Ogita SVD Refinement: Stopping Criteria Analysis

Background

Ogita's iterative SVD refinement (RefSVD algorithm) improves an approximate SVD to arbitrary precision using extended arithmetic (BigFloat). The key question is: how many iterations are needed?

Stopping Criteria Compared

We tested three approaches:

1. **Fixed iterations (default 10)**: Run a fixed number of iterations
2. **Adaptive convergence check**: Stop when $\|F\|, \|G\| < 100\epsilon$ where F, G are the correction matrices
3. **Optimal iterations**: Use quadratic convergence theory to predict the required number of iterations: $\lceil \log_2(p/53) \rceil$ for p-bit precision

Quadratic Convergence Theory

Ogita's algorithm converges quadratically. Starting from Float64 precision (53 bits $\approx$ 15 decimal digits), each iteration approximately doubles the number of correct digits:

Iterations	Precision achieved	Target precision
1	$\sim 10^{-30}$ (30 digits)	—
2	$\sim 10^{-60}$ (60 digits)	128-bit (38 digits)
3	$\sim 10^{-120}$ (120 digits)	256-bit (77 digits)
4	$\sim 10^{-240}$ (240 digits)	512-bit (154 digits)
5	$\sim 10^{-480}$ (480 digits)	1024-bit (308 digits)

Formula: For target precision p bits, starting from 53 bits (Float64):

$$\text{iterations} = \left\lceil \log_2\left(\frac{p}{53}\right) \right\rceil$$

Experimental Results: Direct Comparison

Test matrix: $n = 50$ upper triangular, $\sigma_{\min} \approx 5.6 \times 10^{-8}$

Iterations	Time (s)	Residual norm	Notes
3	1.52	2×10^{-52}	Optimal for 256-bit
4	1.95	2×10^{-64}	Extra precision margin

Key finding: 3 iterations is 22% faster than 4 iterations, and both achieve residuals far smaller than $\sigma_{\min}$, yielding identical bounds.

Warm-Start Analysis

We tested whether reusing the SVD from a nearby point speeds up refinement.

With adaptive convergence check:

Radius	Cold iters	Warm iters	Cold time	Warm time
0.15	6.0	9.5	21.4s	32.8s
0.01	6.0	9.5	21.7s	32.4s
10^{-10}	6.0	8.4	21.9s	29.2s

Finding: Warm-start is *detrimental*! Fresh LAPACK SVD converges in 6 iterations; warm-start from a nearby (different) matrix needs 9.5 iterations.

With optimal iterations (fixed 3):

Radius	Cold iters	Warm iters	Cold time	Warm time
0.15	3.0	3.0	12.2s	11.9s
0.01	3.0	3.0	12.3s	12.2s
10^{-10}	3.0	3.0	12.2s	12.2s

Finding: With fixed iterations, warm-start provides no benefit (same work).

Why Adaptive Convergence Check is Slow

1. The convergence check computes spectral norm bounds of correction matrices—expensive.
2. The check is conservative: it requires 6 iterations when theory says 3 suffice.
3. Fresh LAPACK SVD is already machine-precision accurate for the current matrix.
4. Warm-start SVD from a different matrix is *less* accurate, requiring more iterations.

Implementation Update

Based on these findings, the default `max_ogita_iterations` in `CertifScripts.jl` has been updated:

```
# Old default (conservative)
max_ogita_iterations::Int = 4

# New default (optimal for 256-bit)
max_ogita_iterations::Int = 3
```

This provides a 22% speedup with no loss in bound quality.

Recommendations

1. **Use optimal iterations**: Set iteration count based on target precision using $\lceil \log_2(p/53) \rceil$.
2. **Don't use adaptive convergence check**: It's slower and more conservative than necessary.
3. **Don't use warm-start**: Fresh LAPACK SVD is the best starting point. Warm-start from a nearby matrix is actually worse.
4. **For 256-bit precision**: Use 3 iterations (default).
5. **For higher precision**: Use $\lceil \log_2(p/53) \rceil$ iterations.

Iteration Count Reference

Target precision (bits)	Iterations needed
128	2
256	3
512	4
1024	5
2048	6

Fast Extended Precision: The Hybrid Approach

The "Nonrigorous Oracle Then Certified" Pattern

Throughout BallArithmetic.jl, a common pattern emerges:

1. Compute fast approximate solution (Float64 oracle)
2. Refine to high precision (BigFloat iterations)
3. Certify with rigorous bounds (ball arithmetic)

This pattern appears in 13 components:

- SVD refinement (Ogita's RefSVD)
- Schur decomposition refinement
- Symmetric eigenvalue refinement (RefSyEv)
- Sylvester equation solving
- Linear system verification (H-matrix, inflation)
- Singular value bounds (Oishi 2023, Rump-Oishi 2024)
- Pseudospectra certification (3-tier caching)

The Double64 Opportunity

The key insight: **refinement iterations don't need rigorous arithmetic**. They only need extended precision. The rigor comes from the final certification.

Current approach:

$$\text{Float64} \xrightarrow{\text{3 iters}} \text{BigFloat (slow)} \xrightarrow{\text{certify}} \text{Rigorous bound}$$

Hybrid approach with Double64:

$$\text{Float64} \xrightarrow{\text{2 iters}} \text{Double64 (fast)} \xrightarrow{\text{1 iter}} \text{BigFloat} \xrightarrow{\text{certify}} \text{Rigorous bound}$$

Performance Comparison

Arithmetic	Bits	Relative Speed	Notes
Float64	53	1×	Native hardware
Double64	106	0.02×	DoubleFloats.jl
Float64x4	212	0.03×	MultiFloats.jl
BigFloat(256)	256	0.001×	GMP library

Double64 is **30–50× faster** than BigFloat for matrix operations.

Available Julia Libraries

- **DoubleFloats.jl**: Double64 (∼106 bits), full linear algebra support (SVD, LU, QR, eigen)
- **MultiFloats.jl**: Float64x2 to Float64x8 (106–424 bits), SIMD accelerated

Limitation: Neither provides faithful rounding. They cannot be used for rigorous ball arithmetic directly. But they *can* be used for fast refinement before final BigFloat certification.

Implementation

The `DoubleFloatsExt` extension provides fast Double64 implementations for:

SVD Refinement (Ogita's RefSVD):

- `ogita_svd_refine_fast()`: Double64 refinement with BigFloat certification
- `ogita_svd_refine_hybrid()`: 2 Double64 iterations + 1 BigFloat iteration

Schur Decomposition Refinement:

- `refine_schur_double64()`: Double64 Schur refinement
- `refine_schur_hybrid()`: 2 Double64 iterations + 1 BigFloat iteration

Symmetric Eigenvalue Refinement (RefSyEv):

- `refine_symmetric_eigen_double64()`: Double64 RefSyEv
- `refine_symmetric_eigen_hybrid()`: Hybrid version

Expected speedup: **10–30**× for the iterative refinement phase.

Oishi 2023: Schur Complement Bounds for $\sigma_{\min}$

Background

Oishi (2023) presents a method for computing lower bounds on the minimum singular value $\sigma_{\min}(G)$ of matrices arising from linearized Galerkin equations. The key insight is that these matrices have a "generalized asymptotic diagonal dominant" structure.

Theorem 1 (Oishi 2023)

Partition G as a 2×2 block matrix:

$$G = \begin{pmatrix} A & B \\ C & D \end{pmatrix}$$

where $A \in M_m$, and let D_d, D_f be the diagonal and off-diagonal parts of D.

If the following conditions hold:

1. $\|A^{-1}B\|_2 < 1$
2. $\|CA^{-1}\|_2 < 1$
3. $\|D_d^{-1}(D_f - CA^{-1}B)\|_2 < 1$

Then G is invertible and:

$$\|G^{-1}\|_2 \leq \frac{\max\left\{\|A^{-1}\|_2, \frac{\|D_d^{-1}\|_2}{1-\|D_d^{-1}(D_f-CA^{-1}B)\|_2}\right\}}{(1-\|A^{-1}B\|_2)(1-\|CA^{-1}\|_2)}$$

Computational Pattern

The key computations are:

- A^{-1} (or equivalently, solving $AX = B$ for $X = A^{-1}B$)
- CA^{-1} (or equivalently, solving $A^T X = C^T$ for $X = (CA^{-1})^T$)
- Matrix products $D_d^{-1}(D_f - CA^{-1}B)$

These computations benefit from the Double64 oracle pattern:

1. **Oracle**: Solve $AX = B$ in Double64 (fast)
2. **Certify**: Compute residual $R = B - AX$ in BigFloat (rigorous)
3. **Bound**: Use $\|X - X_{\text{exact}}\| \leq \|A^{-1}\| \cdot \|R\|$

Implementation

The `DoubleFloatsExt` extension provides:

- `oishi_2023_solve_double64()`: Fast linear system solve with certification
- `oishi_2023_sigma_min_bound_fast()`: Full Schur complement method with Double64 oracles

Summary: Where to Apply the Double64 Pattern

Algorithm	Function	Expected Speedup
Ogita SVD (RefSVD)	`ogita_svd_refine_hybrid`	10–30×
Schur refinement	`refine_schur_hybrid`	10–30×
Symmetric eigen (RefSyEv)	`refine_symmetric_eigen_hybrid`	10–30×
Oishi 2023 Schur complement	`oishi_2023_sigma_min_bound_fast`	5–10×

Key insight: Iterative refinement doesn't need rigorous arithmetic—it just needs extended precision. The rigor comes from the final certification step.