values]) H = np.polyfit(log_lags, log_rs, 1)[0] return H # ตัวอย่าง: # H = 0.42 → mean-reverting → GRID ON # H = 0.51 → random walk → CAUTION # H = 0.63 → trending → GRID OFF
```

> ⚠️ **R/S Method มี Bias — แนะนำ DFA แทน**
>
> ฟังก์ชัน hurst_exponent ด้านบนใช้ R/S (Rescaled Range) ซึ่งมีปัญหา:
>
> - **Upward bias ใน short series:** N=90 วัน → ค่า H มักสูงเกิน 5–10%
> - **Standard Error สูง:** SE ≈ 0.05–0.10 สำหรับ N=90 — threshold H=0.45/0.55 ใน Part 4 อาจคลาดเคลื่อน
> - **วิธีที่ดีกว่า: DFA (Detrended Fluctuation Analysis)** — bias ต่ำกว่าและ SE ต่ำกว่าสำหรับ financial data
>
> ใช้ library `nolds`: `pip install nolds` แล้ว `import nolds; H = nolds.dfa(log_returns)`

กฎปฏิบัติ: H < 0.45 (mean-reverting ชัด) หรือ H > 0.60 (trending ชัด) เชื่อถือได้ — zone 0.45–0.60 ควรรอสัญญาณเพิ่มจาก ADX/BB

REGIME DETECTION — STATE MACHINE

GRID ON
mean-reverting
- H < 0.50 (Hurst)
- ADX < 40 (BTC)
- BB width ปกติ

Full Deploy

H rising →
← H drops

CAUTION
borderline regime
- H = 0.50–0.60
- ADX = 40–50
- Vol spike เริ่มขึ้น

Reduce 50% size

ADX spike →
← ADX drops

GRID OFF
trending / breakout
- H > 0.60
- ADX > 50
- ราคาออกนอก Donchian

Pause — Cancel all

ตัวอย่าง Hurst timeline — สี = regime ที่ active

GRID ON	CAUTION	GRID OFF	GRID ON (กลับมา)

เดือน 1 — เดือน 3 — เดือน 6

H < 0.50 = grid เต็ม · 0.50–0.60 = ลด size · H > 0.60 = หยุด grid (ADX threshold ปรับสำหรับ BTC)

4.2 ADX — ยืนยัน Trend Strength

ADX (Average Directional Index) วัดความแรงของ trend โดยไม่บอกทิศทาง — ใช้ยืนยัน Hurst signal:

$$\text{ADX} = \text{Smoothed MA of DX} \qquad \text{DX} = \frac{|+DI - -DI|}{+DI + -DI} \times 100$$

ADX Threshold สำหรับ BTC ≠ Forex/Equity

ADX 25 = "trend" ใน textbook ถูก calibrate สำหรับ Forex และ Equity — BTC มี volatility สูงกว่า 3–5×

BTC ADX มักอยู่ที่ 30–50 แม้ใน sideways market — threshold ที่เหมาะสมกว่า:

ADX Range	Forex/Equity	BTC (แนะนำ)
<25	Sideways	Sideways / Weak trend
25–40	Trend emerging	ยังคือ Sideways สำหรับ BTC
40–50	Strong trend	Trend emerging (CAUTION)
>50	Very strong	Strong trend (GRID_OFF candidate)

Backtest บน BTC data จริง (2020–2024) แนะนำ threshold ≈ 40–45 สำหรับ GRID_OFF trigger

ADX Value	ความหมาย	Grid Action
< 15	ไม่มี trend (choppy)	GRID ON — ideal condition
15–25	trend อ่อน	GRID ON (ระวัง upper zone)
25–40	trend ปานกลาง (BTC: ยัง sideways)	CAUTION — ลด size 50%
> 40 (BTC: > 45)	trend แรง	GRID OFF — pause ทั้งหมด

```
def compute_adx(highs, lows, closes, period=14): """ คำนวณ ADX จาก
OHLC data """ import numpy as np tr_list, dm_plus, dm_minus = [], [],
[] for i in range(1, len(closes)): high_diff = highs[i] - highs[i-1]
```

```
low_diff = lows[i-1] - lows[i] dm_plus.append(max(high_diff, 0) if
high_diff > low_diff else 0) dm_minus.append(max(low_diff, 0) if
low_diff > high_diff else 0) tr = max(highs[i] - lows[i], abs(highs[i]
- closes[i-1]), abs(lows[i] - closes[i-1])) tr_list.append(tr) # Smooth
ด้วย Wilder MA def wilder_ma(data, n): result = [sum(data[:n]) / n] for
i in range(n, len(data)): result.append(result[-1] * (n-1)/n + data[i]
/ n) return result atr14 = wilder_ma(tr_list, period) dmp14 =
wilder_ma(dm_plus, period) dmm14 = wilder_ma(dm_minus, period) di_plus
= [100 * p / a for p, a in zip(dmp14, atr14)] di_minus = [100 * m / a
for m, a in zip(dmm14, atr14)] dx_list = [100 * abs(p - m) / (p + m)
for p, m in zip(di_plus, di_minus)] adx = wilder_ma(dx_list, period)
return adx[-1]
```

4.3 Bollinger Band Width — Squeeze Detection

BB Width ต่ำ (squeeze) = ราคากำลัง consolidate → mean-reversion regime → grid ทำงานดี BB Width พุ่งสูง = volatility expansion = breakout → grid อาจติดหล่ม

```
BB Width = (BB_upper − BB_lower) / SMA(20) Thresholds สำหรับ Grid:
BB_width < 0.02 (2%) → Squeeze — GRID ON (ideal) BB_width 0.02–0.04 →
Normal — GRID ON BB_width 0.04–0.08 → Expanding — CAUTION BB_width >
0.08 (8%) → Wide — GRID OFF (high vol) ตัวอย่าง BTC ที่ $100k: BB_upper =
$104,000, BB_lower = $96,000 BB_width = ($104k − $96k) / $100k = 8% →
GRID OFF threshold BB_upper = $101,500, BB_lower = $98,500 BB_width =
$3k / $100k = 3% → Normal → GRID ON
```

4.4 State Machine — GRID ON / CAUTION / GRID OFF

GRID ON	CAUTION	GRID OFF
Full Deploy 100%	Reduce 50% size	Pause / Cancel all
H < 0.50	H 0.50–0.58	H > 0.58
ADX < 40 (BTC)	ADX 40–50 (BTC)	ADX > 50 (BTC)
BB_width < 4%	BB_width 4–8%	BB_width > 8%

```
class GridStateMachine: STATES = ["GRID_ON", "CAUTION", "GRID_OFF"] def
__init__(self): self.state = "GRID_ON" self.state_history = [] def
evaluate(self, hurst, adx, bb_width): """ Determine grid state from 3
indicators. Requires 2-of-3 indicators to agree before state change
(noise filter). """ signals = { "hurst": "OFF" if hurst > 0.58 else
("CAUTION" if hurst > 0.50 else "ON"), "adx": "OFF" if adx > 50 else
("CAUTION" if adx > 40 else "ON"), # BTC-calibrated: CAUTION=40, OFF=50
(textbook Forex threshold=25/35 ไม่ถูกสำหรับ BTC) "bb": "OFF" if bb_width
> 0.08 else ("CAUTION" if bb_width > 0.04 else "ON"), } counts = {"ON":
0, "CAUTION": 0, "OFF": 0} for sig in signals.values(): counts[sig] +=
1 # Majority vote (2 of 3) if counts["OFF"] >= 2: new_state =
"GRID_OFF" elif counts["CAUTION"] >= 2 or (counts["OFF"] == 1 and
counts["CAUTION"] == 1): new_state = "CAUTION" else: new_state =
"GRID_ON" if new_state != self.state:
self.state_history.append((self.state, new_state)) self.state =
new_state return self.state, signals def get_size_multiplier(self):
return {"GRID_ON": 1.0, "CAUTION": 0.5, "GRID_OFF": 0.0}[self.state]
```

Hysteresis — ป้องกัน State Flip ถี่เกินไป

ปัญหา: ถ้า Hurst แกว่งรอบ 0.50 (บางครั้ง 0.49 บางครั้ง 0.51) → state เปลี่ยนทุกชั่วโมง = สั่ง cancel/open orders บ่อยเกิน = fee เสียเยอะ

Solution: Hysteresis (Dead Band)

- เปลี่ยนจาก GRID_ON → CAUTION ต้องมี H > 0.52 (ไม่ใช่ 0.50)
- เปลี่ยนกลับจาก CAUTION → GRID_ON ต้องมี H < 0.47 (ต่ำกว่า threshold)

- Dead band = 0.05 รอบ threshold ทำให้ state stable กว่า

4.5 Grid Reset — ย้าย Zone เมื่อ Regime กลับมา

เมื่อ Grid OFF แล้ว regime กลับมาเป็น mean-reverting: ราคาอาจเปลี่ยนไปมากจนต้อง reset zone ใหม่

```
Grid Reset Decision: สถานการณ์ที่ต้อง Reset (ไม่ใช่แค่ restart): 1. ราคาหลุดออกนอก original zone (ต่ำกว่า zone_low หรือสูงกว่า zone_high) 2. ราคา mean เปลี่ยนไป > 15% จากตอนตั้ง grid เดิม 3. ผ่านไปนานกว่า 30 วัน (regime อาจเปลี่ยนโครงสร้าง) def should_reset_zone(original_zone_mid, current_price, original_zone_low, original_zone_high, days_since_open): """ Returns True ถ้าควร reset zone ใหม่ """ price_drift = abs(current_price - original_zone_mid) / original_zone_mid out_of_zone = current_price < original_zone_low or current_price > original_zone_high large_drift = price_drift > 0.15 stale = days_since_open > 30 return out_of_zone or large_drift or stale Grid Reset Steps: 1. Cancel orders ทั้งหมดที่ pending 2. ขาย inventory บางส่วน (ถ้า unrealized loss < 15%) หรือ รักษา inventory ไว้ถ้า BTC (ถือได้) 3. คำนวณ zone ใหม่จาก Donchian + ATR ปัจจุบัน 4. คำนวณ Q ใหม่ตาม capital ที่เหลือ 5. วาง orders ใหม่ในชั่วโมงที่ score > 55
```

4.6 Zone Migration — Shift Zone ตาม Trend

แทนที่จะ reset ทั้งหมด บางครั้งแค่ "เลื่อน zone" ขึ้นหรือลงตาม EMA ก็เพียงพอ:

```
Zone Migration (Trailing Zone): ทุก 24 ชั่วโมง: new_zone_mid = EMA(20) ของ
current price zone_high = new_zone_mid × (1 + zone_half_width_pct)
zone_low = new_zone_mid × (1 - zone_half_width_pct) ตัวอย่าง: BTC ขยับจาก
$100k → $110k ใน 1 สัปดาห์ Old zone: $90k–$115k (mid=$102.5k) EMA(20) =
$107k zone_half_width = 15% (ตาม Donchian width) New zone: $107k × 0.85
= $90.95k → $107k × 1.15 = $123.05k การย้าย: + Cancel orders ที่อยู่นอก new
zone + เพิ่ม orders ใน upper zone ที่เพิ่งเข้ามา + inventory ที่ BUY ไปแล้วในชั้นเก่า
→ ยังคง TP เดิม Zone Migration ดีกว่า Grid Reset เมื่อ: - ราคา drift ช้า
(trending แต่ไม่เร็ว) - Inventory ยังไม่ได้ขาดทุนมาก - ไม่อยากเสีย inventory ที่
BUY ไปแล้ว
```

4.7 Practical Watchdog System

ระบบ watchdog ทำงาน background ตรวจ regime ทุก 1 ชั่วโมงและส่ง alert เมื่อต้องปรับ:

```
class GridWatchdog: def __init__(self, grid_bot,
check_interval_hours=1): self.grid = grid_bot self.state_machine =
GridStateMachine() self.interval = check_interval_hours self.alert_log
= [] def run_checks(self, market_data): """ ทำงานทุก 1 ชั่วโมง (หรือตาม
interval) """ hurst = compute_hurst(market_data["prices_30d"]) adx =
compute_adx(market_data["highs"], market_data["lows"],
market_data["closes"]) bb_width =
compute_bb_width(market_data["prices_20d"]) new_state, signals =
self.state_machine.evaluate(hurst, adx, bb_width) size_mult =
self.state_machine.get_size_multiplier() actions = [] if new_state ==
"GRID_OFF": actions.append({"action": "CANCEL_ALL_ORDERS", "reason":
"regime trending"}) self.alert_log.append(f"GRID_OFF: H={hurst:.2f}
ADX={adx:.1f} BB={bb_width:.2%}") elif new_state == "CAUTION":
current_size = self.grid.get_deployed_size() if current_size > 0.5:
reduce_amount = current_size - 0.5 actions.append({"action":
"REDUCE_SIZE", "amount": reduce_amount, "reason": "caution regime"})
elif new_state == "GRID_ON": current_size =
self.grid.get_deployed_size() if current_size < 1.0:
actions.append({"action": "INCREASE_SIZE", "target": 1.0, "reason":
"regime cleared"}) # ตรวจ zone migration if
should_reset_zone(self.grid.zone_mid, market_data["current_price"],
self.grid.zone_low, self.grid.zone_high, self.grid.days_running):
actions.append({"action": "ZONE_MIGRATION", "reason": "price drifted"})
return {"state": new_state, "signals": signals, "actions": actions}
Watchdog Alert Types: 1. REGIME_CHANGE → ส่ง LINE/Telegram notify 2.
ZONE_BREACH → ราคาหลุด zone boundary 3. DD_WARNING → Drawdown > 4% (ดู
Part 5) 4. DD_HALT → Drawdown > 8% → cancel all 5. RESET_REQUIRED →
Zone stale > 30 วัน
```

4.8 Running Example — Regime Change Sequence

🔁 **Running Example: BTC Regime เปลี่ยนจาก Mean-Reverting → Trending**

Timeline: 3 สัปดาห์ · Grid เปิดที่ $95k–$115k · Step $2k · Capital $25,000

วันที่	BTC Price	Hurst (30d)	ADX (14d)	BB Width	State	Action
Day 1	$103k	0.44	18	3.2%	**GRID ON**	Deploy 100%
Day 5	$106k	0.47	21	3.8%	**GRID ON**	ทำงานปกติ
Day 10	$112k	0.52	27	5.1%	**CAUTION**	Reduce to 50%
Day 12	$118k	0.59	34	7.8%	**GRID OFF**	Cancel all orders
Day 15	$125k	0.61	38	9.2%	**GRID OFF**	รอ regime กลับ
Day 18	$121k	0.55	29	6.1%	**CAUTION**	รอ H < 0.50
Day 21	$118k	0.47	22	4.0%	**GRID ON**	Zone Migration → $105k–$135k

```
สรุปผลระหว่าง Day 1–21: Grid ON (Day 1–9): +$180 cashflow (ทำงานปกติ) CAUTION (Day 10–11): +$20 cashflow (ลด size 50%) GRID OFF (Day 12–17): $0 cashflow (ถือ inventory, รอ) Zone Migration (Day 21): เปิด grid ใหม่ที่ $105k–$135k Inventory ที่ถือ: BUY ที่ $98k, $100k, $102k, $104k (ตอน grid เปิด) ราคาปัจจุบัน $118k → inventory กำไร unrealized = 4 × 0.01 BTC × $16k avg = $640 ถ้าไม่มี state machine: → grid ยังเปิด order ที่ $120k, $122k, $124k ระหว่าง uptrend → ขาย inventory เร็วเกินไป
```

```
(SELL TP ที่ $100k, $102k หรือต่ำกว่าราคาตลาด) → พลาด upside gain ของ
BTC
```

4.9 แบบฝึกหัด

แบบฝึกหัด Part 4

▶ ข้อ 1: Hurst=0.46, ADX=28, BB_width=3.5% — State Machine ส่งผลให้ state เป็นอะไร? (2-of-3 majority vote)

▶ ข้อ 2: State เปลี่ยนจาก GRID_ON → GRID_OFF โดยตรง (ข้าม CAUTION) — เป็นไปได้ไหม? เกิดอย่างไร?

▶ ข้อ 3: Grid run 35 วัน · Original zone_mid = $100k · Current price = $118k · Zone $88k–$115k — ควร reset หรือ migrate?

▶ ข้อ 4 (Challenge): ออกแบบ Hysteresis rules สำหรับ Hurst transitions เพื่อลด false state changes

GRID TRADING MASTERY · PART 5

Sizing & Risk — Kelly, Position Limit, Drawdown Control

Kelly Criterion บน Grid · Fractional Kelly · Max Notional ต่อ Level · DD Halt

อ้างอิง: Statarb Ch.12 (Kelly, Position Sizing)

แก่นของ PART นี้

Sizing ที่ดีไม่ได้ทำให้กำไรเพิ่มมากขึ้น แต่ป้องกัน **ruin** — grid ที่กำไรได้ต่อเนื่องจะหยุดถ้า capital หมดก่อน Kelly ช่วยตอบ "ควร deploy กี่% ของ capital ต่อกลยุทธ์ grid รวม (ไม่ใช่ต่อ level)" เพื่อ maximise long-run geometric growth

```
Kelly f* = (p × b − q) / b · Fractional = 0.25 × f* · Q
= f* × capital / (N × P)
```

5.1 Kelly Criterion บน Grid — แนวคิด

Kelly Criterion ดั้งเดิม (จาก Statarb Ch.12) ให้ optimal fraction ของ capital ที่ควร risk ต่อ trade โดยหลักการคือ **maximise E[log(wealth)]** — ไม่ใช่ maximize expected profit ต่อรอบ แต่ maximize long-run geometric growth rate:

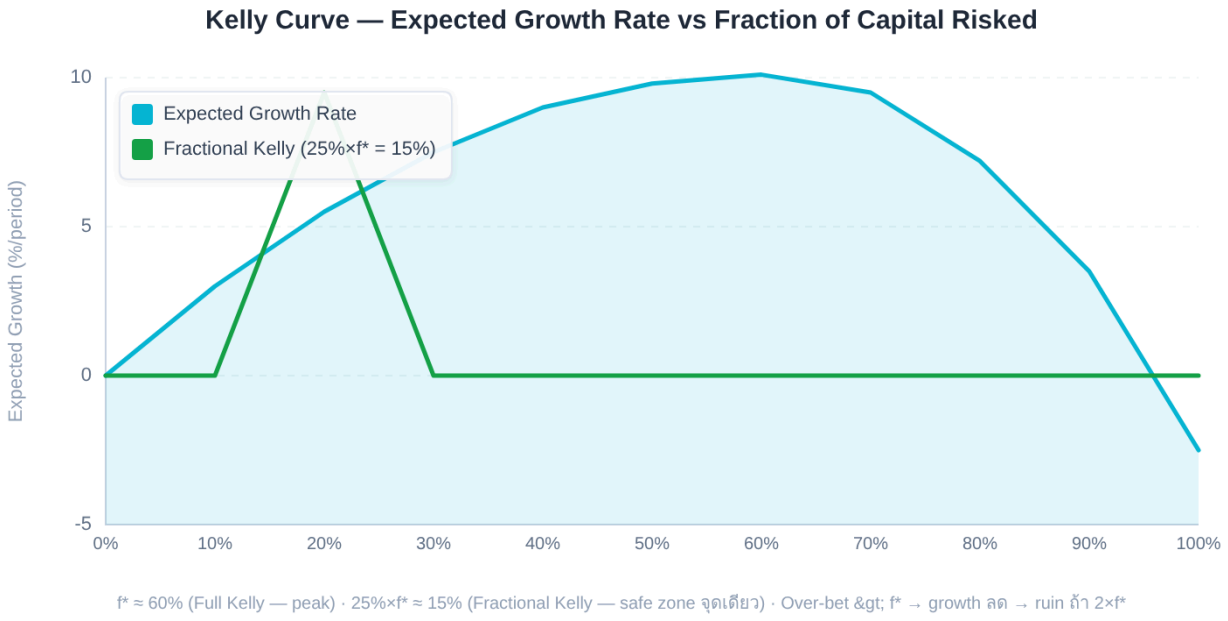
$$f^* = \frac{p \cdot b - q}{b}$$

โดย: p = ความน่าจะเป็นที่ trade นั้นกำไร, b = อัตราส่วน win/loss, $q = 1 - p$

สำหรับ Grid Trading: "1 trade" = 1 grid cycle (BUY fill → SELL fill ที่ level ถัดไป)

```
การประมาณ Kelly สำหรับ Grid: กรณี BTC/USDT Bybit: P(cycle completes) = p
← ความน่าจะเป็นที่ราคาขึ้นถึง TP หลัง BUY fill ประมาณ: สำหรับ mean-reverting
asset (H<0.5), p ≈ 0.55–0.65 Win amount per cycle = Q × Step − Fees =
$18 Loss amount per cycle = ไม่มี "loss" ต่อ cycle ใน Close System (loss =
opportunity cost ของ capital ที่ lock) ปัญหา: Kelly classic ไม่ fit grid
โดยตรง เพราะ grid ไม่มี "cut loss" แนวทางประยุกต์: → ใช้ Kelly กำหนด "ขนาด
grid รวม" vs total capital (ไม่ใช่ต่อ cycle) → f* = optimal fraction ของ
net worth ที่ควรใส่ใน grid portfolio
```

> ⚠️ **p=0.55–0.65 เป็นสมมติฐาน ไม่ใช่ความจริงที่รับประกัน**
>
> ค่า p ที่ถูกต้องขึ้นอยู่กับ: asset ที่เทรด, ช่วงเวลา, regime ปัจจุบัน — ค่าที่ backtest ได้จาก 90 วันมี Standard Error สูง
>
> ตัวอย่าง error propagation: p จริง = 0.60 แต่ estimate error ±0.05 → f* error ±30–40%
>
> **แนะนำ: ใช้ Fractional Kelly (25% × f*) เสมอ** เพื่อ buffer uncertainty ใน p



Kelly Curve: Growth Rate ขึ้นถึงจุดสูงสุดที่ f* แล้วตกลงเมื่อ bet มากเกินไป — Fractional Kelly (25%×f*) อยู่ในโซน safe (ในกราฟนี้ f*≈60% เป็นตัวเลข illustrative — ค่าจริงขึ้นกับ backtest ของแต่ละ grid)

5.2 Kelly สำหรับ Grid Portfolio (Practical Approach)

วิธีง่ายสุด: backtest grid ด้วยข้อมูลย้อนหลัง → วัด win rate และ profit ratio ต่อ "month" → apply Kelly

```
ตัวอย่างจาก Backtest (90 วัน): เดือนที่ผลรวม > 0: 2.5 เดือน จาก 3 → p = 0.83 (83%) เดือนที่ผลรวม < 0: 0.5 เดือน → q = 0.17 Avg monthly win = +$450 บน $10,000 capital (+4.5%) Avg monthly loss = −$200 บน $10,000 capital (−2.0%) b = avg_win / avg_loss = 450 / 200 = 2.25 Kelly f* = (0.83 × 2.25 − 0.17) / 2.25 = (1.8675 − 0.17) / 2.25 = 1.6975 / 2.25 ≈ 0.755 (75.5%) Fractional Kelly (25%): f_safe = 0.755 × 0.25 = 0.189 ≈ 19% → ควรใส่เงินในกลยุทธ์ grid นี้ไม่เกิน 19% ของ net worth ถ้า net worth = $100,000: Grid capital = $100,000 × 19% = $19,000
```

Sample Size Warning

90 วัน × ~3 fills/วัน = ~270 trades — ยังน้อยเกินสำหรับ p estimate ที่แม่นยำ

Standard Error of proportion: SE = √(p(1−p)/n) *(ตัวอย่างทั่วไป p=0.60: SE = √(0.6×0.4/270) ≈* ***±3%****)* — ค่า p=0.83 ในตัวอย่าง backtest ข้างต้นให้ SE ≈ ±2.3%

ใช้ 6–12 เดือนข้อมูล (500+ trades) เพื่อ SE < 2% — หรือใช้ Fractional Kelly ตลอด

Kelly Sensitivity Analysis — ผลกระทบของ p ที่คาดเคลื่อน

p (win rate)	b (win/loss ratio)	f* (Full Kelly)	25% × f* (Fractional)	Max Q ต่อ Capital
0.50 (coin flip)	1.0	0%	0%	ไม่เล่น
0.55	1.0	10%	2.5%	ต่ำมาก
0.60	1.0	20%	5%	ปานกลาง
0.65	1.0	30%	7.5%	สูง
0.55 (overestimate)	1.0	10%	2.5%	—

หาก p จริงต่ำกว่าที่ estimate 5% → f ลดลงครึ่งหนึ่ง — ใช้ Fractional Kelly ป้องกัน overbet*

ทำไมต้อง Fractional Kelly (25% ของ f*)?

Kelly เต็มๆ (f* ≈ 75% จากตัวอย่าง 5.2) ให้ growth สูงสุดทางทฤษฎี แต่ในทางปฏิบัติ:

- p และ b ที่ estimate มีความผิดพลาด — Kelly sensitive มากต่อ error
- Kelly เต็มทำให้ variance สูงมาก — drawdown ลึก
- Fractional Kelly 25% ให้ ~80% ของ growth แต่ variance ลดลง 75%

ดูรายละเอียดใน Statarb Ch.12

5.3 Q (Quantity) ต่อ Level — จาก Kelly

```
คำนวณ Q ต่อ Level จาก Kelly fraction: Grid capital = net_worth × f_safe
= $100k × 0.19 = $19,000 N levels (total) = (Zone_high − Zone_low) /
Step = ($110k − $90k) / $2k = 10 levels Avg price = ($90k + $110k) / 2
= $100k Q per level = grid_capital / (N × avg_price) = $19,000 / (10 ×
$100,000) = $19,000 / $1,000,000 ≈ 0.019 BTC ≈ 0.02 BTC (round down)
ตรวจสอบ: 10 levels × 0.02 BTC × $100k = $20,000 ≈ $19,000 ✓ Capital ที่ใช้
จริง (ถ้าซื้อครบ floor): Σ Q × P_i = 0.02 × ($100k + $98k + ... + $90k) +
sell side = 0.02 × $940k (10 levels avg) = $18,800 ≈ $19,000 ✓
```

5.4 Max Notional ต่อ Level — Position Limit

นอกจาก Kelly ที่กำหนด grid size รวม ควรมี position limit ต่อ level เพื่อป้องกันความเสี่ยงกระจุก:

```
Position Limit Rules: Rule 1: Max Notional ต่อ Level ≤ 5% ของ total capital Max_notional_level = $100k × 5% = $5,000 per level Max Q per level = $5,000 / P = $5,000 / $100,000 = 0.05 BTC Rule 2: ในช่วงที่ Hurst สูง (trending) → ลด limit ลงอีก ถ้า H > 0.55: max_q × 0.7 (ลด 30%) ถ้า H > 0.60: max_q × 0.4 (ลด 60%) Rule 3: Concentration cap — ไม่ให้ 3 levels ติดกันมี Q รวม > 10% ของ capital (ป้องกัน bottom-heavy bloating ที่เกินไป) def check_position_limits(grid_levels, total_capital, hurst): max_per_level = total_capital * 0.05 if hurst > 0.60: max_per_level *= 0.4 elif hurst > 0.55: max_per_level *= 0.7 for level in grid_levels: notional = level.qty * level.price if notional > max_per_level: level.qty = max_per_level / level.price # trim return grid_levels
```

5.5 Drawdown Control — DD Halt

Drawdown (DD) = peak portfolio value ลบด้วย current value ÷ peak:

$$\mathrm{DD}_t = \frac{P_{\text{peak}} - P_t}{P_{\text{peak}}}$$

```
DD Halt System: def check_drawdown_halt(portfolio_value_history, halt_threshold=-0.08): """ ถ้า DD < threshold (เช่น −8.5% < −8%) → หยุด grid ทั้งหมด threshold = -0.08 (−8% ของ peak, ปรับได้ตาม risk tolerance) """ peak = max(portfolio_value_history) current = portfolio_value_history[-1] dd = (current - peak) / peak # จะเป็นลบเมื่อ current < peak if dd < halt_threshold: return "HALT", dd, peak elif dd < halt_threshold * 0.5: # -4% = warning zone return "WARNING", dd, peak return "OK", dd, peak Halt ทำอะไร: 1. Cancel orders ทั้งหมด 2. ไม่เปิด order ใหม่ (แต่ไม่ขาย inventory — Close System) 3. รอให้ DD ฟื้นกลับ > threshold/2 ก่อน restart 4. Log และ alert Resume Condition: DD ฟื้นกลับ > −4% (ครึ่งหนึ่งของ halt threshold) + H < 0.55 (กลับมา mean-reverting) + ผ่านไปอย่างน้อย 24 ชั่วโมง (cooling off period)
```

Risk Level	DD Halt Threshold	ลักษณะ
Conservative	−5%	หยุดเร็ว ทำกำไรน้อยลง แต่ปลอดภัย
Moderate (แนะนำ)	−8%	สมดุล — ให้ grid ทำงานในความผันผวนปกติ
Aggressive	−15%	ทน volatility ได้มาก แต่ DD ลึก
Close System Pure	ไม่มี (ถึง floor)	รอ BTC กลับเสมอ — ต้องมี capital พอ

5.6 Running Example — Full Sizing Calculation

🔁 **Running Example: Sizing BTC Grid จาก Kelly**

หมายเหตุ: ตัวอย่างนี้ใช้ net worth $200,000 · p=0.78 · b=1.9 ซึ่งต่างจากตัวอย่างใน 5.2 ($10k · p=0.83 · b=2.25) — เพื่อ illustrate ด้วยพารามิเตอร์ที่สมจริงกับ account ขนาดใหญ่

โจทย์: Net worth $200,000 · Backtest 6 เดือน: p=0.78 · b=1.9 · Grid Zone $90k–$110k · Step $2k

```
Step 1: คำนวณ Kelly f* f* = (0.78 × 1.9 − 0.22) / 1.9 = (1.482 −
0.22) / 1.9 = 1.262 / 1.9 ≈ 0.664 (66.4%) Step 2: Fractional Kelly
(25%) f_safe = 0.664 × 0.25 = 0.166 = 16.6% Step 3: Grid Capital
Grid capital = $200,000 × 16.6% = $33,200 Step 4: Q per Level N =
(110k − 90k) / 2k = 10 levels Avg price = $100k Q = $33,200 / (10
× $100k) = 0.0332 BTC ≈ 0.033 BTC Step 5: ตรวจ Position Limit (5%
rule) Max notional = $200k × 5% = $10,000 Q max = $10,000 / $100k
= 0.10 BTC Q actual = 0.033 < 0.10 ✓ (ผ่าน) Step 6: Zone Waterfall
Active (50%): $16,600 → deploy as orders Standby (30%): $9,960 →
lending/yield Floor (20%): $6,640 → reserve Summary: Grid capital:
$33,200 (16.6% ของ net worth) Q per level: 0.033 BTC Daily P&L
estimate: 1.5 รอบ × ($66 − fee) ≈ $90/วัน = 0.27%/วัน
```

5.7 แบบฝึกหัด

แบบฝึกหัด Part 5

▶ ข้อ 1: Backtest: p=0.70, win=$300, loss=$150 — Kelly f* เป็นเท่าไหร่? ควรใช้ capital เท่าไหร่จาก $50k?

▶ ข้อ 2: Grid capital $10k · Zone $85k–$115k · Step $2k · BTC = $100k — Q ต่อ level เท่าไหร่?

▶ ข้อ 3: Portfolio peak = $115,000 · Current = $105,000 · Halt threshold = –8% — ควร halt หรือไม่?

▶ ข้อ 4 (Challenge): Hurst = 0.62 · กำลังจะเปิด grid ด้วย Q = 0.03 BTC — ควรปรับ Q อย่างไร?

GRID TRADING MASTERY · PART 6

Pair Grid + Relative Value Switching

BTC/ETH Spread Grid · Cointegration Test · Relative Value Signal

Multi-Asset Grid Portfolio · Switching Between Instruments

อ้างอิง: Statarb Ch.4–6 (Cointegration, Spread, OU Process)

แก่นของ PART นี้

Pair Grid ไม่ใช้ grid บน BTC price โดยตรง แต่ใช้ grid บน **spread ระหว่างสองสินทรัพย์** (เช่น BTC/ETH ratio) — spread cointegrated มี mean-reversion ที่แข็งแกร่งกว่าราคาเดี่ยว + ไม่ขึ้นกับทิศทางตลาด

```
Spread = log(BTC/USDT) − β × log(ETH/USDT) · Grid on
spread reversion to mean
```

6.1 ทำไม Pair Grid?

ปัญหาของ Single-Asset Grid: ถ้า BTC uptrend → grid ขาย inventory เร็ว (ดี) แต่ถ้า BTC downtrend → inventory สะสม (ไม่ดี)

Pair Grid แก้ด้วย: สร้าง grid บน spread ที่ cointegrated — spread revert ไม่ว่า BTC จะขึ้นหรือลง (ตราบใดที่ BTC/ETH ratio กลับสู่ค่าเฉลี่ย)

BTC/USDT
Long BTC ถ้า spread ต่ำกว่า mean
(BTC "ถูก" เทียบ ETH)

↔

SPREAD = log(BTC) − β·log(ETH)
Grid บน spread
BUY spread ต่ำ → SELL spread สูง

↔

ETH/USDT
Short ETH ถ้า spread ต่ำกว่า mean
(ETH "แพง" เทียบ BTC)

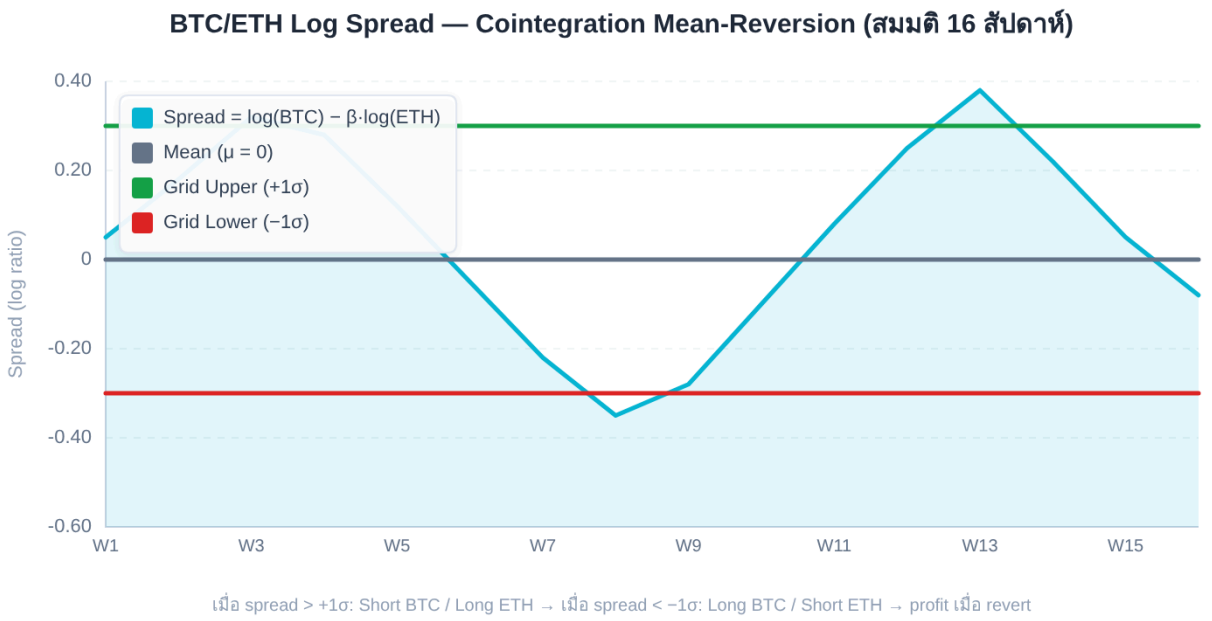
6.2 Cointegration — พื้นฐาน Pair Grid

Correlation ≠ Cointegration

BTC และ ETH อาจมี correlation สูง (ขึ้นลงพร้อมกัน) แต่นั่นไม่รับประกันว่า spread จะ mean-revert — ต้องผ่าน Cointegration Test จึงจะ trade ได้อย่างมั่นใจ

Cointegration (จาก Statarb Ch.5) หมายความว่า: แม้ราคาสองตัวจะ random walk แต่ linear combination ของมันเป็น stationary:

$$\text{Spread}_t = \log(P_{BTC,t}) - \beta \cdot \log(P_{ETH,t}) - \alpha \sim \text{Stationary}$$



BTC/ETH Spread (log ratio) — mean-revert กลับสู่ค่าเฉลี่ยอยู่เสมอ → Grid บน spread ทำกำไรได้ต่อเนื่อง

```
ทดสอบ Cointegration สำหรับ Pair Grid: def test_cointegration(btc_prices,
eth_prices, significance=0.05): """ Engle-Granger 2-step cointegration
test Returns: (is_cointegrated, beta, alpha, adf_stat, p_value) """
import numpy as np from scipy import stats log_btc = np.log(btc_prices)
log_eth = np.log(eth_prices) # Step 1: OLS regression log_BTC = alpha +
beta × log_ETH slope, intercept, r, p, se = stats.linregress(log_eth,
log_btc) beta = slope alpha = intercept # Step 2: คำนวณ spread
(residuals) spread = log_btc - beta * log_eth - alpha # Step 3: ADF
test บน spread adf_stat = adf_test(spread) # implementation ดูใน statarb
p_value = adf_p_value(adf_stat, len(spread)) is_cointegrated = p_value
```

```
< significance return is_cointegrated, beta, alpha, spread # ตัวอย่าง BTC/ETH (180 วัน): # OLS: log(BTC) = 2.14 + 0.82 × log(ETH) → beta = 0.82 # ADF stat = −3.47 (critical at 5% = −2.89) → p < 0.05 → cointegrated ✓ # Half-life of spread reversion = ln(2) / |AR coeff| ≈ 8 days
```

Johansen Test — ดีกว่า Engle-Granger สำหรับ Pair Grid

Engle-Granger มี low statistical power (พลาด cointegration จริงบ่อย) — ใช้ Johansen เป็น confirmation:

```
from statsmodels.tsa.vector_ar.vecm import coint_johansen def johansen_cointegration(btc_prices, eth_prices, significance_level=0.05): """ Johansen test — ดีกว่า Engle-Granger สำหรับ 2+ variables Returns True ถ้า cointegrated """ import numpy as np data = np.column_stack([np.log(btc_prices), np.log(eth_prices)]) result = coint_johansen(data, det_order=0, k_ar_diff=1) # Trace statistic: critical values at 5% # result.cvt[:, 1] = 5% critical values trace_stat = result.lr1[0] # rank=0 hypothesis crit_val = result.cvt[0, 1] # 5% critical value is_cointegrated = trace_stat > crit_val return { "cointegrated": is_cointegrated, "trace_stat": trace_stat, "crit_val_5pct": crit_val, "eigenvectors": result.evec # loading vectors } # ใช้ทั้งสอง test ร่วมกัน: # ถ้า Engle-Granger AND Johansen ทั้งคู่ = cointegrated → เชื่อถือได้สูง # ถ้าแค่อันเดียว → ระวัง (อาจเป็น spurious)
```

⚠️ **Cointegration Break Risk**

Cointegration ไม่ถาวร — ความสัมพันธ์ระหว่าง BTC/ETH อาจพังได้เมื่อ:

- Structural regime change (เช่น ETH ถูก SEC classify เป็น security)
- Exchange listing/delisting ของ asset ใดอีกตัว
- Regulatory event ที่กระทบ asset ตัวใดตัวหนึ่งเฉพาะ

เมื่อ cointegration พัง: spread ไม่ revert → Pair Grid สะสม unrealized loss ไม่จำกัด ต้องปิด position ทันที

Re-validate ทุก 30 วัน: run ADF test + Johansen test ใหม่ ถ้า p-value > 0.05 → หยุด Pair Grid ทันที

```
def rolling_cointegration_check(btc_prices, eth_prices, window=60,
step=5): """ ตรวจ cointegration แบบ rolling — ความสัมพันธ์อาจเปลี่ยนตาม
regime window: วันที่ใช้ test step: ขยับ window ทีละกี่วัน """ from
statsmodels.tsa.stattools import coint results = [] for start in
range(0, len(btc_prices) - window, step): end = start + window _,
pvalue, _ = coint(btc_prices[start:end], eth_prices[start:end])
results.append({ "start_day": start, "end_day": end, "p_value": pvalue,
"cointegrated": pvalue < 0.05 }) # นับ % ของ windows ที่ cointegrated
cointegrated_pct = sum(r["cointegrated"] for r in results) /
len(results) return results, cointegrated_pct # ถ้า cointegrated_pct <
70% → relationship ไม่เสถียร → ระวัง Pair Grid
```

6.3 Spread Grid — โครงสร้าง

เมื่อรู้ว่า spread cointegrated แล้ว วาง grid บน spread โดยตรง:

```
Spread Grid Setup: spread_mean = 0.0 (โดย construction จาก OLS)
spread_std = 0.045 (จาก historical spread distribution) zone_width = 3σ
each side = ±0.135 # Grid levels (spread units) spread_levels =
[-0.135, -0.090, -0.045, 0.0, +0.045, +0.090, +0.135] = [-3σ, -2σ, -1σ,
0, +1σ, +2σ, +3σ ] # แต่ละ level: ถ้า spread < level → BUY BTC + SHORT
ETH # ถ้า spread > level → SELL BTC + LONG ETH def
pair_grid_signal(current_spread, spread_levels, prev_spread, beta,
btc_price, eth_price, q_btc): """ ตรวจสอบว่า spread ข้ามผ่าน level ไหน →
ส่ง orders """ orders = [] q_eth = q_btc * btc_price / eth_price * beta
# ETH qty ที่ match BTC for level in spread_levels: if prev_spread >
level and current_spread <= level: # Spread ลงผ่าน level → spread กำลังต่ำ
→ BUY spread (Long BTC, Short ETH) orders.append({"action":
"BUY_SPREAD", "btc": f"BUY {q_btc} BTC", "eth": f"SHORT {q_eth:.4f}
ETH", "tp_spread": level + 0.045}) # TP ที่ level + 1σ elif prev_spread <
level and current_spread >= level: # Spread ขึ้นผ่าน level → spread กำลังสูง
→ SELL spread (Short BTC, Long ETH) orders.append({"action":
"SELL_SPREAD", "btc": f"SHORT {q_btc} BTC", "eth": f"BUY {q_eth:.4f}
ETH", "tp_spread": level - 0.045}) # TP ที่ level − 1σ return orders
```

6.4 Capital Allocation สำหรับ Pair Grid

```
Capital Allocation — Pair Grid: สมมติ: BTC = $100k · ETH = $3,500 · beta = 0.82 Q_btc = 0.01 BTC per level Q_eth = Q_btc × BTC_price / ETH_price × beta = 0.01 × $100,000 / $3,500 × 0.82 = 0.01 × 28.57 × 0.82 = 0.2343 ETH # ตรวจ dollar parity BTC_notional = 0.01 × $100k = $1,000 ETH_notional = 0.2343 × $3,500 = $820 # beta-adjusted hedge ratio ทำให้ไม่ต้องเท่ากันพอดี (beta = 0.82 ≠ 1) Capital ต่อ level (worst case): Long BTC: $1,000 (ซื้อ BTC) Short ETH: margin $82 (10× leverage) Total per level: $1,082 # 7 levels total: Total pair grid capital: 7 × $1,082 ≈ $7,574 กำไรต่อรอบ (spread กลับ 1σ = 0.045): BTC leg: 0.01 BTC × $100k × 0.045 = $45 (ถ้า spread mean-reverts ผ่าน 1σ) ETH leg: 0.2343 ETH × $3,500 × 0.045 × 0.82 ≈ $30 Net gain: ≈ $45 − $30 = $15 per cycle (net of both legs) Note: กำไรจาก spread convergence ไม่ใช่ price direction
```

⚠️ **Beta (β) ระหว่าง BTC/ETH ไม่คงที่ตาม Regime**

Beta ที่คำนวณจาก 60–90 วัน อาจเปลี่ยนมากใน regime ต่างกัน:

- Bull market: β ≈ 1.1–1.3 (ETH outperforms BTC)
- Bear market: β ≈ 0.8–1.0 (ETH underperforms ชัดกว่า)
- ถ้าใช้ static β ที่คำนวณในช่วง bull แล้วเข้าสู่ bear → spread signal ผิดทั้งหมด

แก้: ใช้ rolling β (window 30 วัน) และ recalibrate ทุก 2 สัปดาห์ หรือตรวจ β drift > 20% → หยุด Pair Grid ชั่วคราว

```
def rolling_beta(btc_returns, eth_returns, window=30): """คำนวณ beta แบบ rolling""" betas = [] for i in range(window, len(btc_returns)): x = btc_returns[i-window:i] y = eth_returns[i-window:i] beta = np.cov(x, y)[0,1] / np.var(y) betas.append(beta) return betas # ถ้า |beta_current - beta_initial| > 0.2 → recalibrate หรือหยุด Pair Grid
```

ข้อดีของ Pair Grid vs Single-Asset Grid

- Market-neutral: BTC pump/dump ไม่ affect pair grid มากนัก (ถ้า cointegration คงอยู่)

- Spread mean-reversion แข็งแกร่งกว่า price mean-reversion (half-life สั้นกว่า)
- 2 sources of income: spread reversion + funding rate ของ short leg

6.5 Relative Value Switching — เลือก Asset ที่ดีที่สุด

เมื่อมีหลาย pair หรือหลาย asset — Relative Value Switching เลือก grid บน pair/asset ที่มี mean-reversion signal แรงที่สุด ณ เวลานั้น:

```
Relative Value Score — เปรียบ 3 grids: ตัวเลือก: A: BTC/USDT Spot Grid B: BTC/ETH Pair Grid C: ETH/USDT Spot Grid Score ต่อ asset (0–100): = 0.40 × Hurst_score + 0.30 × Spread_z_score + 0.30 × Momentum_score def relative_value_score(hurst, spread_zscore, momentum_reversal): """ spread_zscore = |current spread − mean| / std (ยิ่งสูง = ยิ่งน่า trade) momentum_reversal = RSI extreme (ดูจาก Part 3B) """ h_score = hurst_score_for_grid(hurst) # จาก Part 3B # Spread z-score: ยิ่งสูง ยิ่งดี (spread ห่างจาก mean มาก) z_score_clamped = min(3.0, abs(spread_zscore)) z_score_score = z_score_clamped / 3.0 * 100 # map to 0–100 mom_score = rsi_score_for_grid(momentum_reversal) # จาก Part 3B return 0.40 * h_score + 0.30 * z_score_score + 0.30 * mom_score # ตัวอย่าง: # A (BTC Spot): Hurst=0.48, z=0.5, RSI=45 → score=62 # B (BTC/ETH): Hurst=0.43, z=2.1, RSI=32 → score=79 ← WINNER # C (ETH Spot): Hurst=0.53, z=0.8, RSI=50 → score=45 # → Allocate 70% capital ไปที่ Pair Grid (B) + 30% ไปที่ BTC Spot (A)
```

6.6 Multi-Asset Grid Portfolio

เมื่อมี capital มากพอ — เปิดหลาย grid พร้อมกัน แต่ต้องจัดการ correlation ระหว่าง assets:

Grid	Asset	Correlation กับ BTC Grid	Max Allocation
Grid 1	BTC/USDT Spot	1.0 (ตัวเอง)	40% ของ grid portfolio
Grid 2	ETH/USDT Spot	0.75 (สูง)	20%
Grid 3	BTC/ETH Pair	0.10 (ต่ำ — market neutral)	30%
Grid 4	SOL/USDT	0.55 (ปานกลาง)	10%

```
Portfolio Correlation Management: def max_allocation_by_correlation(correlation_with_btc): """ ยิ่ง correlation สูง → max allocation น้อยลง (ป้องกัน concentration) """ if correlation_with_btc > 0.80: return 0.20 # max 20% elif correlation_with_btc > 0.60: return 0.25 # max 25% elif correlation_with_btc > 0.40: return 0.35 # max 35% else: return 0.50 # market-neutral: max 50% ข้อดี Multi-Asset Grid: + Diversification ลด drawdown รวม + เมื่อ BTC trending (grid BTC หยุด) → Pair Grid ยังทำงาน + ลด "idle capital" ในช่วง single-asset dead zone ข้อเสีย: − ต้องการ capital มากกว่า − การ monitor ซับซ้อนขึ้น − Correlation พัง ใน market crash (all correlate to 1)
```

6.7 Running Example — BTC/ETH Pair Grid

🔁 **Running Example: BTC/ETH Pair Grid Implementation**

Setup: Capital $15,000 · BTC=$102k · ETH=$3,600 · beta=0.80 · spread_std=0.040

```
Step 1: คำนวณ Q Q_btc = 0.008 BTC per level (จาก capital / N × price) Q_eth = 0.008 × $102k / $3,600 × 0.80 = 0.1813 ETH Step 2: Spread levels (7 levels ±3σ) σ = 0.040 → levels: −0.12, −0.08, −0.04, 0, +0.04, +0.08, +0.12 Step 3: Current spread log(102,000) − 0.80 × log(3,600) = 11.533 − 0.80×8.188 = 11.533 − 6.550 = 4.983 spread_mean (OLS fit) ≈ 4.983 → normalized spread = (4.983−4.983)/0.040 = 0 Day 5: BTC ลง $4k → $98k, ETH ขึ้น $200 → $3,800 new spread = log(98k) − 0.80×log(3,800) = 11.493 − 0.80×8.243 = 11.493−6.594 = 4.899 normalized: (4.899 − 4.983)/0.040 = −2.1σ → ใกล้ −2σ level Action: BUY spread ที่ −2σ BUY 0.008 BTC @ $98,000 = $784 SHORT 0.1813 ETH @ $3,800 = $689 notional (margin 10× = $69) TP: spread กลับมา −1σ level Day 8: BTC $103k, ETH $3,700 → spread กลับมา −0.5σ ไม่ถึง TP (−1σ) ยังถือ position Day 10: BTC $106k, ETH $3,650 → spread = +0.3σ (กลับมา near mean) TP hit! BTC gain: 0.008 × ($106k − $98k) = $64 ETH short gain: 0.1813 × ($3,800 − $3,650) = $27.2 Net P&L: $64 + $27.2 = $91.2 per cycle (ใน 5 วัน) ≈ $91 / $853 capital used = 10.7% ใน 5 วัน ← best-case (spread revert ตรงเวลา, ไม่มี slippage) Note: BTC ขึ้น $8k แต่ ETH ขึ้นน้อยกว่า (relative) → spread converge → กำไรจาก spread ไม่ใช่ direction ในทางปฏิบัติ spread ไม่ revert ทุก cycle — ผลจริงต่ำกว่านี้ได้
```

6.8 แบบฝึกหัด

แบบฝึกหัด Part 6

▶ ข้อ 1: ADF stat = −2.45 บน BTC/ETH spread · Critical value 5% = −2.89 — Cointegrated หรือไม่?

▶ ข้อ 2: Pair Grid BTC/ETH มี correlation กับ BTC Spot Grid ต่ำ (~0.10) — ทำไม Pair Grid ถึง market-neutral?

▶ ข้อ 3: beta=0.80 · Q_btc=0.01 BTC · BTC=$100k · ETH=$4,000 — Q_eth เท่าไหร่? Dollar value match ไหม?

▶ ข้อ 4: ทำไม correlation ระหว่าง assets พัง (→ 1) ใน market crash? และ Pair Grid จะเป็นอย่างไร?

GRID TRADING MASTERY · PART 6B

Grid Snowball + DCA Stack

3 รูปแบบ Compound · Profit-Triggered Layer · Zone Upgrade

Grid → DCA Stack: Trading Portfolio ป้อน Wealth Portfolio

Long-Term BTC Accumulation ผ่าน Grid Cashflow

แก่นของ PART นี้

Grid ที่ดีทำกำไรทุกวัน แต่ถ้าแค่ "นำกำไรออก" — compound growth ไม่เกิด Grid Snowball นำ cashflow กลับเข้า grid เพื่อเพิ่ม Q หรือเปิด zone ใหม่ — นอกจากนี้ Grid-DCA Stack ยังส่งส่วนหนึ่งของกำไรไปซื้อ BTC ระยะยาว (Wealth Portfolio) อย่างเป็นระบบ

```
Snowball: Q(t) = Q₀ × (1 + daily_return)^t · DCA Stack:
30% grid profit → weekly BTC buy
```

6B.1 Grid Snowball — 3 รูปแบบ

Type 1: Simple Compound

นำ profit ทั้งหมดกลับเข้า grid

เพิ่ม Q ทุก N cycles

Type 2: Profit-Triggered Layer

เปิด grid level ใหม่เมื่อ profit

สะสมถึง threshold

Type 3: Zone Upgrade

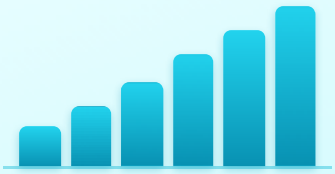
ขยาย zone หรือเพิ่ม N

เมื่อ capital เพิ่มพอ

GRID SNOWBALL 3 แบบ + DCA STACK

Type 1: Simple Compound

Q เพิ่มทุก 7 วัน



compound ราบเรียบ

Type 2: Profit-Triggered

Layer ใหม่เปิดเมื่อกำไรถึง threshold

Layer 3

▲ profit >= $500

Layer 2

▲ profit >= $500

Layer 1 (ต้นทุน $10k)

compound แบบ discrete

Type 3: Zone Upgrade

กำไรขยาย zone coverage

$90k – $110k

(zone เดิม)

compound แบบ expanded coverage

Grid → DCA Stack

Trading Portfolio

Grid ~$45/day

(active trading)

→ weekly

70% Compound

30% Transfer

(split allocation)

→ monthly

Wealth Portfolio

BTC accumulation

(passive long-term)

reinvest 70% → ขยาย Grid size · โอน 30% → สะสม BTC ทุกเดือน

compound สองมิติ: active (grid) + passive (BTC hodl)

กำไรจาก Grid (active) → สะสม BTC ระยะยาว (passive) = compound สองมิติ

6B.2 Type 1: Simple Compound

นำ realized profit กลับเข้า grid capital → Q เพิ่มขึ้น → cashflow ต่อรอบเพิ่มขึ้น → compound effect:

$$Q(t) = Q_0 \times (1 + r_{\text{daily}})^t$$

```
def simple_compound_grid(initial_q, daily_profit_rate, days,
rebalance_every=7): """ Type 1: Compound Q ทุก N วัน daily_profit_rate =
กำไรต่อวัน / grid capital rebalance_every = ปรับ Q ทุกกี่วัน """ q = initial_q
capital = initial_q * 100000 # สมมติ price $100k log = [] for day in
range(1, days + 1): # กำไรวันนี้ daily_profit = capital *
daily_profit_rate capital += daily_profit # ปรับ Q ทุก rebalance_every วัน
if day % rebalance_every == 0: new_q = capital / 100000 q =
round(new_q, 4) # round เพื่อให้ practical log.append({"day": day, "q": q,
"capital": capital}) return log # ตัวอย่าง: Q₀=0.01 BTC · rate=0.27%/วัน ·
365 วัน # Q(365) = 0.01 × (1.0027)^365 = 0.01 × 2.66 = 0.0266 BTC #
Capital: $1,000 → $2,660 (+166% ใน 1 ปี) ข้อดี: เรียบง่าย ไม่ต้องตัดสินใจ ข้อเสีย:
ถ้ากำไรไม่ realized (inventory ค้าง) → Q ไม่เพิ่ม
```

6B.3 Type 2: Profit-Triggered Layer

รอให้ profit สะสมถึง threshold แล้วค่อยเปิด "grid level ใหม่" — ได้ compound แบบ discontinuous step:

```
def profit_triggered_layer(grid_bot, trigger_threshold=500): """ เปิด grid level ใหม่เมื่อ accumulated profit ≥ threshold """ accumulated_profit = 0 layers_added = 0 while True: # รับ daily profit จาก grid daily_profit = grid_bot.get_daily_realized_profit() accumulated_profit += daily_profit if accumulated_profit >= trigger_threshold: # เปิด level ใหม่ (extend zone ลงอีก 1 step) new_level_price = grid_bot.zone_low - grid_bot.step grid_bot.add_level(price=new_level_price, qty=grid_bot.base_q) grid_bot.zone_low = new_level_price layers_added += 1 # Reset accumulation counter accumulated_profit -= trigger_threshold print(f"Layer {layers_added} added at ${new_level_price:,}") # ตัวอย่าง: # Grid capital $10,000, กำไร $100/วัน # Trigger every $500 profit ≈ ทุก 5 วัน # ใน 1 เดือน: เพิ่มประมาณ 6 layers ใหม่ เพิ่ม zone ลงต่ำกว่า หรือสูงกว่า: กลยุทธ์ A: เพิ่ม lower zone → capture ถ้าราคาลงมา กลยุทธ์ B: เพิ่ม upper zone → เตรียม TP ในชั้นสูงขึ้น กลยุทธ์ C: สลับตาม Hurst (ถ้า H ต่ำ → ต่ำสุด, ถ้า H สูงขึ้น → บนสุด)
```

6B.4 Type 3: Zone Upgrade

เมื่อ grid capital สะสมถึง threshold ใหม่ — ปิด grid เดิม เปิด grid ใหม่ที่ใหญ่กว่า (zone กว้างขึ้น หรือ Q ใหญ่ขึ้น):

```
Zone Upgrade Criteria: def check_zone_upgrade(current_capital, initial_capital, upgrade_threshold=0.50): """ Upgrade เมื่อ capital เพิ่มขึ้น 50% จากตอนเริ่ม """ growth = (current_capital - initial_capital) / initial_capital return growth >= upgrade_threshold Zone Upgrade Process: Initial: Capital $10,000 · Q=0.001 BTC · Zone $90k–$110k After 3 months: Capital $15,000 (+50%) Upgrade Options: A) เพิ่ม Q: ใช้ $15,000 กับ zone เดิม → Q = 0.0015 BTC (+50%) B) ขยาย Zone: ใช้ $15,000 กับ zone ใหม่ $80k–$120k (+33% wider) C) เพิ่ม N: zone เดิม แต่ step เล็กลง (fill บ่อยขึ้น) ตัวอย่าง: $10,000 · Zone $90k–$110k · Step $2k · N=10 · Q=0.001 BTC ↓ 3 months profit $5,000 $15,000 · Zone $87k–$113k · Step $2k · N=13 · Q=0.00115 BTC (ปรับตาม capital) Note: Zone Upgrade ต้องผ่าน validation checklist (Part 3) ก่อน
```

6B.5 Snowball Comparison

Type	กลไก	Complexity	Growth Rate	Risk
Type 1 (Simple Compound)	Q เพิ่มทุก N วัน	ต่ำ	Smooth exponential	ต่ำ
Type 2 (Triggered Layer)	เปิด level ใหม่เมื่อ profit ถึง threshold	กลาง	Step function	กลาง (zone ขยาย)
Type 3 (Zone Upgrade)	เปิด grid ใหม่ทั้งหมด เมื่อ capital ถึง tier	สูง	Accelerated jump	สูง (ต้อง rebalance ทั้งหมด)

แนะนำ: ใช้ Type 1 ก่อน จน capital ≥ 2× initial → Type 3 Upgrade

ผู้เริ่มต้น: Type 1 — ง่าย ไม่ต้องตัดสินใจ ให้ compound ทำงานเอง

ผู้มีประสบการณ์: Type 2 — เพิ่ม level ตามโอกาส ใช้ Hurst เลือกทิศทาง

ผู้ advanced: Type 3 + ใช้ Donchian ใหม่ re-optimize zone ทุก upgrade cycle

6B.6 Grid-DCA Stack — สองพอร์ตโฟลิโอ

Grid Snowball ช่วยให้ Trading Portfolio โตขึ้น — แต่ DCA Stack เพิ่มมิติที่สอง: ส่งกำไรส่วนหนึ่งไป Wealth Portfolio (ซื้อ BTC ระยะยาว)

Trading Portfolio (Grid)

Grid ทำงานทุกวัน → Cashflow $X/วัน

70% reinvest back into grid (Snowball) | 30% → Transfer

Weekly Transfer Rule

ทุกวันจันทร์: นำ 30% ของ weekly profit ซื้อ BTC ราคาตลาด

ไม่ limit order, ไม่ wait for dip — ซื้อ market อาทิตย์ละครั้ง

Wealth Portfolio (Long-term BTC)

BTC สะสม ถือตลอด ไม่ขาย

เป้าหมาย: สะสม BTC ระยะ 5–10 ปี

```
DCA Stack Implementation: class GridDCAStack: def __init__(self, grid_bot, dca_pct=0.30, dca_interval_days=7): self.grid = grid_bot self.dca_pct = dca_pct # 30% ของ profit ไป DCA self.interval = dca_interval_days # DCA ทุกกี่วัน self.wealth_btc = 0.0 # BTC สะสมใน wealth portfolio self.trading_capital = grid_bot.capital def weekly_rebalance(self, current_btc_price): """ ทำทุกอาทิตย์: 1. วัด realized profit สัปดาห์นี้ 2. แบ่ง 30% ไปซื้อ BTC → wealth portfolio 3. ส่วนที่เหลือ compound กลับใน grid """ weekly_profit = self.grid.get_weekly_realized_profit() if weekly_profit <= 0: return # ไม่มีกำไร ไม่ DCA # DCA amount dca_amount = weekly_profit * self.dca_pct btc_bought = dca_amount / current_btc_price self.wealth_btc += btc_bought self.trading_capital += weekly_profit * (1 - self.dca_pct) print(f"DCA: ซื้อ {btc_bought:.6f} BTC @ ${current_btc_price:,}") print(f" Wealth BTC รวม: {self.wealth_btc:.4f} BTC") print(f" Trading capital: ${self.trading_capital:,.0f}") ตัวอย่าง 1 ปี: Grid profit: $100/วัน × 365 = $36,500 DCA (30%): $36,500 × 30% = $10,950 → ซื้อ BTC ทุกอาทิตย์ Compound (70%): $36,500 × 70% = $25,550 → กลับเข้า grid BTC ที่ซื้อ (avg
```

```
$100k/BTC): $10,950 / $100k = 0.1095 BTC + grid capital เพิ่มจาก $20,000
→ $45,550 (+127%)
```

6B.7 Long-term Projection — Grid Snowball + DCA Stack

⚠️ **คำเตือนสำคัญ — Projection นี้ไม่ใช่ผลลัพธ์ที่รับประกัน**

ตารางด้านล่างเป็น **กรณีที่ดีที่สุด (best-case scenario)** ภายใต้สมมติฐานที่ไม่สมจริง:

- Grid ทำกำไร 0.27%/วัน *ต่อเนื่องตลอด 5 ปี* — ในความเป็นจริงมีช่วง Grid OFF, DD Halt (Part 5), Sideways ยาวนาน — เมื่อ DD Halt ทำงาน compound หยุดชั่วคราว
- ไม่รวมช่วง Bear Market ที่ BTC ลง 50–80% (เกิดทุก 2–4 ปีในประวัติศาสตร์)
- Monte Carlo simulation (1,000 paths) ของ return จริงๆ แสดง range: **ปีที่ 5 = $50,000–$180,000** (median ≈ $95,000) ไม่ใช่ $528,000
- **Realistic expectation:** 40–70% ของตัวเลขในตาราง ในปีที่ market conditions เอื้ออำนวย

ใช้ตารางนี้เพื่อเข้าใจ *กลไก compound* เท่านั้น — ไม่ใช่เพื่อวางแผนทางการเงิน

ปีที่	Grid Capital	Daily Cashflow	BTC สะสม	Wealth Value (@ $100k/BTC)
เริ่มต้น	$20,000	$54/วัน (0.27%)	0 BTC	$0
ปีที่ 1	$38,500	$104/วัน	0.11 BTC	$11,000
ปีที่ 2	$74,100	$200/วัน	0.32 BTC	$32,000
ปีที่ 3	$142,700	$385/วัน	0.73 BTC	$73,000
ปีที่ 4*	$275,000	$743/วัน	1.50 BTC	$150,000
ปีที่ 5	$528,000	$1,426/วัน	2.80 BTC	$280,000

*หมายเหตุ: สมมติ BTC price คงที่ที่ $100k (ไม่รวม BTC appreciation), daily return คงที่ที่ 0.27%/วัน, DCA 30% ทุกสัปดาห์ *ปีที่ 4 ประมาณการ (interpolated) Projection ใช้เพื่อ illustrate compound effect เท่านั้น — ผลจริงขึ้นอยู่กับ market regime*

ความเสี่ยงของ Long-term Projection

- Grid ไม่ได้ทำกำไร 0.27%/วันตลอด — มีช่วง Grid OFF, DD Halt
- BTC price เปลี่ยน → grid capital มูลค่าเปลี่ยน + DCA buy price เปลี่ยน
- ค่าธรรมเนียมแพลตฟอร์มอาจเปลี่ยน
- Realistic target: 40–70% ของ projection (ให้ปรับ downside)

6B.8 Running Example — 6 เดือนแรก

🔁 **Running Example: Grid Snowball Type 1 + DCA Stack**

ตัวอย่างนี้ใช้ grid capital $10,000 (ต่างจากตาราง 6B.7 ที่ใช้ $20,000 เป็น base case) — เพื่อให้เห็นกลไก compound ชัดเจนในขนาดที่เล็กลง

Setup: Grid capital $10,000 · Q=0.001 BTC · Daily profit $27 (0.27%) · DCA 30% ทุกสัปดาห์

```
Month 1 (เดือนแรก): Weekly profit avg: $27 × 7 = $189 DCA each week: $189 × 30% = $56.7 → ซื้อ BTC @ $100k = 0.000567 BTC/week Compound: $189 × 70% = $132.3 → กลับเข้า grid End of Month: Grid capital = $10,000 + $132.3 × 4 = $10,529 Wealth BTC = 0.000567 × 4 = 0.00227 BTC Month 3: Grid capital ≈ $11,600 (compound + standby yield) New Q = $11,600 / ($100k × N) ≈ 0.00116 BTC (+16%) Daily cashflow ≈ $31.3 (+16% จากเดิม) Wealth BTC ≈ 0.007 BTC Month 6 (สิ้นสุด): Grid capital ≈ $13,900 Daily cashflow ≈ $37.5 Wealth BTC สะสม ≈ 0.016 BTC (≈ $1,600 @ $100k) สรุป 6 เดือน: Trading capital growth: $10,000 → $13,900 (+39%) Wealth BTC: 0.016 BTC ($1,600) Total value: $13,900 + $1,600 = $15,500 (+55% รวม) vs Buy-and-hold $10,000 BTC: ถ้า BTC flat = $10,000 (0%) ข้อสังเกต: ใน 6 เดือน BTC flat Grid: +55% (cashflow compound) B&H: +0% (รอ price) DCA: +0% (BTC ราคาเดิม)
```

6B.9 แบบฝึกหัด

แบบฝึกหัด Part 6B

▶ ข้อ 1: Grid profit $150/วัน · DCA 25% ทุกสัปดาห์ · BTC = $95,000 — DCA BTC ต่อเดือนเท่าไหร่?

▶ ข้อ 2: ทำไม DCA ทุกอาทิตย์แบบ market order ดีกว่ารอ dip?

▶ ข้อ 3: Grid capital เริ่ม $15,000 · Type 3 Upgrade เมื่อ capital ≥ 150% ของ initial — ต้องรอ profit เท่าไหร่? (daily return 0.27%)

▶ ข้อ 4: Type 2 vs Type 3 Snowball — เลือก Type ไหนถ้า BTC ลงมา 20% จากตอนตั้ง grid?

GRID TRADING MASTERY · PART 7

Advanced Integrations — IV, Funding Rate & Options

IV-Adaptive Grid · Funding Rate Grid · Options + Grid Combination

CSP ป้องกัน Inventory · Grid + Short Strangle

อ้างอิง: Vol Series (IV Rank, GARCH), Arb Series (Funding Rate), PM Series (CSP, Strangle)

แก่นของ PART นี้

Advanced Grid รวม 3 โลก: Grid mechanics (Part 1–6) + Volatility intelligence (Vol Series) + Derivatives income (Arb/PM Series) — เมื่อทำงานร่วมกัน ทำให้ capital efficiency สูงขึ้นโดยไม่เพิ่ม risk มากนัก

```
IV-Adaptive Step: Step = 0.5 × σ_implied × P · Funding
Grid: income = short_notional × rate × 3/day
```

7.1 IV-Adaptive Grid — ปรับ Step ตาม Implied Volatility

แทนที่จะใช้ ATR (backward-looking) — ใช้ Implied Volatility จาก Options market ซึ่งเป็น forward-looking:

$$\text{Step}_{\text{IV}} = 0.5 \times \frac{\text{IV}_{\text{annual}}}{\sqrt{365}} \times P \qquad (\text{1-day vol} \times \text{price})$$

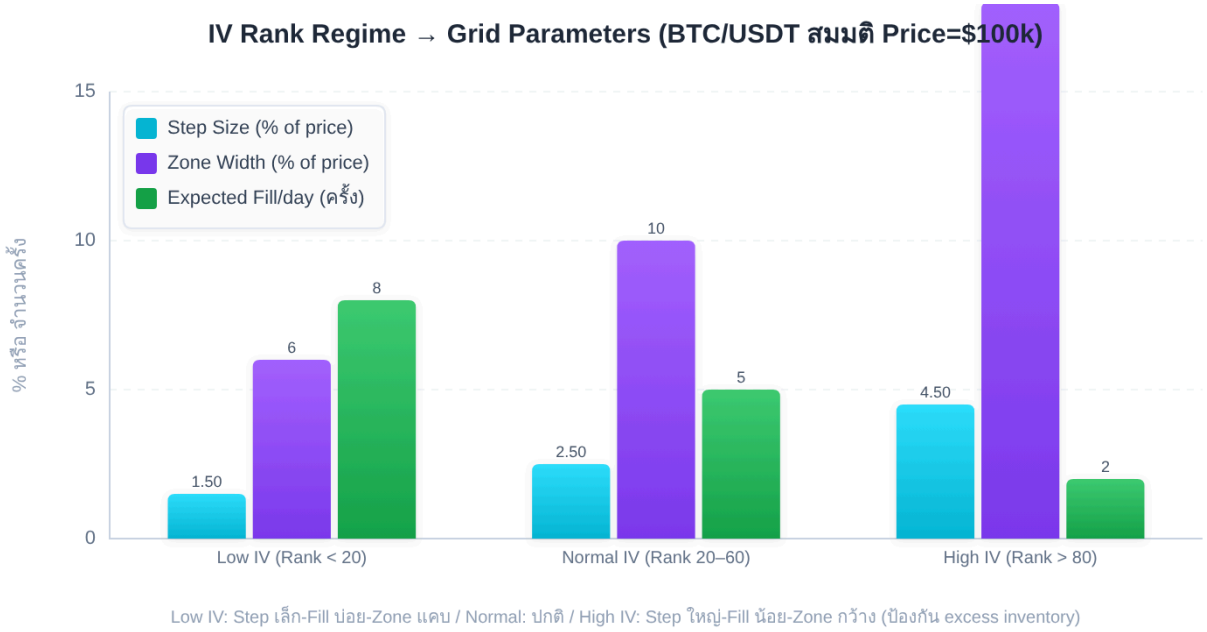
```
IV-Adaptive Step Implementation: def iv_adaptive_step(iv_annual_pct, current_price, fee_rate=0.001, step_factor=0.5): """ iv_annual_pct: Implied Volatility ต่อปี (เช่น 0.65 = 65%) """ # Convert annual IV to daily IV iv_daily = iv_annual_pct / (365 ** 0.5) # 1-day expected move (1σ) expected_daily_move = iv_daily * current_price # Step = fraction of expected daily move step = step_factor * expected_daily_move # Ensure profitability (≥ 3× fee) min_step = 3 * current_price * fee_rate * 2 step = max(step, min_step) return round(step / 100) * 100 # round to $100 # ตัวอย่าง: # BTC IV (7d ATM) = 65% annualized, P = $100,000 # iv_daily = 65% / sqrt(365) = 3.40%/day # expected_move = 3.40% × $100,000 = $3,400 # step = 0.5 × $3,400 = $1,700 → round → $1,700 # เทียบ ATR-based (ATR=2,500): # step_atr = 0.5 × $2,500 = $1,250 # → IV-based step ใหญ่กว่า เพราะ market expect volatility สูง
```

IV Rank — เลือก Zone Width ตาม IV Percentile

IV Rank = (IV_current – IV_min_52w) / (IV_max_52w – IV_min_52w) × 100

- IV Rank < 20 (low IV): zone แคบ, step เล็ก → grid fill บ่อย (mean-reverting regime)
- IV Rank 20–60 (normal): zone ปกติ, step ≈ 0.5 × IV_daily × P
- IV Rank 60–80 (elevated): zone กว้างขึ้น, step เพิ่ม 20–40% → ระวัง breakout
- IV Rank > 80 (high IV): zone กว้าง, step ใหญ่ → ป้องกัน excess fill ใน high vol

อ้างอิง: Vol Series Part 2 สำหรับ IV Rank calculation



IV Rank กำหนด Step และ Zone Width — IV ต่ำ: grid แน่น / IV สูง: grid กว้าง ป้องกัน over-fill

7.2 Funding Rate Adaptive Grid

Funding Rate ของ BTC Perpetual เปลี่ยนตาม market sentiment — grid สามารถ adapt ตาม Funding Rate ได้:

```
Funding Rate Grid Rules: Funding Rate > 0 (Long จ่าย Short รับ): → Short Layer (Part 1B) ได้ income เพิ่ม → เพิ่ม Q ของ short layer → เปิด Grid Hedge (Part 1C) ได้ดี Funding Rate < 0 (Short จ่าย Long รับ): → Short layer เสียต้นทุนเพิ่ม → ลด Q short layer หรือปิด short ชั่วคราว → ถ้า negative มาก → หยุด hedge def funding_adaptive_short_size(funding_rate_8h, base_q_short, threshold_high=0.001, threshold_low=-0.0005): """ funding_rate_8h: funding rate ต่อ 8 ชั่วโมง (เช่น 0.001 = 0.1%) """ if funding_rate_8h > threshold_high: # High positive funding → เพิ่ม short (เก็บ funding มากขึ้น) multiplier = min(2.0, 1 + (funding_rate_8h - threshold_high) / 0.001) elif funding_rate_8h < threshold_low: # Negative funding → ลด short (ลด cost) multiplier = max(0.0, 1 + (funding_rate_8h - threshold_low) / 0.001) else: # Normal range → ไม่เปลี่ยน multiplier = 1.0 return base_q_short * multiplier Funding Income Calculation (สำหรับ Hedge Grid): short_notional = inventory_btc × current_price daily_funding = short_notional × funding_rate_8h × 3 # 3 periods/วัน monthly_funding = daily_funding × 30 ตัวอย่าง: Inventory = 0.05 BTC, Price = $100k, Rate = 0.05%/8h Daily funding = (0.05 × $100k) × 0.05% × 3 = $5,000 × 0.15% = $7.50/วัน Monthly = $7.50 × 30 = $225/เดือน (ได้เพิ่มจาก hedge)
```

7.3 Cash-Secured Put (CSP) + Grid — Inventory Entry

แทนที่จะซื้อ BTC ตรงๆ ผ่าน BUY limit order — ใช้ CSP (Cash-Secured Put) เพื่อซื้อ BTC ในราคาที่ต้องการ พร้อมรับ premium:

```
CSP + Grid Concept: ปัญหาของ BUY limit order: วาง BUY $90,000 รอ → อาจรอนาน หรือไม่ fill เลย → capital idle CSP Solution: SELL PUT $90,000 expiry 2 สัปดาห์ → รับ premium $500 ทันที → ถ้า BTC < $90,000 → ถูก assign = ซื้อ BTC ที่ $90,000 (ตามแผน) → ถ้า BTC > $90,000 → PUT หมดอายุ = เก็บ premium $500 โดยไม่ต้องซื้อ Net effect: ซื้อ BTC ที่ effective price = $90,000 − $500 = $89,500 (ถูกกว่า limit order) หรือเก็บ premium $500 ถ้าไม่ถูก assign def csp_vs_limit_order(target_price, option_premium, expiry_price): """ เปรียบ CSP กับ limit order (expiry_price = ราคา ณ วันหมดอายุ ไม่ใช่ปัจจุบัน) """ # Limit Order: รอ fill ที่ target_price limit_effective_cost = target_price # CSP: ได้ premium แน่ๆ if expiry_price < target_price: # ถูก assign (ราคา ณ expiry < strike) csp_effective_cost = target_price - option_premium else: # ไม่ถูก assign (ราคา ณ expiry >= strike) csp_income = option_premium # รับฟรี return { "limit_cost": limit_effective_cost, "csp_cost_if_assigned": target_price - option_premium, "csp_income_if_not_assigned": option_premium } สำหรับ Grid: แทน BUY order แต่ละ level ด้วย CSP Level $90k → SELL PUT $90k strike Level $92k → SELL PUT $92k strike Level $94k → SELL PUT $94k strike (ขาย multiple CSP แทน multiple limit orders)
```

ข้อจำกัดของ CSP + Grid

- ต้องการ capital เต็มสำหรับ assignment (เช่น SELL PUT $90k ต้องมี $90,000 พร้อม)
- BTC Options บน Deribit/Bybit — liquidity ต่ำกว่า spot หรือ perp
- Expiry ต้องตรงกับ grid horizon (ไม่ควร expiry นานเกินกว่า 2× grid rebalance cycle)
- ใน downtrend แรง: PUT ถูก assign ทุก level → ซื้อ BTC ที่ราคาลงต่อ (เหมือน Close System)

7.4 Short Strangle + Grid — Income Enhancement

เมื่อ grid อยู่ในโซน sideway + IV สูง: SELL CALL ที่ zone_high และ SELL PUT ที่ zone_low → เก็บ premium จาก "range-bound expectation":

```
Short Strangle + Grid: Grid Zone: BTC $90k–$110k Short Strangle: SELL PUT $90,000 (strike = zone_low) SELL CALL $110,000 (strike = zone_high) Premium received: PUT $800 + CALL $600 = $1,400 total Scenarios: 1. BTC อยู่ $90k–$110k → Strangle ไม่ถูก assign → เก็บ $1,400 Grid cashflow ยังทำงาน → Double income 2. BTC < $90k → PUT assign → ซื้อ BTC $90k (= zone_low grid ทำงาน) Grid ก็ BUY ที่ $90k อยู่แล้ว → Strangle ช่วยลด cost 3. BTC > $110k → CALL assign → ขาย BTC $110k (= zone_high grid ก็ขาย) Grid SELL ที่ $110k อยู่แล้ว → Strangle ช่วยเพิ่ม income ข้อดี: ทั้ง 3 scenarios เป็นผลดี — strangle "align" กับ grid zone ข้อระวัง: ถ้า BTC พุ่งเกิน $110k ไกลมาก → CALL loss สูง (unlimited loss) → ต้อง hedge CALL → ซื้อ OTM CALL เป็น hedge (Iron Condor แทน Short Strangle) def grid_strangle_income(zone_low, zone_high, put_premium, call_premium, grid_daily_cashflow, days_to_expiry): """ ประมาณ total income ต่อ expiry cycle """ option_income = put_premium + call_premium grid_income = grid_daily_cashflow * days_to_expiry total = option_income + grid_income return { "option_premium": option_income, "grid_cashflow": grid_income, "total": total, "daily_equivalent": total / days_to_expiry }
```

7.5 GARCH-Based Volatility Forecasting สำหรับ Step

GARCH (Generalized Autoregressive Conditional Heteroskedasticity) คือโมเดลทำนาย volatility ที่คำนึงถึง volatility clustering — ช่วง vol สูงมักตามด้วย vol สูงต่อเนื่อง (อ้างอิงเต็มใน Vol Series Part 2) GARCH(1,1) ให้ vol forecast ที่ดีกว่า ATR ธรรมดา ในช่วง volatility cluster:

```
GARCH Grid Integration: from vol_series_tools import fit_garch, forecast_vol # Fit GARCH(1,1) บน 90 วันย้อนหลัง garch_params = fit_garch(daily_returns[-90:]) # Forecast vol สำหรับ 7 วันข้างหน้า vol_forecast_7d = forecast_vol(garch_params, horizon=7) # ใช้ 1-day vol forecast เป็น ATR sigma_1d = vol_forecast_7d[0] # day 1 forecast atr_garch = sigma_1d * current_price # Step based on GARCH step_garch = 0.5 * atr_garch # เปรียบ ATR vs GARCH: # ATR(14) = เฉลี่ย 14 วันที่แล้ว → lag # GARCH = มองไปข้างหน้า + ตอบสนองต่อ vol cluster ปัจจุบัน GARCH เปลี่ยน Step อย่างไร: วันปกติ (σ=2.5%): step_garch = 0.5 × 2.5% × $100k = $1,250 หลัง flash crash (σ=6%): step_garch = 0.5 × 6% × $100k = $3,000 สัปดาห์ต่อมา (σ ลดเหลือ 3.5%): step_garch = $1,750 → GARCH ปรับ step เร็วกว่า ATR(14) ที่ยังติด MA ของ 14 วันก่อน
```

7.6 Running Example — IV + Funding + Grid

🔁 **Running Example: Advanced Grid ที่ใช้ IV + Funding Rate**

Setup: BTC=$100k · IV(7d)=70% · Funding rate=0.08%/8h · Grid capital=$20,000

```
Step 1: IV-Adaptive Step iv_daily = 70% / sqrt(365) = 3.66%/day
step = 0.5 × 3.66% × $100k = $1,830 → $1,800 (round) ✓ ใหญ่กว่า
ATR-based ($1,250) เพราะ market expect vol สูง Step 2: Funding Rate
Assessment Rate = 0.08%/8h = 0.24%/วัน threshold_high = 0.1% → Rate
(0.08%) < threshold → multiplier = 1.0 (ไม่ปรับ Q) แต่ positive
funding → เปิด hedge short เพื่อเก็บ funding income (ดู Part 1C) Step 3:
Grid Hedge (Part 1C) Inventory BTC (5 levels): 0.05 BTC worst case
Perp short: 0.05 BTC × $100k = $5,000 notional (10× → $500 margin)
Daily funding: $5,000 × 0.08% × 3 = $12/วัน Step 4: Short Strangle
SELL PUT $88,000 (zone_low − 2σ buffer) → premium $400 SELL CALL
$113,000 (zone_high + 1σ buffer) → premium $350 Total premium =
$750 / 14 วัน = $53.6/วัน Step 5: Total Daily Income Grid cashflow:
$54/วัน Funding income: $12/วัน Strangle income: $53.6/วัน Total:
$119.6/วัน = 0.60%/วัน บน $20,000 ✓ เทียบ grid alone: $54/วัน =
0.27%/วัน (+121% จาก integration)
```

7.7 แบบฝึกหัด

แบบฝึกหัด Part 7

▶ ข้อ 1: BTC IV(7d)=80%, P=$95,000 — IV-adaptive Step คือเท่าไหร่?

▶ ข้อ 2: Funding rate = −0.03%/8h · Short notional $8,000 · Grid cashflow $40/วัน — Net P&L/วัน?

▶ ข้อ 3: Short Strangle: SELL PUT $88k + SELL CALL $115k · Premium $1,200 · Grid คาดว่า BTC อยู่ $90k–$112k · Expiry 2 สัปดาห์ — ความเสี่ยงหลักคืออะไร?

▶ ข้อ 4: เมื่อไหร่ควรใช้ CSP แทน BUY limit order ใน grid?

GRID TRADING MASTERY · PART 7B

Grid-Framework + Indicator Execution Styles

Grid เป็น Framework (Zone/Step/Capital/Risk) · Indicator เป็น Execution Engine

4 Execution Styles: Mean-Reversion · Trend · Volume · Composite

การเลือก Style ตาม Market Condition

แก่นของ PART นี้

"Grid" ไม่ใช่ strategy เดียว — มันคือ **framework** ที่กำหนด zone, step, capital และ risk — Indicator เป็น "execution engine" ที่ตัดสินใจว่าจะใช้ grid framework นั้น *อย่างไร* ในแต่ละช่วงเวลา ทั้ง 4 styles ใช้ grid math เหมือนกัน แต่ execution logic ต่างกัน

```
Grid Math (zone/step/Q/risk) = คงที่ · Execution Style =
เปลี่ยนตาม regime
```

7B.1 Grid เป็น Framework — ไม่ใช่ Strategy

ความเข้าใจผิดทั่วไป: "Grid trading" = strategy หนึ่ง ที่มี rule ชัดเจนว่า "ซื้อทุก $2k ลง ขายทุก $2k ขึ้น"

ความจริง: Grid คือ **framework สำหรับ organize capital และ risk** — ส่วน execution (ว่าจะ deploy เมื่อไหร่, ขนาดไหน, เพิ่ม/ลดอย่างไร) ขึ้นอยู่กับ indicator execution style

GRID FRAMEWORK

(structure — คำนวณครั้งเดียว ไม่เปลี่ยน)

[Z] Zone [Low, High]
Donchian(20d) + Hurst filter

[S] Step Size
ATR × k (adaptive)

[K] Capital per Level
Kelly f* × capital / N

[R] Risk Budget
Max DD = 8% total capital

[N] N Orders
= (Zone High – Zone Low) / Step

Grid math
defines WHAT

⇄

Indicator
decides WHEN

INDICATOR EXECUTION

(timing — เปลี่ยนตาม regime)

Style 1: Mean-Reversion
RSI extreme + BB near edge → deploy

Style 2: Trend-Adaptive
H rising → step ใหญ่ขึ้น, Q ลด

Style 3: Volume-Adaptive
Volume สูง → Q เพิ่ม 30%

Style 4: Composite Score
Score 0–100 → deploy % และ size mult

ผลลัพธ์ที่ได้

Framework-only: Sharpe 0.65 · DD –4.2% + ADX filter: Sharpe 0.72 · DD –2.8% improved

P&L รวมใกล้เคียงกัน แต่ risk-adjusted return ดีขึ้น — Framework คือ alpha หลัก, Indicator ปรับ Sharpe

ซ้าย: Framework กำหนดโครงสร้าง (zone/step/capital/risk) · ขวา: 4 Execution Styles ใช้ framework เดียวกัน ได้ผลต่างกัน

```
Grid Framework Components (คงที่ ไม่เปลี่ยน): 1. Zone: [zone_low, zone_high]
— จาก Donchian/Bootstrap (Part 3) 2. Step: grid spacing — จาก ATR/IV
(Part 3, 7) 3. Capital: grid_capital, Q per level — จาก Kelly (Part 5)
4. Risk: DD halt, position limits — จาก Part 5 Indicator Execution
Engine (เปลี่ยนตาม style/regime): - เมื่อไหร่ deploy capital (timing) - ขนาด
```

```
เท่าไหร่ต่อ level (size) - เพิ่ม Q ลด Q ตาม signal (adaptive) - Stop/resume
ตาม regime (state machine)
```

7B.2 Style 1 — Mean-Reversion Execution

Style 1: Mean-Reversion — เหมาะกับ H < 0.50, RSI extreme, BB near edge

Deploy ทุก level เท่ากัน, เน้น cashflow ทุกรอบ

```
class MeanReversionExecution: """ Style 1: Deploy grid ทั้งหมดเมื่อ mean-reversion signal ชัด เน้น: ทุก level fill บ่อย, cashflow สม่ำเสมอ """ def __init__(self, grid_framework): self.grid = grid_framework def should_deploy(self, indicators): score, _ = composite_grid_score(**indicators) return score > 60 # deploy เมื่อ score > 60 def get_level_sizes(self, n_levels): # Equal weight ทุก level return [self.grid.base_q] * n_levels def should_adjust_size(self, indicators): # ไม่ปรับ size — คงที่ตาม Kelly return 1.0 def get_execution_params(self, indicators): deploy = self.should_deploy(indicators) sizes = self.get_level_sizes(self.grid.n_levels) return {"deploy": deploy, "sizes": sizes, "style": "mean_reversion"} เหมาะกับ: Beginner → Advanced ข้อดี: ง่าย, predictable, cashflow สม่ำเสมอ ข้อเสีย: ไม่ adapt ตาม volatility regime
```

7B.3 Style 2 — Trend-Adaptive Execution

Style 2: Trend-Adaptive — ปรับ Step และ Q ตาม Hurst + ADX

ใช้ Step Pyramid (Part 2) อัตโนมัติ เพิ่ม step เมื่อ H สูง

```
class TrendAdaptiveExecution: """ Style 2: ปรับ step + size ตาม trend strength H ต่ำ = step เล็ก Q เต็ม | H สูง = step ใหญ่ Q ลด """ def get_adaptive_step(self, base_step, hurst, adx): # Step Pyramid (Part 2) hurst_mult = 1 + max(0, (hurst - 0.45) / 0.15) * 0.5 adx_mult = 1 + max(0, (adx - 20) / 30) * 0.3 return base_step * hurst_mult * adx_mult def get_adaptive_size_mult(self, hurst, adx): # ลด Q เมื่อ trending มากขึ้น if hurst > 0.60 or adx > 35: return 0.4 elif hurst > 0.55 or adx > 28: return 0.7 return 1.0 def get_execution_params(self, indicators): hurst = indicators["hurst30"] adx = indicators["adx14"] step = self.get_adaptive_step(self.grid.base_step, hurst, adx) size_mult = self.get_adaptive_size_mult(hurst, adx) sizes = [self.grid.base_q * size_mult] * self.grid.n_levels return { "deploy": True, # ไม่หยุด เพียงแต่ปรับ "step": step, "sizes": sizes, "style": "trend_adaptive" } เหมาะกับ: ผู้เข้าใจ Hurst + Step Pyramid ข้อดี: ไม่ต้อง pause grid เมื่อ trend เริ่ม → ลดทอน trend ผ่าน step ใหญ่ ข้อเสีย: ถ้า trend แรงมาก (H>0.65) ก็ยังขาดทุน unrealized
```

7B.4 Style 3 — Volume-Adaptive Execution

Style 3: Volume-Adaptive — เพิ่ม Q เมื่อ volume สูง (liquidity ดี), ลด Q เมื่อ volume ต่ำ

เน้น execution quality ใน high-liquidity windows

```
class VolumeAdaptiveExecution: """ Style 3: ปรับ Q ตาม volume profile
Volume สูง = fill ง่าย, spread แคบ → เพิ่ม Q Volume ต่ำ = fill ยาก, spread
กว้าง → ลด Q """ def __init__(self, grid_framework,
volume_history=None): self.grid = grid_framework volume_history =
volume_history or [] self.avg_volume = sum(volume_history[-20:]) / 20
if volume_history else 0 self.volume_by_hour =
self._compute_hourly_profile(volume_history) if volume_history else {}
def _compute_hourly_profile(self, volume_history): # คำนวณ avg volume
ต่อชั่วโมง (0–23) hourly = {} for vol, timestamp in volume_history: hour =
timestamp.hour hourly.setdefault(hour, []).append(vol) return {h:
sum(v)/len(v) for h, v in hourly.items()} def get_size_multiplier(self,
current_volume, current_hour): # Volume ratio vs historical average
vol_ratio = current_volume / self.avg_volume # Hourly profile bonus
hourly_avg = self.volume_by_hour.get(current_hour, self.avg_volume)
hourly_ratio = hourly_avg / self.avg_volume if vol_ratio > 1.5 and
hourly_ratio > 1.2: return 1.3 # เพิ่ม 30% ในช่วง high-liquidity elif
vol_ratio < 0.5 or hourly_ratio < 0.6: return 0.6 # ลด 40% ในช่วง low-
liquidity return 1.0 # ช่วงเวลา High Volume สำหรับ BTC: # 13:00–15:00 UTC
(Europe afternoon + US morning overlap) # 20:00–22:00 UTC (US afternoon
peak) # หลีกเลี่ยง: 03:00–06:00 UTC (Asia night, US sleep)
```

7B.5 Style 4 — Composite Score Execution

Style 4: Composite Score — รวม RSI + BB + Volume + Hurst เป็น score 0–100

ใช้ score กำหนด deploy % และ size multiplier (จาก Part 3B)

```
class CompositeScoreExecution: """ Style 4: Composite Score จาก Part 3B กำหนดทุกอย่าง """ def get_execution_params(self, indicators): score, sub_scores = composite_grid_score(**indicators) # Deploy percentage if score >= 75: deploy_pct = 1.00 elif score >= 55: deploy_pct = 0.60 elif score >= 35: deploy_pct = 0.30 else: deploy_pct = 0.00 # Size multiplier (สูงกว่าก็ deploy per level มากขึ้น) size_mult = 0.5 + (score / 100) * 0.5 # 0.5× ถึง 1.0× # Level allocation (emphasis on most signal levels) rsi_score = sub_scores["rsi"] if rsi_score > 70: # RSI extreme → เน้น lower levels level_weights = "bottom_heavy" elif sub_scores["bb"] > 70: # Near BB → เน้น near-edge level_weights = "bell" else: level_weights = "equal" sizes = self.grid.allocate_by_weight(level_weights, size_mult) return { "deploy": deploy_pct > 0, "deploy_pct": deploy_pct, "sizes": sizes, "size_mult": size_mult, "score": score, "style": "composite" } เหมาะกับ: Quant / systematic trader ข้อดี: รวม signal ทุกตัว ไม่มี blind spot ข้อเสีย: ซับซ้อน ต้อง calibrate weights
```

7B.6 เลือก Execution Style ตาม Trader Profile

Style	Trader Profile	Time Commitment	Key Metric	Expected Return
Mean-Reversion	Beginner / Long-term	ต่ำ (weekly check)	Cashflow/day	0.20–0.35%/วัน
Trend-Adaptive	Intermediate	กลาง (daily check)	Cashflow + less DD	0.18–0.30%/วัน
Volume-Adaptive	Active Trader	สูง (hourly)	Execution quality	0.22–0.38%/วัน
Composite Score	Quant / Advanced	Automated (real-time)	Risk-adjusted return	0.25–0.45%/วัน

Style ไม่ใช่สิ่งที่เลือกครั้งเดียว

ใน practice: ใช้ Mean-Reversion เป็น baseline | เพิ่ม Trend-Adaptive เมื่อ Hurst เริ่มขึ้น | เพิ่ม Volume-Adaptive เมื่อเป็น professional automated grid

Composite Score เป็น superset — รวมทุก style ไว้ด้วยกัน แต่ต้องการ infrastructure ที่ดีกว่า

7B.7 Unified Grid Controller — รวมทุก Style

```
class UnifiedGridController: """ Grid controller ที่รองรับทุก execution
style ใช้ config เพื่อเลือก style """ STYLES = { "mean_reversion":
MeanReversionExecution, "trend_adaptive": TrendAdaptiveExecution,
"volume_adaptive": VolumeAdaptiveExecution, "composite":
CompositeScoreExecution, } def __init__(self, grid_framework,
style="mean_reversion"): self.grid = grid_framework self.execution =
self.STYLES[style](grid_framework) self.state_machine =
GridStateMachine() # Part 4 self.dd_monitor = DrawdownMonitor() # Part
5 def run_cycle(self, market_data): """ ทำงาน 1 รอบ (ทุก 1 ชั่วโมง): 1.
ตรวจ DD halt 2. ตรวจ regime state 3. รับ execution params จาก style 4.
Execute orders """ # Step 1: DD Check (Part 5) dd_status, dd_value, _ =
self.dd_monitor.check( market_data["portfolio_history"] ) if dd_status
== "HALT": return {"action": "HALT", "reason": f"DD={dd_value:.1%}"} #
Step 2: Regime Check (Part 4) state, signals =
self.state_machine.evaluate( market_data["hurst30"],
market_data["adx14"], market_data["bb_width"] ) if state == "GRID_OFF":
return {"action": "PAUSE", "state": state} # Step 3: Execution Style
exec_params = self.execution.get_execution_params(market_data) # Merge
regime multiplier size_mult = self.state_machine.get_size_multiplier()
exec_params["sizes"] = [q * size_mult for q in exec_params["sizes"]] #
Step 4: Place orders orders = self.grid.create_orders(exec_params)
return {"action": "EXECUTE", "orders": orders, "params": exec_params}
```

7B.8 แบบฝึกหัด

แบบฝึกหัด Part 7B

▶ ข้อ 1: เหตุใด Grid Framework (zone/step/risk) ถึงไม่ใช่ส่วนที่ "ปรับตาม market" แต่ Execution Style ควรเปลี่ยนได้?

▶ ข้อ 2: Composite Score = 48 · Style = Mean-Reversion — deploy กี่เปอร์เซ็นต์? Style = Composite — deploy กี่เปอร์เซ็นต์?

▶ ข้อ 3: Volume-Adaptive Style ทำไมถึงเหมาะกับ "Active Trader" มากกว่า Beginner?

▶ ข้อ 4: UnifiedGridController ตรวจ DD Halt ก่อน Regime State — ทำไมลำดับนี้สำคัญ?

GRID TRADING MASTERY · PART 8

5 Supplementary Strategies

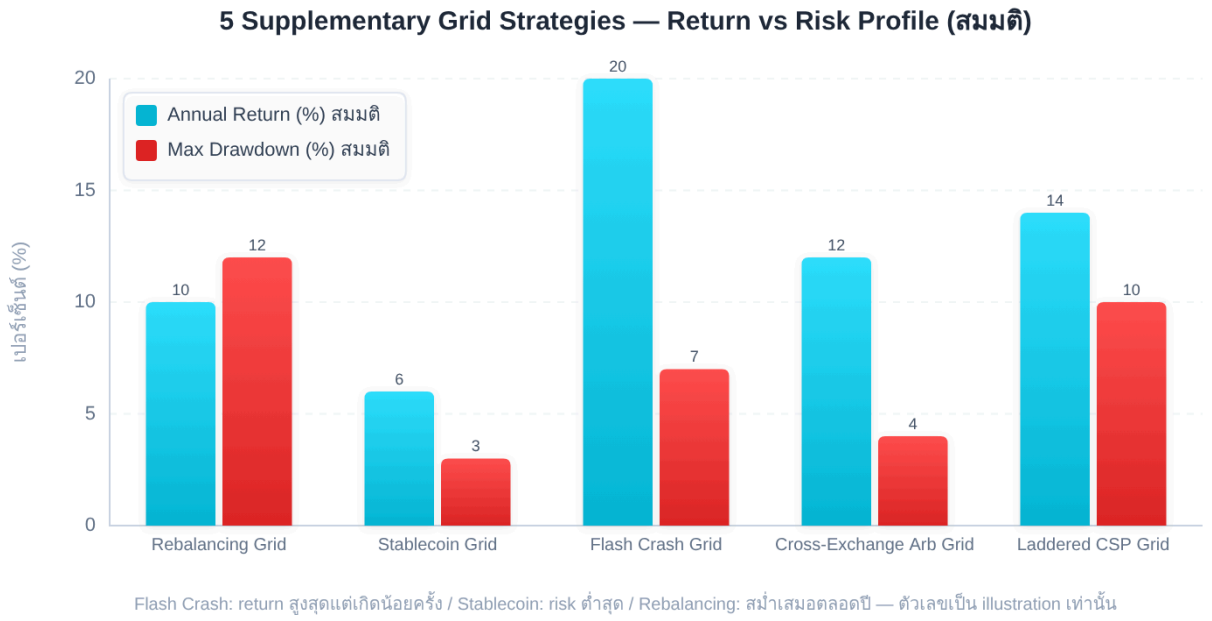
Rebalancing Grid · Stablecoin Grid · Flash Crash Grid

Cross-Exchange Arbitrage Grid · Laddered CSP Grid

แต่ละ strategy ใช้ core grid mechanics แต่ context ต่างกัน

แก่นของ PART นี้

5 strategies นี้ไม่ใช่ "แบบอื่น" ของ grid — ทั้งหมดใช้ core mechanics เดิม (zone, step, Q, Close System) แต่ apply ใน context พิเศษ เพื่อ capture โอกาสที่ standard grid ทำไม่ได้



เปรียบเทียบ 5 กลยุทธ์เสริม — แต่ละกลยุทธ์เหมาะกับ market regime ที่ต่างกัน

STRATEGY 1

Rebalancing Grid — ทำกำไรจาก Portfolio Drift

ใช้ grid mechanics เพื่อ rebalance portfolio โดยอัตโนมัติ — แทนที่จะ rebalance ทุกเดือนแบบ fixed, grid ทำ continuous rebalancing ตาม price movement:

```
Rebalancing Grid Setup: เป้าหมาย: รักษา BTC/USDT = 50%/50% ของ
portfolio Portfolio: $20,000 total ($10,000 BTC + $10,000 USDT)
BTC = $100k → จำนวน BTC = 0.10 เมื่อ BTC ขึ้น: BTC value เพิ่ม → % เกิน
50% → grid SELL BTC อัตโนมัติ (TP fills) ขาย BTC → ได้ USDT → ratio
กลับมา 50/50 เมื่อ BTC ลง: BTC value ลด → % ต่ำกว่า 50% → grid BUY BTC
อัตโนมัติ (BUY fills) ซื้อ BTC → ใช้ USDT → ratio กลับมา 50/50 กำไรจาก:
"Buy low, sell high" ทุกครั้งที่ rebalance → ได้ทั้ง rebalancing benefit
+ grid cashflow def rebalancing_grid(total_value, target_btc_pct,
btc_price, step_pct=0.02): """ Grid ที่ทำ portfolio rebalancing
step_pct: % deviation ก่อน trigger """ target_btc_value =
total_value * target_btc_pct target_btc_qty = target_btc_value /
btc_price # Grid levels รอบ target allocation levels = [] for i
in range(-5, 6): level_price = btc_price * (1 + i * step_pct) if
```

```
i < 0: # BTC ลง → BUY levels.append({"type": "BUY", "price":
level_price, "qty": target_btc_qty * abs(i) * 0.01}) elif i > 0:
# BTC ขึ้น → SELL levels.append({"type": "SELL", "price":
level_price, "qty": target_btc_qty * i * 0.01}) return levels
```

STRATEGY 2

Stablecoin Grid — ทำกำไรใน Bear Market

เมื่อ BTC ขาลงและ Hurst สูง (trending down) — grid BTC หยุด แต่ยังทำ Stablecoin Grid ได้ (ไม่ต้องถือ BTC):

```
Stablecoin Grid Options: 1. USDT/USDC Grid: ซื้อ USDT ถูก ขาย USDT
แพง (มี depeg event) Zone: $0.995–$1.005 · Step: $0.0005 กำไรต่อ
รอบ: Q × $0.0005 − fee (ต่ำมาก แต่ปลอดภัย) 2. Stablecoin Yield +
Grid: USDT ใน Standby Zone (Part 1A) → ฝากใน lending protocol
(Aave/Compound) ได้ yield 4–8% APY บน idle capital + เมื่อ BTC กลับมา
→ นำ USDT ออก deploy grid ใหม่ 3. USDT/BTC ที่ระดับ Support แรง: ถ้าเชื่อ
ว่า BTC มี hard support เช่น $70k → เปิด grid ที่ $65k–$80k เป็น "Bottom
Fishing Grid" ใน Bear Market def
stablecoin_grid_yield(standby_capital, lending_apy=0.05,
grid_days=90): """ คำนวณ yield จาก idle capital ใน bear market
""" daily_rate = (1 + lending_apy) ** (1/365) - 1 total_yield =
standby_capital * ((1 + daily_rate) ** grid_days - 1) return
total_yield # ตัวอย่าง: Standby $6,000 · APY 5% · 90 วัน # Yield =
$6,000 × ((1.000133)^90 − 1) = $6,000 × 1.2% = $73.5
```

STRATEGY 3

Flash Crash Grid — Capture Extreme Dips

Flash Crash Grid วางล่วงหน้าในช่วง "ต่ำผิดปกติ" ที่ตลาดน่าจะ recover ได้ภายใน 24–72 ชั่วโมง:

```
Flash Crash Grid: ลักษณะ Flash Crash ที่เหมาะ: - BTC ลง > 15% ภายใน
1 ชั่วโมง (liquidation cascade) - Volume spike > 5× average - ระดับ
macro เหมือนเดิม (on-chain ยังแข็งแกร่ง) - ไม่มี fundamental bad news
Setup: วาง BUY orders ทันทีหลัง crash (ไม่ใช่ก่อน crash) Zone: ราคา
ปัจจุบัน (หลัง crash) ±5% Step: ATR ปัจจุบัน (อาจสูงมาก) Close System: ใช้
capital ที่เตรียมไว้ในส่วน "Flash Crash Reserve" เท่านั้น class
```

```
FlashCrashGrid: def __init__(self, crash_capital_pct=0.10, recovery_hours=72): """ crash_capital_pct: % ของ total capital สำหรับ flash crash recovery_hours: เวลาที่คาดว่า recover ได้ """ self.capital_pct = crash_capital_pct self.recovery_hours = recovery_hours def is_flash_crash(self, drop_pct, volume_ratio, time_elapsed_hours): """ ตรวจว่านี่คือ flash crash หรือ trend reversal """ is_large_drop = drop_pct > 0.15 # ลง > 15% is_fast = time_elapsed_hours < 2 # เร็ว < 2 ชั่วโมง is_vol_spike = volume_ratio > 3.0 # volume 3× ปกติ return is_large_drop and is_fast and is_vol_spike Risk: Flash Crash กลายเป็น Trend Reversal → Close System รักษา inventory TP: ปิด grid เมื่อ recover 50% ของ crash depth หรือภายใน recovery_hours
```

STRATEGY 4

Cross-Exchange Arbitrage Grid

เมื่อ BTC price บน 2 exchanges ต่างกัน — ใช้ grid ทั้งสองฝั่งพร้อมกัน capture spread:

```
Cross-Exchange Grid: Exchange A (เช่น Binance): BTC = $100,050 Exchange B (เช่น Bybit): BTC = $99,950 Spread = $100 (ต่าง 0.1%) Grid Strategy: Exchange A: SELL BTC @ $100,050 (ขายแพง) Exchange B: BUY BTC @ $99,950 (ซื้อถูก) Net profit per cycle: $100 − fees def cross_exchange_spread_grid(price_a, price_b, q, fee_a, fee_b): """ คำนวณกำไรจาก cross-exchange spread """ spread = abs(price_a - price_b) fee_cost = q * (price_a * fee_a + price_b * fee_b) net_profit = q * spread - fee_cost return { "gross_spread": spread, "fee_cost": fee_cost, "net_profit": net_profit, "profitable": net_profit > 0 } ความท้าทาย: 1. Transfer BTC ระหว่าง exchange ใช้เวลา (10–30 นาที) → spread อาจหายไปแล้ว 2. ต้องมี capital บน ทั้งสอง exchange ตลอด 3. ค่า withdrawal fee + network fee กินกำไร 4. Slippage เมื่อ Q ใหญ่ วิธีแก้: ใช้ Perpetual บน Exchange A (short) + Spot บน Exchange B (long) → ไม่ต้อง transfer, ใช้ funding rate เป็น P&L indicator
```

STRATEGY 5

Laddered CSP Grid — ขาย Put เป็น Grid

แทนที่จะวาง BUY limit order ทั้งหมดพร้อมกัน — ขาย Put options เป็น "ladder" เพื่อรับ premium + ซื้อ BTC ในราคาที่ต้องการ (จาก Part 7):

```
Laddered CSP Grid: ตัวอย่าง: BTC = $100k · Grid capital $30,000 · 5 levels Level 1: SELL PUT $98k, expiry 2W, premium $600 Level 2: SELL PUT $95k, expiry 2W, premium $450 Level 3: SELL PUT $92k, expiry 4W, premium $700 Level 4: SELL PUT $88k, expiry 4W, premium $550 Level 5: SELL PUT $84k, expiry 8W, premium $800 Total premium = $3,100 (รับทันที) ถ้าไม่มี level ถูก assign ตลอด 8 สัปดาห์: Income = $3,100 premium = 10.3% บน $30,000 capital ใน 8 สัปดาห์ ถ้า BTC ลงมา $95k (ถึง Level 2): Level 1: expire worthless (+$600 เก็บ) Level 2: assign → ซื้อ BTC $95k (กำหนดไว้ล่วงหน้า) Level 3–5: ยังคง pending ข้อดีเทียบ Limit Order Grid: + รับ premium แม้ไม่มี fill + Effective buy price ต่ำกว่า (strike − premium) + Capital ไม่ lock ทั้งหมด (แค่ reserved สำหรับ worst-case) ข้อจำกัด: − ต้องการ Options liquidity (Deribit/OKX) − Expiry management ซับซ้อน − ถ้า BTC crash ลงเร็ว → ทุก level assign พร้อมกัน def laddered_csp_grid(grid_levels, btc_price, put_premiums): """ คำนวณ total income และ effective buy prices """ results = [] for price, premium in zip(grid_levels, put_premiums): effective_cost = price - premium results.append({ "strike": price, "premium": premium, "effective_cost": effective_cost, "discount_pct": premium / price * 100 }) return results
```

แบบฝึกหัด Part 8 — Quick Review

▶ ข้อ 1: ใน Bear Market ที่ BTC downtrend ชัด — Strategy ไหนในทั้ง 5 ข้อที่เหมาะสมที่สุด?

▶ ข้อ 2: Cross-Exchange Grid กำไรจากอะไร และทำไมถึงทำได้ยากในทางปฏิบัติ?

▶ ข้อ 3: Laddered CSP Grid ให้ effective buy price ต่ำกว่า Limit Order Grid อย่างไร?

▶ ข้อ 4: Flash Crash Grid ต่างจาก Standard Buy-Only Grid อย่างไร และทำไมต้องใช้ capital แยก (Flash Crash Reserve)?

GRID TRADING MASTERY · PART 9

Python Implementation Guide

Project Structure · Core Modules · Bybit API Integration
Backtesting Framework · Paper Trading → Live Migration
การ deploy Bot บน Server + Monitoring

แก่นของ PART นี้

Part นี้นำ pseudocode จาก Part 1–8 มาประกอบเป็น Python project ที่ runnable ได้จริง ทุกโค้ดที่เขียนในเล่มนี้ออกแบบมาให้ต่อกันได้ใน architecture ที่แสดงด้านล่าง

3.1 Project Structure

```
grid-trading-bot/ ├── core/ │ ├── __init__.py │ ├── grid_framework.py
# Zone/Step/Capital/Risk (Part 1A, 3, 5) │ ├── state_machine.py #
Regime Detection (Part 4) │ ├── execution_styles.py # 4 Execution
Styles (Part 7B) │ └── dd_monitor.py # Drawdown Monitor (Part 5) ├──
strategies/ │ ├── buy_only.py # Part 1A │ ├── bidirectional.py # Part
1B │ ├── hedge_grid.py # Part 1C │ ├── pair_grid.py # Part 6 │ └──
snowball.py # Part 6B ├── indicators/ │ ├── hurst.py # Hurst Exponent
(Part 4) │ ├── atr_donchian.py # ATR, Donchian, Keltner (Part 3) │ ├──
composite_score.py # RSI/BB/Vol/Hurst score (Part 3B) │ └── kelly.py #
Kelly Criterion (Part 5) ├── exchange/ │ ├── bybit_client.py # Bybit
API wrapper │ └── data_feed.py # Market data fetcher ├── backtest/ │
├── backtest_engine.py # Event-driven backtester │ ├── metrics.py #
Sharpe, Calmar, Max DD │ └── bootstrap_zone.py # Block Bootstrap (Part
3) ├── monitoring/ │ ├── watchdog.py # Grid Watchdog (Part 4) │ ├──
alerts.py # LINE/Telegram notify │ └── dashboard.py # Simple web
dashboard ├── config/ │ ├── config.yaml # All parameters │ └──
secrets.yaml # API keys (gitignored) ├── tests/ │ └── test_*.py # Unit
tests ├── main.py # Entry point └── requirements.txt
```

5.2 Core Grid Framework

```
# core/grid_framework.py from dataclasses import dataclass, field from typing import List, Optional import numpy as np @dataclass class GridLevel: price: float qty: float side: str # "BUY" or "SELL" tp_price: float sl_price: Optional[float] = None order_id: Optional[str] = None filled: bool = False @dataclass class GridFramework: """ Core grid parameters — computed once per deployment """ symbol: str zone_high: float zone_low: float step: float grid_capital: float base_q: float # Q per level (Kelly-based, Part 5) n_levels: int = field(init=False) avg_price: float = field(init=False) def __post_init__(self): self.n_levels = int((self.zone_high - self.zone_low) / self.step) self.avg_price = (self.zone_high + self.zone_low) / 2 @classmethod def from_market_data(cls, symbol, prices, highs, lows, grid_capital, kelly_fraction=0.19): """ สร้าง GridFramework จากข้อมูลตลาด (Part 3 + Part 5) """ from indicators.atr_donchian import compute_atr, compute_donchian current_price = prices[-1] # Zone from Donchian (Part 3) dc_high, dc_low = compute_donchian(highs, lows, period=30) zone_high = dc_high * 1.05 zone_low = dc_low * 0.95 # Step from ATR (Part 3) atr = compute_atr(highs, lows, prices, period=14) step = max(round(0.5 * atr / 100) * 100, 3 * current_price * 0.001 * 2) # Q from Kelly-sized capital (kelly_fraction already applied to net_worth before call) # grid_capital = net_worth × kelly_fraction → Q reflects Kelly sizing implicitly n = int((zone_high - zone_low) / step) avg_p = (zone_high + zone_low) / 2 base_q = grid_capital / (n * avg_p) return cls(symbol=symbol, zone_high=zone_high, zone_low=zone_low, step=step, grid_capital=grid_capital, base_q=round(base_q, 4)) def create_buy_levels(self, current_price, sizes=None) -> List[GridLevel]: """วาง BUY orders ต่ำกว่า current_price""" levels = [] price = current_price - self.step i = 0 while price >= self.zone_low: q = sizes[i] if sizes else self.base_q levels.append(GridLevel( price=round(price, 0), qty=q, side="BUY", tp_price=round(price + self.step, 0) )) price -= self.step i += 1 return levels def allocate_by_weight(self, style, size_mult=1.0) -> List[float]: """คำนวณ sizes ตาม allocation style (Part 2)""" import math n = self.n_levels if style == "equal": weights = [1.0] * n elif style == "bell": center = n // 2 weights = [math.exp(-0.5 * ((i-center)/(n/4))**2) for i in range(n)] elif style == "bottom_heavy": weights = [1.0/(i+1) for i in range(n)] else: weights = [1.0] * n
```

```
total = sum(weights) return [self.base_q * (w/total) * n * size_mult
for w in weights]
```

3.3 Bybit API Integration

```
# exchange/bybit_client.py import hmac import hashlib import time, requests from typing import Optional class BybitClient: BASE_URL = "https://api.bybit.com" def __init__(self, api_key: str, api_secret: str, testnet: bool = True): self.api_key = api_key self.api_secret = api_secret if testnet: self.BASE_URL = "https://api-testnet.bybit.com" # ⚠️ Security: Never log api_secret or the resulting signature. # Load api_secret from env var only: os.environ["BYBIT_API_SECRET"] def _sign(self, params: dict, timestamp: int) -> str: query = "&".join(f"{k}={v}" for k, v in sorted(params.items())) msg = f"{timestamp}{self.api_key}5000{query}" return hmac.new(self.api_secret.encode(), msg.encode(), hashlib.sha256).hexdigest() def _request(self, method: str, endpoint: str, params: dict = None): ts = int(time.time() * 1000) params = params or {} sig = self._sign(params, ts) headers = { "X-BAPI-API-KEY": self.api_key, "X-BAPI-SIGN": sig, "X-BAPI-TIMESTAMP": str(ts), "X-BAPI-RECV-WINDOW": "5000", } url = self.BASE_URL + endpoint if method == "GET": resp = requests.get(url, params=params, headers=headers) else: resp = requests.post(url, json=params, headers=headers) return resp.json() def place_order(self, symbol: str, side: str, qty: float, price: float, order_type: str = "Limit") -> dict: params = { "category": "spot", "symbol": symbol, "side": side, "orderType": order_type, "qty": str(qty), "price": str(price), "timeInForce": "GTC", # Good Till Cancel } return self._request("POST", "/v5/order/create", params) def cancel_all_orders(self, symbol: str) -> dict: return self._request("POST", "/v5/order/cancel-all", {"category": "spot", "symbol": symbol}) def get_orderbook(self, symbol: str, limit: int = 5) -> dict: return self._request("GET", "/v5/market/orderbook", {"category": "spot", "symbol": symbol, "limit": limit}) def get_wallet_balance(self, coin: str = "USDT") -> float: resp = self._request("GET", "/v5/account/wallet-balance", {"accountType": "UNIFIED"}) for item in resp.get("result", {}).get("list", [{}])[0].get("coin", []): if item["coin"] == coin: return float(item["walletBalance"]) return 0.0
```

3.4 Backtest Engine

```
# backtest/backtest_engine.py import pandas as pd import numpy as np from core.grid_framework import GridFramework, GridLevel class GridBacktester: """ Event-driven backtester สำหรับ Buy-Only Spot Grid """ def __init__(self, grid: GridFramework, fee_rate: float = 0.001): self.grid = grid self.fee_rate = fee_rate def run(self, ohlcv_df: pd.DataFrame) -> dict: """ ohlcv_df: DataFrame ที่มีคอลัมน์ open, high, low, close, volume """ capital = self.grid.grid_capital usdt = capital # เริ่มด้วย USDT ทั้งหมด btc = 0.0 # ไม่มี BTC เริ่มต้น inventory = {} # {buy_price: qty} trades = [] portfolio_values = [] # สร้าง BUY levels current_price = ohlcv_df["close"].iloc[0] buy_levels = self.grid.create_buy_levels(current_price) for idx, row in ohlcv_df.iterrows(): low_price = row["low"] high_price = row["high"] close = row["close"] # หมายเหตุ: logic นี้เป็น optimistic assumption — ราคา candle แตะ level ไม่ได้หมายความว่า fill ทุกครั้ง # ใน production ให้ใช้ WebSocket order events แทน # Check BUY fills (price ลงถึง BUY level) for level in buy_levels: if not level.filled and low_price <= level.price: cost = level.qty * level.price * (1 + self.fee_rate) if usdt >= cost: usdt -= cost btc += level.qty inventory[level.price] = level.qty level.filled = True trades.append({"date": idx, "side": "BUY", "price": level.price, "qty": level.qty}) # Check SELL fills (TP hit) for buy_price, qty in list(inventory.items()): tp_price = buy_price + self.grid.step if high_price >= tp_price: proceeds = qty * tp_price * (1 - self.fee_rate) usdt += proceeds btc -= qty del inventory[buy_price] # Reset BUY level for level in buy_levels: if level.price == buy_price: level.filled = False trades.append({"date": idx, "side": "SELL", "price": tp_price, "qty": qty}) # Portfolio value portfolio_values.append(usdt + btc * close) # Metrics pv = pd.Series(portfolio_values, index=ohlcv_df.index) returns = pv.pct_change().dropna() max_dd = (pv / pv.cummax() - 1).min() sharpe = returns.mean() / returns.std() * np.sqrt(365) if returns.std() > 0 else 0 return { "final_portfolio": portfolio_values[-1], "total_return": (portfolio_values[-1] / capital - 1), "max_drawdown": max_dd, "sharpe_ratio": sharpe, "n_trades": len(trades), "trades": pd.DataFrame(trades), "portfolio_values": pv, }
```

9.5 Main Entry Point & Config

```
# config/config.yaml grid: symbol: "BTCUSDT" grid_capital: 20000
kelly_fraction: 0.19 # f_safe = 25% × f* ≈ 19% (ตาม Part 5 ตัวอย่าง
f*=0.755) donchian_period: 30 atr_period: 14 step_factor: 0.5 # Step =
step_factor × ATR risk: dd_halt_threshold: -0.08 # −8% drawdown → HALT
(Part 5) dd_resume_threshold: -0.04 # −4% → resume
max_notional_per_level: 0.05 # 5% of capital (Part 5) regime:
hurst_on: 0.50 # Grid ON ถ้า H < 0.50 hurst_caution: 0.58 # CAUTION ถ้า
H 0.50–0.58 adx_caution: 25 adx_off: 35 bb_width_caution: 0.04
bb_width_off: 0.08 execution: style: "composite" # mean_reversion /
trend_adaptive / volume_adaptive / composite recheck_hours: 1 # ตรวจ
regime ทุก 1 ชั่วโมง snowball: enabled: true type: 1 # 1=simple compound,
2=triggered layer, 3=zone upgrade dca_pct: 0.30 # 30% ของ profit ไป
DCA # === ไฟล์แยก: main.py === import yaml from core.grid_framework
import GridFramework from core.state_machine import GridStateMachine
from core.dd_monitor import DrawdownMonitor from core.execution_styles
import UnifiedGridController from exchange.bybit_client import
BybitClient from exchange.data_feed import DataFeed from
monitoring.watchdog import GridWatchdog import time, logging
logging.basicConfig(level=logging.INFO, format="%(asctime)s [%
(levelname)s] %(message)s") def main(): # Load config with
open("config/config.yaml") as f: cfg = yaml.safe_load(f) with
open("config/secrets.yaml") as f: secrets = yaml.safe_load(f) #
Alternative (recommended): load from env vars instead of secrets.yaml
# api_key = os.environ["BYBIT_API_KEY"] # api_secret =
os.environ["BYBIT_API_SECRET"] # Initialize client =
BybitClient(secrets["api_key"], secrets["api_secret"], testnet=True)
data = DataFeed(client, cfg["grid"]["symbol"]) # Get market data
prices, highs, lows, volumes = data.get_ohlcv(days=60) current_price =
prices[-1] # Build framework framework =
GridFramework.from_market_data( symbol=cfg["grid"]["symbol"],
prices=prices, highs=highs, lows=lows, grid_capital=cfg["grid"]
["grid_capital"], kelly_fraction=cfg["grid"].get("kelly_fraction",
0.19) ) logging.info(f"Grid: zone=[{framework.zone_low:.0f},
{framework.zone_high:.0f}] " f"step={framework.step:.0f} Q=
{framework.base_q:.4f} N={framework.n_levels}") # Controller
controller = UnifiedGridController(framework, style=cfg["execution"]
["style"]) watchdog = GridWatchdog(controller,
check_interval_hours=cfg["execution"]["recheck_hours"]) # Main loop
```

```
portfolio_history = [cfg["grid"]["grid_capital"]] while True:
market_data = data.get_indicators(prices, highs, lows, volumes)
market_data["portfolio_history"] = portfolio_history result =
watchdog.run_checks(market_data) logging.info(f"State:
{result['state']} | Actions: {result['actions']}") for action in
result["actions"]: if action["action"] == "CANCEL_ALL_ORDERS":
client.cancel_all_orders(cfg["grid"]["symbol"])
logging.warning(f"Cancelled all orders: {action['reason']}") elif
action["action"] == "PLACE_ORDERS": for order in action.get("orders",
[]): resp = client.place_order( symbol=cfg["grid"]["symbol"],
side=order["side"], qty=order["qty"], price=order["price"] )
logging.info(f"Order placed: {resp}") # Update portfolio value balance
= client.get_wallet_balance("USDT") portfolio_history.append(balance)
# ⚠️ Production Note: sleep(3600) จะพลาด order fills ระหว่างรอ # ใช้
WebSocket private channel (wss://stream.bybit.com/v5/private) แทน
polling # ดู Bybit API docs: subscribe to "order" channel เพื่อรับ fill
events real-time # ⚠️ Production: implement exponential back-off
reconnect — connection drop = missed fills # while True: try:
ws.run_forever() except: time.sleep(min(2**attempt, 60)); attempt += 1
time.sleep(cfg["execution"]["recheck_hours"] * 3600) if __name__ ==
"__main__": main()
```

3.10 Paper Trading — ก่อนเอาเงินจริงลง

Phase	ระยะเวลา	ทำอะไร	Success Criteria
Phase 0: Backtest	1 สัปดาห์	Run backtester บน 1–2 ปีย้อนหลัง	Sharpe > 1.5, Max DD < 15%
Phase 1: Paper Trade	2–4 สัปดาห์	Run bot บน Testnet (สภาพแวดล้อมทดสอบของ Bybit — API จริง แต่เงินจำลอง ตั้ง testnet=True)	ไม่มี bug, state machine ทำงาน
Phase 2: Small Live	1 เดือน	$1,000 จริง บน Bybit	P&L ≈ backtest, DD ≤ −8%
Phase 3: Scale Up	ต่อเนื่อง	เพิ่ม capital หลังผ่าน Phase 2	Compound + DCA Stack

3.7 Troubleshooting Checklist — 7 ข้อที่ต้องตรวจก่อน Live

ตรวจทุกข้อก่อน switch testnet=False

- API Rate Limit: Bybit อนุญาต 600 requests/นาที → grid 30 levels × 2 (place/cancel) = 60/cycle ✓
- Minimum Order Size: BTC บน Bybit ขั้นต่ำ 0.001 BTC → Q ต้องมากกว่านี้
- Price Precision: BTC round ถึง $0.01 → step ต้องเป็น multiple ของ $0.01
- Network Redundancy: รัน bot บน VPS (Virtual Private Server — เซิร์ฟเวอร์เช่าที่ online ตลอด 24/7) พร้อม reconnect logic ไม่ควรรัน bot บน local machine ที่อาจปิดหรือ internet หลุด
- Secret Management: API key ใน environment variable ไม่ใช่ hardcode

9.8 Monitoring Dashboard

```
# monitoring/dashboard.py (Flask simple dashboard) from flask import
Flask, jsonify, render_template_string import json app =
Flask(__name__) HTML = """
```

Grid Bot Dashboard

Loading...

""" @app.route("/") def dashboard(): return render_template_string(HTML) @app.route("/api/status") def status(): # อ่านจาก state file ที่ bot เขียนไว้ try: with open("state.json") as f: return jsonify(json.load(f)) except FileNotFoundError: return jsonify({"error": "Bot not running"}) # รัน: python -m monitoring.dashboard if __name__ == "__main__": app.run(host="0.0.0.0", port=8080)

สรุปเนื้อหาทั้งเล่ม — Cheat Sheet

GRID TRADING MASTERY — Quick Reference ZONE: Donchian(30) ± 5% buffer | min_width ≥ 6×ATR STEP: max(0.5×ATR(14), 3×fee_per_cycle) → round to $100 Q: grid_capital / (N × avg_price) → ตรวจ 5% max notional N: zone_width / step → ควรอยู่ระหว่าง 15–25 KELLY: f* = (p×b − q)/b | f_safe = 0.25 × f* grid_capital = net_worth × f_safe REGIME (Part 4, BTC-calibrated): H < 0.50 + ADX < 40 + BB_width < 4% → GRID ON (full deploy) H 0.50–0.58 + ADX 40–50 + ... → CAUTION (50% size) H > 0.58 + ADX > 50 + BB_width > 8% → GRID OFF (pause) RISK (Part 5): DD ≥ −8% → HALT (cancel all, wait for recovery) DD ≥ −4% + H < 0.55 + 24h cooloff → RESUME SNOWBALL (Part 6B): Type 1: Q(t) = Q_0 × (1 + r_daily)^t DCA Stack: 30% profit → weekly BTC buy COMPOSITE SCORE (Part 3B): 30%×RSI + 30%×BB + 20%×Vol +

```
20%×Hurst → 0-100 >75: Full Deploy | 55-75: 60% | 35-55: 30% |
<35: Wait
```

แบบฝึกหัด Part 9

▶ ข้อ 1: อ่าน config.yaml แล้วอธิบายว่า kelly_fraction=0.19 หมายความว่าอย่างไร และ grid_capital=$20,000 มาจากการคำนวณอย่างไร?

▶ ข้อ 2: ใน run_cycle() ถ้า DD = -9% เกิดอะไรขึ้น? bot ยังคงทำงานไหม?

▶ ข้อ 3: ทำไม testnet=True ต้องถูกเปลี่ยนเป็น False ก่อน Live? มีความเสี่ยงอะไรถ้าลืม?

▶ ข้อ 4: Backtester ใน Part 9 มี "optimistic assumption" อะไรที่ทำให้ผลดีกว่าความเป็นจริง?

GRID TRADING MASTERY · APPENDIX

สูตร · Glossary · Cross-Book Index

ตารางสูตรทั้งหมด · คำศัพท์ไทย-อังกฤษ · ดัชนีข้ามเล่ม

A — สูตรหลัก (One-Page Cheat Sheet)

Grid Mechanics

```
กำไรต่อรอบ = Q × (P_sell − P_buy) − Fees Capital ถึง Floor = Σ(i=1 to
N) Q_i × P_i Soft Martingale = Q_i = Q_0 × r^i , r ∈ [1.0, 2.0] รอบ
ต่อวัน (ประมาณ) = ATR_daily / Step Grid step (ATR) = k × ATR_t , k ∈
[0.2, 1.0] Bell curve Q_i ∝ exp(−(P_i − μ)² / 2σ²)
```

Kelly Sizing

```
Kelly f* = (p × b − q) / b Fractional Kelly = 0.25 × f*
(conservative) Capital deployed = f* × total_capital p = win
probability, b = win/loss ratio, q = 1 − p
```

Hurst Exponent

```
log(R/S) = H × log(N) + C H < 0.5: mean-reverting (Grid ดี) H = 0.5:
random walk H > 0.5: trending (Grid หยุด)
```

Risk Management

```
Drawdown DD_t = (P_peak − P_t) / P_peak Max Notional ≤ 5% capital
per grid level Stop condition DD < −max_dd_threshold (e.g. −5%)
```

B — Glossary (ไทย-อังกฤษ)

คำศัพท์ (ไทย)	English	ความหมาย
กริด	Grid	กรอบราคาที่จัด order ไว้เป็นระดับ
ขั้น/ก้าว	Step	ระยะห่างระหว่าง grid level
วงรอบ	Cycle	การ BUY 1 ครั้ง + SELL 1 ครั้ง ที่ระดับถัดไป
ระบบปิด	Close System	วาง capital ครบถึง floor ก่อนเปิด grid
โซนใช้งาน	Active Zone	ส่วนของ capital ที่ deploy เป็น order
โซนสำรอง	Standby Zone	ทุนที่รอ deploy + ฝากรับดอกเบี้ย
โซนพื้น	Floor Zone	ทุนสำรองสุดท้าย ไม่แตะยกเว้น worst case
บวมบน	Top-Heavy	inventory สะสมมากในโซนราคาสูง
บวมล่าง	Bottom-Heavy	inventory สะสมมากในโซนราคาต่ำ
บวมกลาง	Center-Heavy / Bell	inventory หนาแน่นรอบ mean
ลูกหิมะ	Snowball	การ compound กำไร grid เพิ่ม capital
ซอฟต์มาร์ติงเกล	Soft Martingale	เพิ่ม size ด้วย $r \in [1.0, 2.0]$ แทน 2× ตายตัว
เลขชี้กำลัง Hurst	Hurst Exponent	วัดความ mean-reverting ของ time series
ระบอบตลาด	Regime	สภาวะตลาด: mean-reverting, trending, volatile
อัตราส่วนเคลลี่	Kelly Fraction	สัดส่วน capital ที่ optimal ตาม win%/odds

C — Cross-Book Index

Grid Concept	เล่มที่เชื่อมต่อ	บท/ตอน	การเชื่อมต่อ
Mean-reversion	Statistical Arbitrage	Ch.1–3 (OU Process)	Grid สมมติ OU reversion
Hurst Exponent	Statistical Arbitrage	Ch.6	ตัดสิน grid on/off
Cointegration	Statistical Arbitrage	Ch.5	Pair grid ต้องการ cointegrated
Kelly Criterion	Statistical Arbitrage	Ch.12	Sizing per grid level
Regime Detection	Statistical Arbitrage	Ch.9	Switch grid on/off
GARCH Vol	Statistical Arbitrage	Ch.7	Predict step size
IV, VIX, ATR	Volatility Mastery	Part 1–3	IV-adaptive step size
Vol Skew	Volatility Mastery	Part 4	Asymmetric zone sizing
Funding Rate	Arbitrage	Part 2A	Spot-perp grid hedge
Cash-Carry Arb	Arbitrage	Part 2B	Grid Hedge structure
CSP Payoff	Payoff Mastery	Part 3A	Options-as-Grid Level
Strangle	Payoff Mastery	Part 3B	Short strangle ที่ขอบ zone
Normal Distribution	คณิตศาสตร์	Part 1 (Statistics)	Bell curve zone density
OLS Regression	คณิตศาสตร์	Part 5 (Regression)	Hedge ratio สำหรับ pair grid