November 14, 2026

CoNGA '26

Revisiting the LINPACK Benchmark with Posits

By Arman Singh, Gil Speyer, and John L. Gustafson

Need for Arithmetic

$$\begin{pmatrix} 0.2 & 0.3 & 0.5 \\ -0.1 & 0.7 & 0.4 \\ 0.6 & 0.1 & 0.3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

We know x = 1 by construction, but the solver must recover it numerically

Exact Inverses

$$\mathbf{H}_2 = \begin{bmatrix} 1 & \frac{1}{2} \\ \frac{1}{2} & \frac{1}{3} \end{bmatrix}.$$

$$\mathbf{H}_2^{-1} = \begin{bmatrix} 4 & -6 \\ -6 & 12 \end{bmatrix},$$

$$\begin{pmatrix} a & b \\ c & d \end{pmatrix}^{-1} = \frac{1}{ad - bc} \begin{pmatrix} d & -b \\ -c & a \end{pmatrix}$$

Exact Inverses

$$round(\mathbf{H}_2) = \begin{bmatrix} 1 & \frac{1}{2} \\ \frac{1}{2} & \frac{11184811}{33554432} \end{bmatrix}.$$

Rounding the matrix can destroy exact algebraic structure

$$(round(\mathbf{H}_2))^{-1} = \begin{bmatrix} \frac{11184811}{2796203} & -\frac{16777216}{2796203} \\ -\frac{16777216}{2796203} & \frac{33554432}{2796203} \end{bmatrix}.$$

Exact Inverses

$$round((round(\mathbf{H}_2))^{-1}) = \begin{bmatrix} \frac{16777215}{4194304} & -\frac{12582911}{2097152} \\ -\frac{12582911}{2097152} & \frac{12582911}{1048576} \end{bmatrix}$$

$$\mathbf{H}_2^{-1} = \begin{bmatrix} 4 & -6 \\ -6 & 12 \end{bmatrix},$$

A Brief History of LINPACK

- 1979: Introduced as a standardized Linear Algebra Package written in Fortran
- Originally 100 × 100 linear systems: Gaussian elimination w partial pivoting
- Primary goal: Minimize execution time and evaluate computational performance

A Brief History of LINPACK

- 1984: 300 × 300 (Moore's Law)
- 1986: 1000 × 1000, parallel computing and matrix-matrix operations
- LINPACK became a foundational benchmark for evaluating high-performance computing systems

Key Takeaway: As computing power increased, LINPACK evolved to test increasingly large linear systems while prioritizing computational speed

A Principle for Numerical Benchmarking

Whenever possible, the INPUTS should be exactly expressible, and the exact answer should also be exactly expressible

LINPACK benchmark designers took a step in this direction but failed

2 Major Issues to Address

1. We want an output of all 1s!

$$\begin{pmatrix} 0.2 & 0.3 & 0.5 \\ -0.1 & 0.7 & 0.4 \\ 0.6 & 0.1 & 0.3 \end{pmatrix} \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} = \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

- This goal often fails!

- Need Purification so the vector of all 1's is correct

- Posits with Quire makes this goal more achievable

2 Major Issues to Address

2. The Process should be Iterative, not Direct

$$r = b - A\tilde{x}$$
$$A\delta = r$$
$$\tilde{x} \leftarrow \tilde{x} + \delta$$

- Rounding produces residuals which is perfect for iterations

- Greatly improves convergence chances

- Can reuse the same factorization with ease

Experimental Setup

$A \rightarrow$ Purify $\rightarrow PLU$

$\rightarrow$ Solve $Ax = b$

$\rightarrow r = b - Ax$

$\rightarrow$ Correction

$\rightarrow$ Repeat

Purification

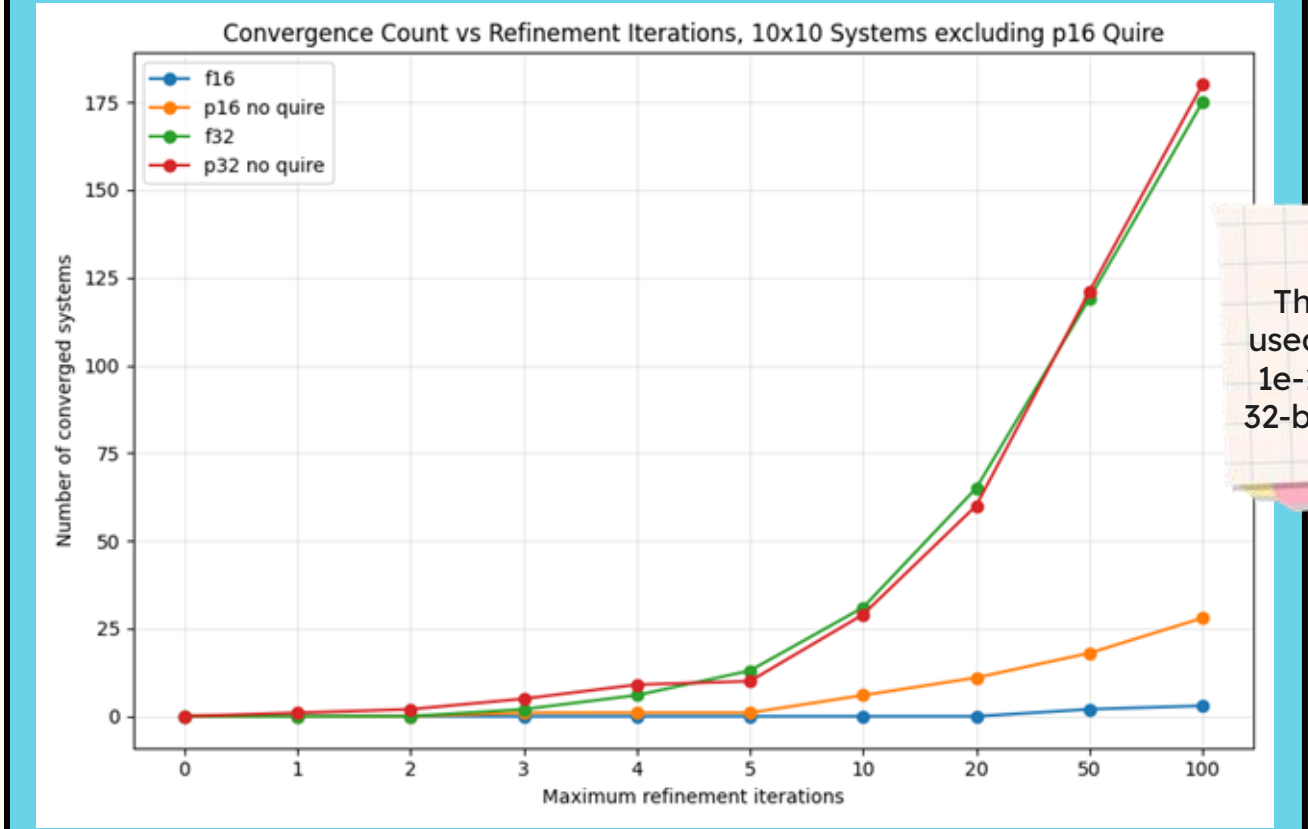

```
For each row of the matrix:
binade = math.floor(math.log2(abs(sum(row))))
scale = 2.0 ** (10 - binade)

For each element of the row:
purified row = round(float(element) * scale) / scale
exact_sum = sum(purified row) / scale
Check element == f16(element) and exact_sum == f16(exact_sum)

If not:
    scale = scale / 2.0 and repeat the purification process
```

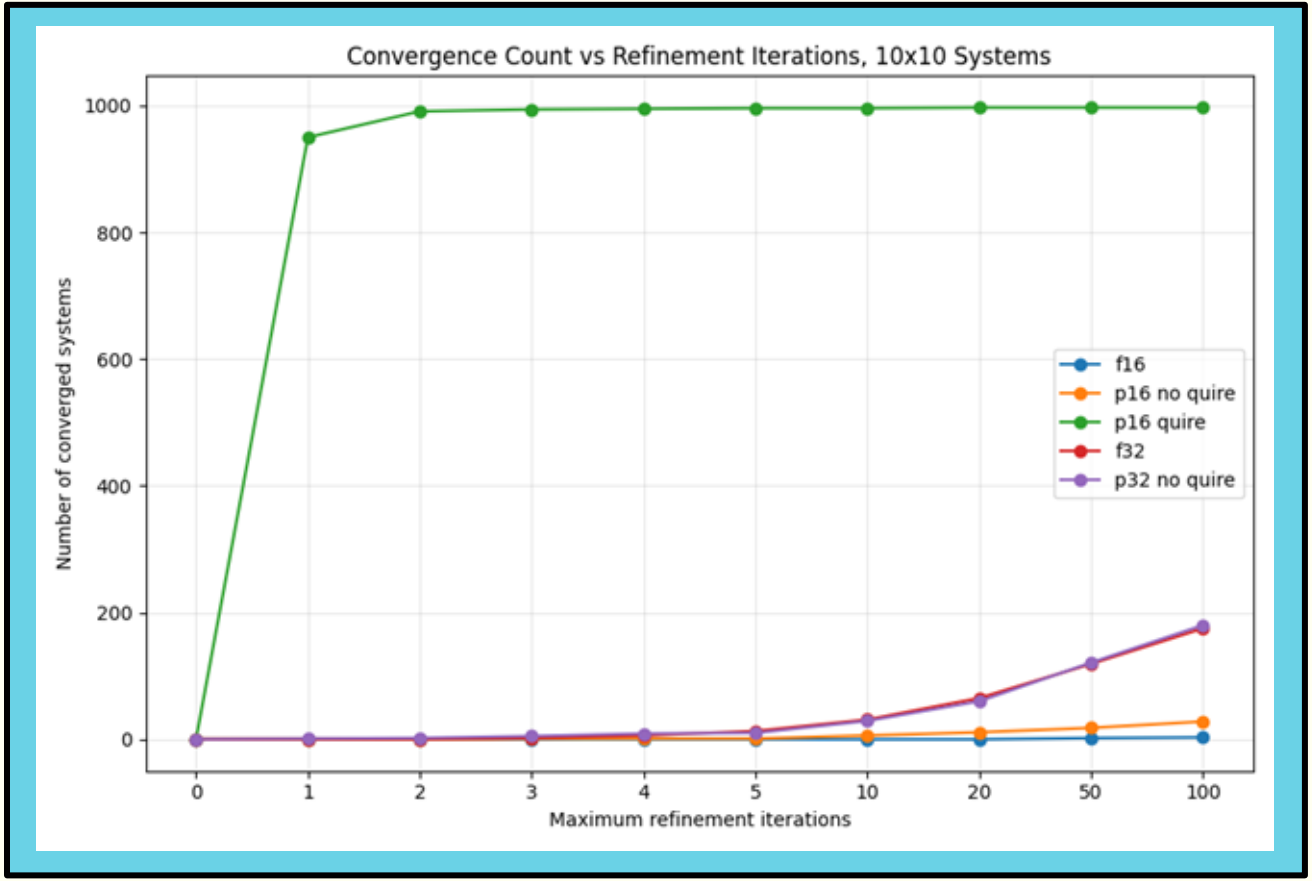
original value $\longrightarrow$ $\times 2^k$ $\longrightarrow$ round $\longrightarrow$ $\div 2^k$

Results: 10x10

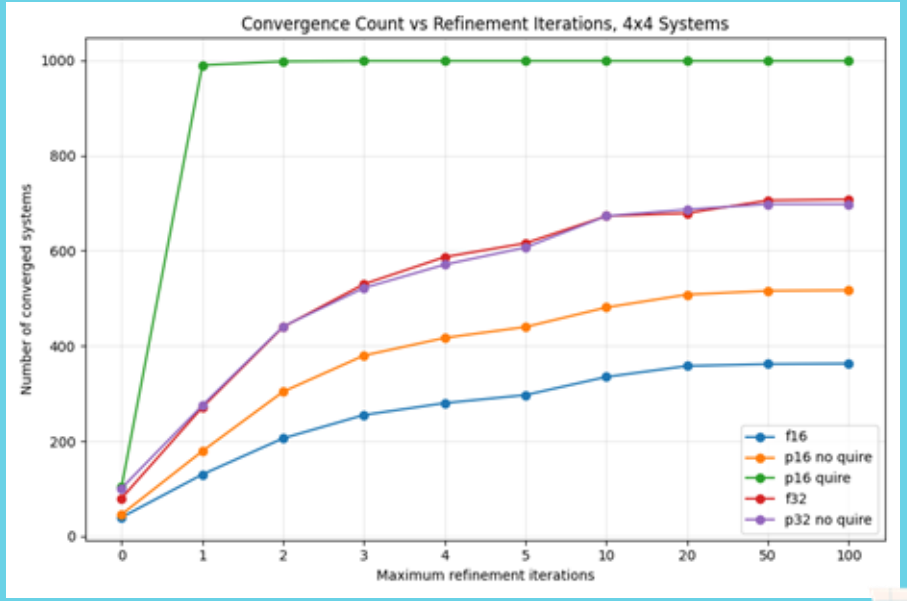


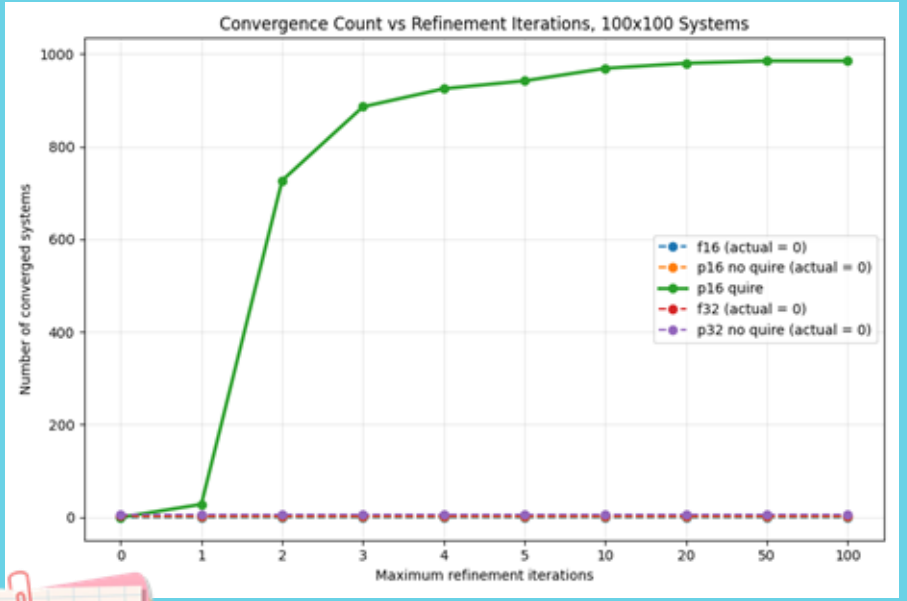
The tolerances used are 1e-6 and 1e-12 for 16- and 32-bit respectively

Results: 10x10



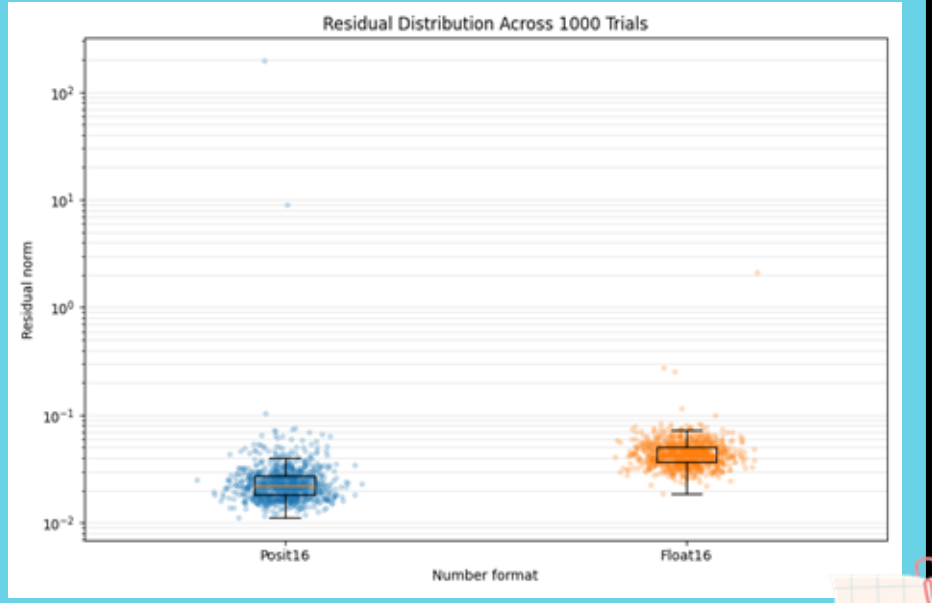
Results: Small and Large Systems

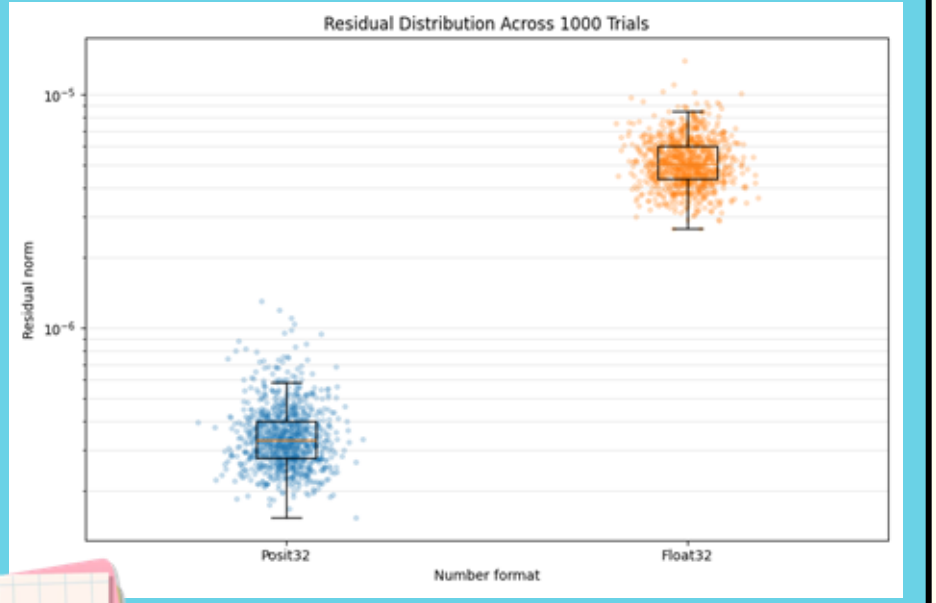




p16+Q staying near perfect!

Results: Residual Spread





Posits here do not use quire!

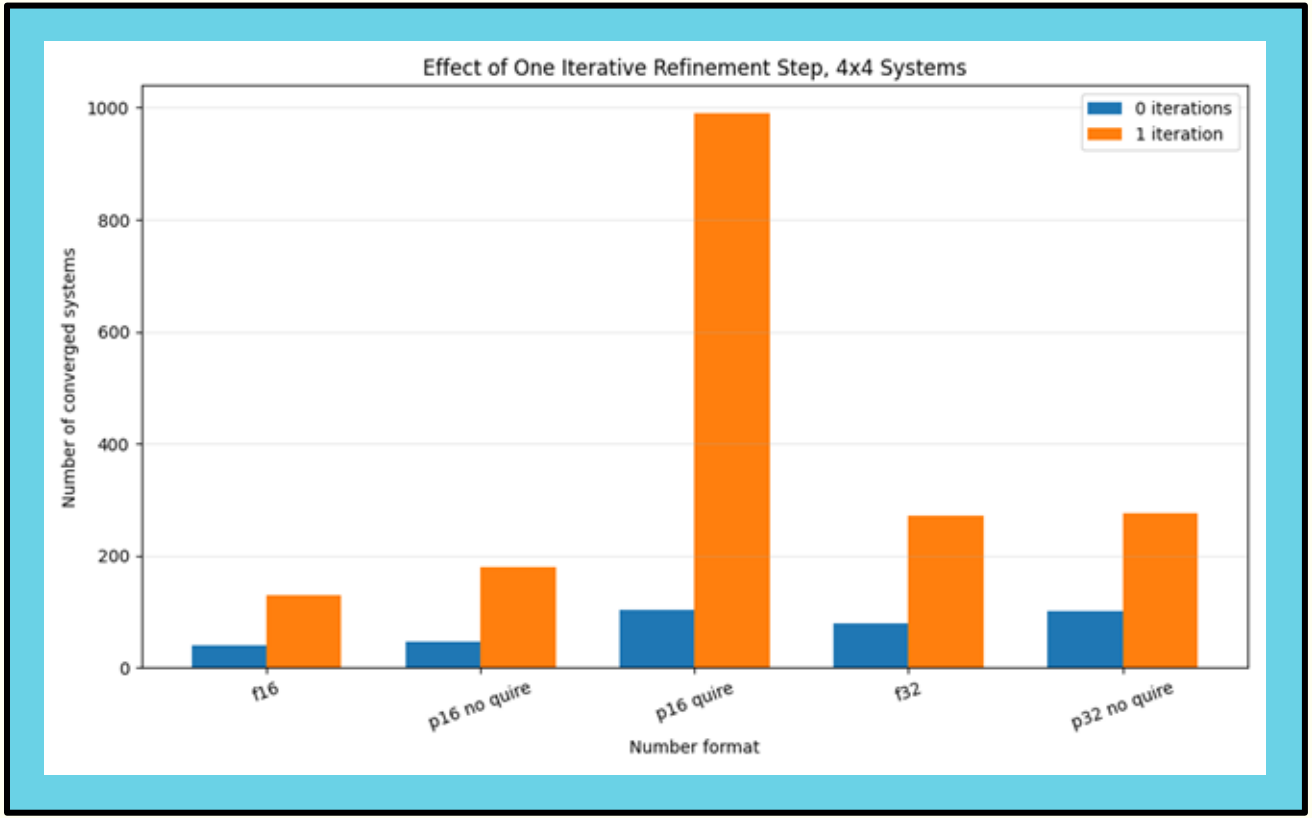
Iteration Ideals

Why even refine at all?

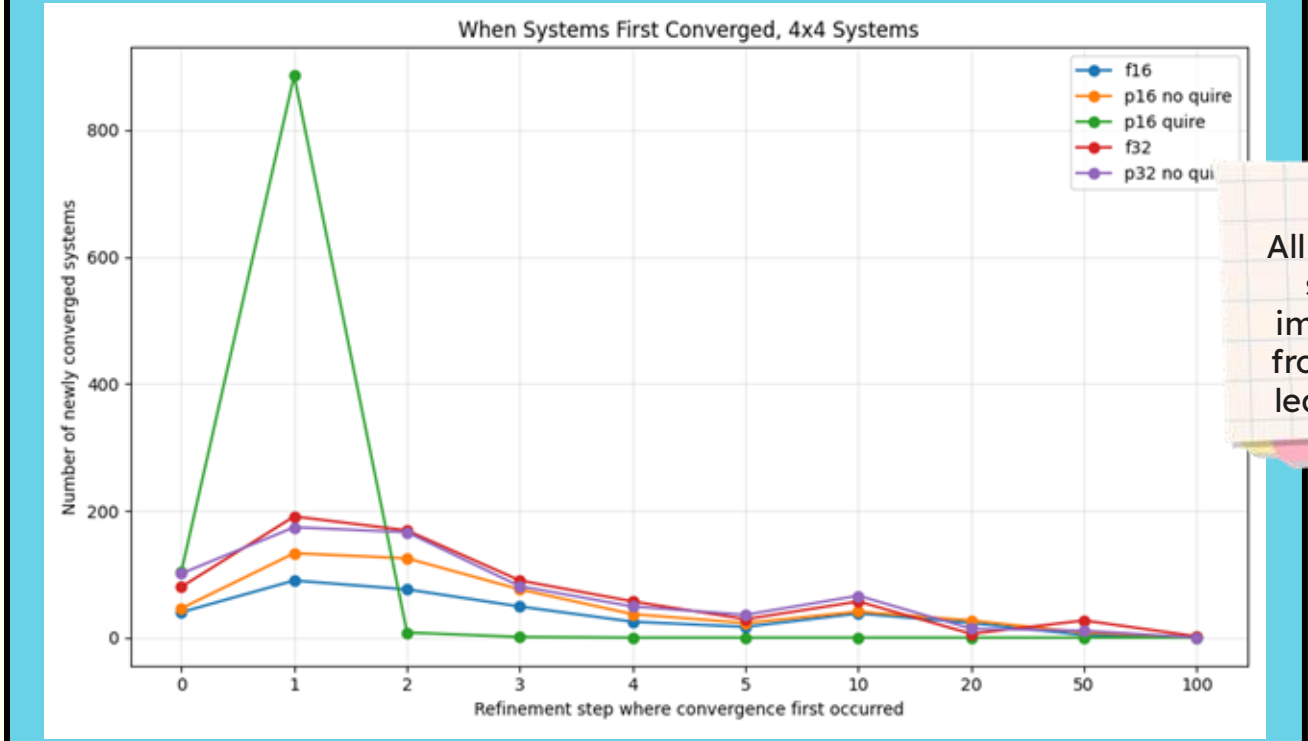
- Limited precision introduces rounding errors
- Initial solutions may not meet the desired accuracy
- Iterative refinement uses residuals to correct these errors
- Even 1–2 iterations can significantly improve accuracy

How much can just one or two refinement steps improve convergence?

Results: One Iterative Step

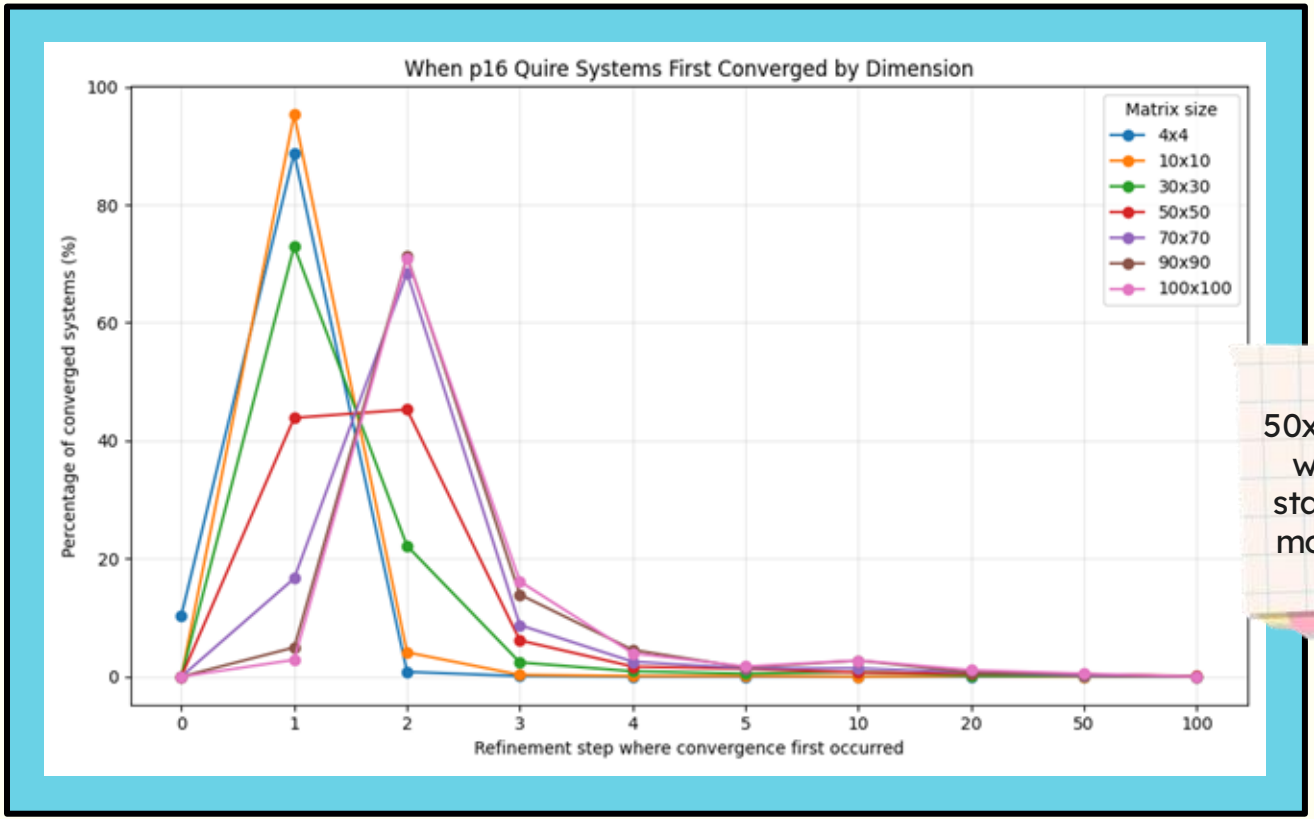


Results: Step Convergence 4x4



All formats saw significant improvements from having at least one step!

Results: Step Convergence p16+Q



50x50 matrices is where 2 steps starts becoming more impactful!

Scaling the Experimental Framework (1000x1000)

Methodology & Results

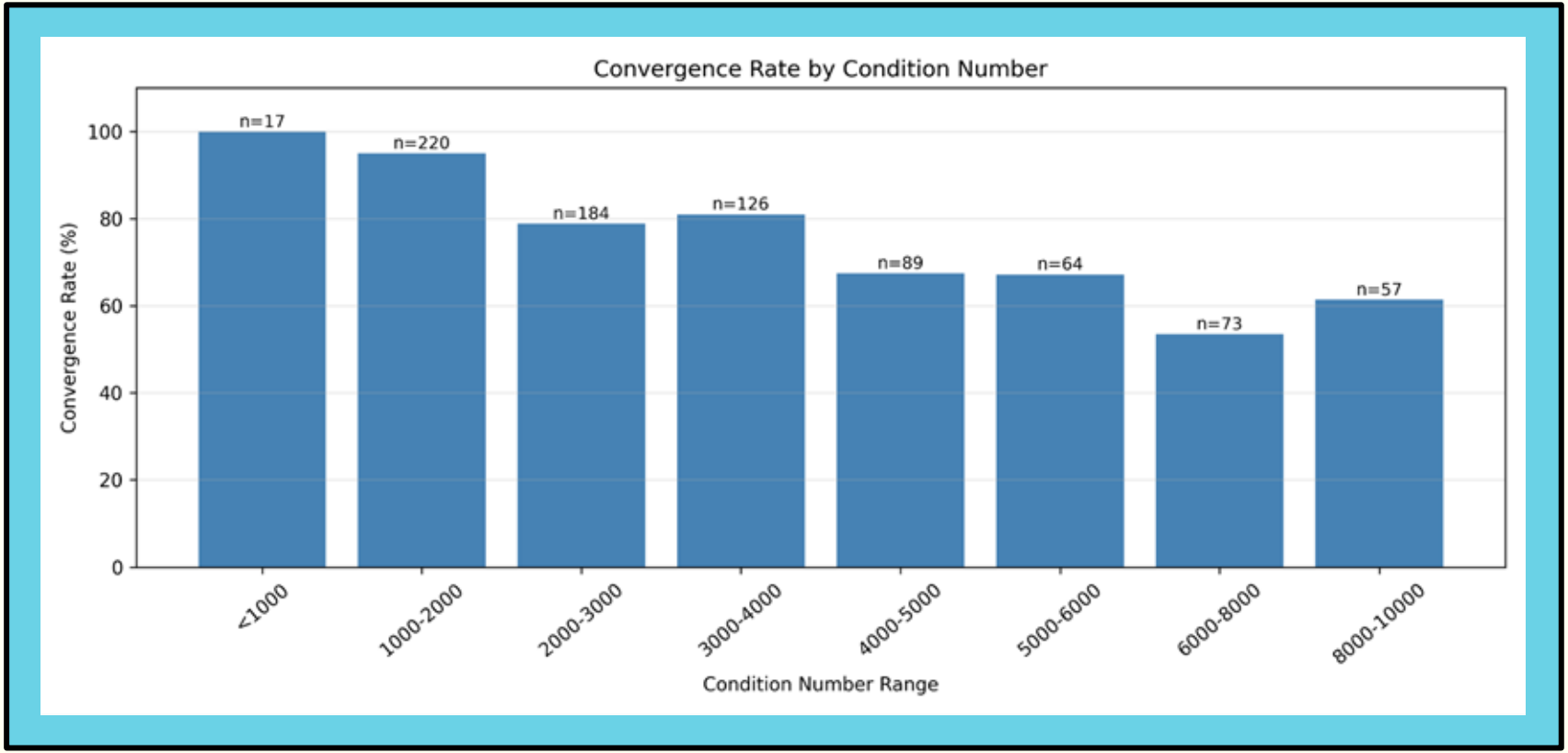
- Generated 1000 random matrices of size 1000×1000
- Converted matrix entries to 16-bit posits
- Excluded matrices with condition numbers ≥ 10,000
- Evaluated convergence within 100 iterations

$$830/1000 \text{ matrices retained}$$

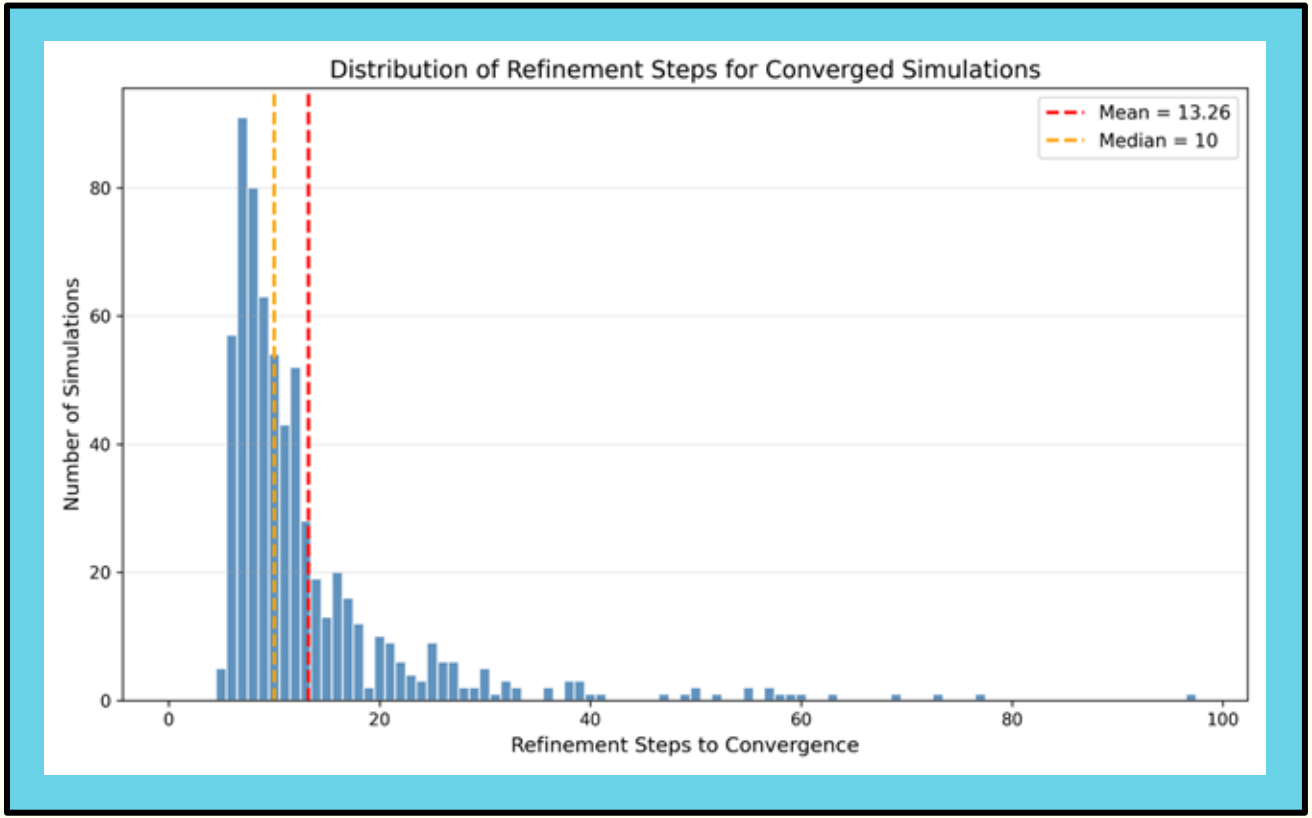
$$650/830 = 78.31\% \text{ converged}$$

$$533/636 = 83.81\% \text{ converged for } \kappa \leq 5000$$

Results: 1000x1000 Conv. Rate

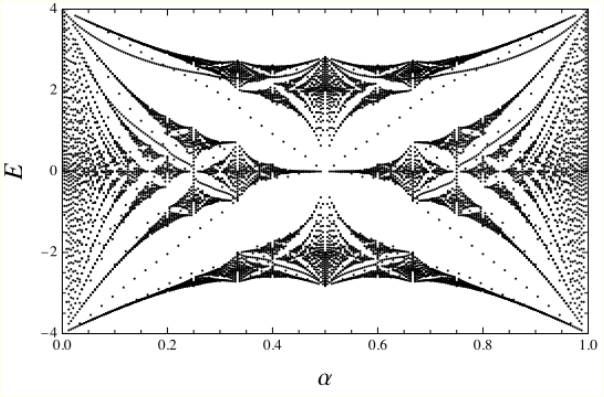


Results: 1000x1000 Conv. Steps



Future Work

- Larger systems
- Other Linear Algebra problems
    - Eigenvalue computations
    - Singular Value Decompositions



$$A = U\Sigma V^T$$

$$A = U \Sigma V^T$$

Conclusion

Exact answers to linear systems are possible with posit environments, even when floats fail!

Thank you!

Questions?

!