

MIXED PRECISION COMPUTING:
ITERATIVE REFINEMENT AND OZAKI MATRIX MULTIPLICATION

Per-Gunnar Martinsson, University of Texas at Austin

Text produced with LLM assistance.

1 Why mixed precision?

Modern processors often perform low-precision arithmetic much faster than high-precision arithmetic. This is especially true for matrix operations. The natural numerical-analysis question is therefore:

Can we perform most of the expensive work in low precision, while still obtaining a high-precision answer?

This note describes two ways to answer this in the affirmative:

1. **Iterative refinement:** first compute a cheap low-precision approximation to the answer, then use high-precision residuals to repair it.
2. **Ozaki-type matrix multiplication:** decompose a high-precision matrix product into a collection of carefully chosen low-precision matrix products, and recombine their outputs accurately.

The two methods illustrate a broader principle: *precision can be treated as an algorithmic resource*. Not every part of a computation needs to be carried out at the final requested precision.

Throughout the first part of the note, we use two unit roundoffs,

$$u_{\text{low}} \gg u_{\text{high}},$$

corresponding to a low and a high precision. As a rough guide, we may think of $u_{\text{low}} \approx 10^{-7}$ as IEEE single precision, and $u_{\text{high}} \approx 10^{-16}$ as IEEE double precision.

2 Iterative refinement

2.1 The basic setup

Suppose we wish to solve

$$Ax = b, \quad A \in \mathbb{R}^{n \times n},$$

and suppose that the desired answer is a high-precision answer.

Assume that, using low-precision arithmetic, we have already computed a matrix

$$B \approx A^{-1}.$$

For the moment, we deliberately leave open how this approximate inverse was obtained. We will also treat the application of B as our low-precision operation. This abstraction makes the basic idea particularly transparent.

A natural first approximation is

$$x_0 = Bb.$$

If B were the exact inverse, we would of course be finished. But if B contains only low-precision information, then x_0 will generally contain only low-precision accuracy.

The key observation is that the error can be turned into another linear system. Let

$$x_\star = A^{-1}b$$

denote the exact solution, and define

$$e_k = x_\star - x_k.$$

Then

$$Ae_k = A(x_\star - x_k) = b - Ax_k = r_k.$$

Therefore the exact correction would be

$$e_k = A^{-1}r_k.$$

We do not have A^{-1} , but we do have $B \approx A^{-1}$. This suggests the iteration

$$\begin{aligned} r_k &= b - Ax_k, \\ d_k &= Br_k, \\ x_{k+1} &= x_k + d_k. \end{aligned}$$

The important mixed-precision choice is:

- compute the residual $r_k = b - Ax_k$ in **high precision**;
- apply the approximate inverse B in **low precision**;
- accumulate the update $x_{k+1} = x_k + d_k$ in **high precision**.

Thus the expensive object B may have been constructed using low precision, and it continues to be applied in low precision, but high precision is used where it is most valuable: in measuring what is still wrong and in storing the improved answer.

2.2 Error propagation

First ignore all rounding errors except for the fact that B is only an approximate inverse. Since

$$r_k = Ae_k,$$

the correction is

$$d_k = BAe_k.$$

Consequently,

$$e_{k+1} = x_\star - x_{k+1} = (x_\star - x_k) - (x_{k+1} - x_k) = e_k - d_k = (I - BA)e_k.$$

Hence

$$e_{k+1} = (I - BA)e_k.$$

This is the central formula. Iterative refinement works when B is sufficiently close to A^{-1} that

$$I - BA$$

is a contraction. A convenient sufficient condition is

$$\|I - BA\| < 1.$$

Then

$$\|e_k\| \leq \|I - BA\|^k \|e_0\|.$$

This interpretation is useful: the matrix B does not need to be accurate enough to give the final answer directly. It only needs to be accurate enough to reduce the error by a reliable factor on each refinement step.

2.3 Where conditioning enters

Why should a low-precision computation of an inverse produce a contraction at all? A useful model is to suppose that the low-precision computation effectively produces the exact inverse of a nearby matrix:

$$B = (A + \Delta A)^{-1}, \quad \|\Delta A\| \leq u_{\text{low}} \|A\|.$$

Then

$$I - BA = B(B^{-1} - A) = B(A + \Delta A - A) = B \Delta A.$$

Therefore

$$\|I - BA\| \leq \|B\| \|\Delta A\|.$$

If the approximation is good enough that $\|B\| \approx \|A^{-1}\|$, then we further get

$$\|I - BA\| \lesssim \|A^{-1}\| \|\Delta A\| \leq \|A^{-1}\| \|A\| u_{\text{low}} = \kappa(A) u_{\text{low}}.$$

Consequently

$$\boxed{\|I - BA\| \lesssim \kappa(A) u_{\text{low}}}.$$

This leads to the rule of thumb

$$\boxed{\kappa(A) u_{\text{low}} < 1}$$

for ordinary mixed-precision refinement to have a reasonable chance of converging. This is not a sharp theorem in this form—the details depend on the norm, the low-precision solver, scaling, and the precise rounding-error model—but it captures the main phenomenon.

As an illustration, suppose that u_{low} represents single precision, so that $u_{\text{low}} \approx 10^{-7}$, and that A is relatively well conditioned; say $\kappa(A) \leq 10$. Then the contraction factor $\|I - BA\|$ would be about 10^{-6} , and we would gain six correct digits with each step. In other words, we should comfortably reach double precision accuracy in two refinement steps.

By contrast, if

$$\kappa(A) u_{\text{low}} \gtrsim 1,$$

the low-precision inverse may simply be too inaccurate to define a useful correction step. Refinement may converge very slowly, stagnate, or diverge.

2.4 Why the residual must be computed accurately

The mixed-precision scheme may initially look paradoxical. If the correction $d_k = Br_k$ is computed using low precision, how can the final answer contain many more correct digits than the low-precision arithmetic itself?

The answer is that the correction becomes smaller as the approximation improves.

Suppose x_k is already quite accurate. Then

$$r_k = b - Ax_k = Ae_k$$

is small, and therefore

$$d_k \approx e_k$$

is small as well. A low-precision computation of d_k may have a relative error of order u_{low} , but its *absolute* error is proportional to the size of d_k . As the correction shrinks, the absolute low-precision rounding error in that correction shrinks with it.

This is precisely why the residual and the update must be treated differently.

High-precision residual. The residual

$$r_k = b - Ax_k$$

measures the remaining defect in the current high-precision iterate. If it were formed only in low precision, then once x_k became accurate to roughly low precision, the subtraction could no longer reliably expose the much smaller remaining defect. Computing Ax_k and the subtraction in high precision allows the iteration to keep seeing errors far below the low-precision scale.

Strictly speaking, when r_k is extremely small, the computed residual need not have small *relative* error as a residual. A standard high-precision matrix-vector product instead gives a small *absolute* error of size roughly

$$O(u_{\text{high}})(\|A\| \|x_k\| + \|b\|).$$

That is enough to drive the iteration down to a high-precision backward-error floor.

High-precision accumulation. The update

$$x_{k+1} = x_k + d_k$$

should also be accumulated in high precision. Near convergence, d_k may be far smaller than x_k . If the addition were carried out only in low precision, those small corrections could simply disappear under rounding.

The logic can therefore be summarized as

measure the remaining error accurately,
approximately solve for a correction cheaply,
accumulate the correction accurately.

2.5 A simple finite-precision model

The exact error recursion

$$e_{k+1} = (I - BA)e_k$$

is useful, but a real implementation contains additional rounding errors. A schematic model is

$$\hat{r}_k = b - Ax_k + \eta_k,$$

where the high-precision residual error satisfies roughly

$$\|\eta_k\| \lesssim u_{\text{high}}(\|A\| \|x_k\| + \|b\|),$$

and

$$\hat{d}_k = B\hat{r}_k + \zeta_k,$$

where the low-precision correction error is proportional to the correction calculation and hence, roughly, to the size of the residual.

Ignoring the final high-precision addition error for the moment gives

$$e_{k+1} \approx (I - BA)e_k - B\eta_k - \zeta_k.$$

The first term decreases geometrically when B defines a contraction. The other terms eventually stop the convergence. Thus one typically sees

rapid geometric improvement at first, followed by stagnation near the accuracy permitted by the high-precision arithmetic and the conditioning of the problem.

For a well-conditioned problem, the final relative forward error can be of order u_{high} . More generally, even a tiny backward error may be amplified by the condition number, so the attainable forward accuracy is naturally tied to a scale such as

$$\kappa(A)u_{\text{high}}.$$

2.6 Stopping criteria

Next let us consider the question of when to stop the iteration. Since we compute the residual r_k at every step, a simple and scale-invariant criterion is to monitor the *relative residual*

$$\frac{\|r_k\|}{\|b\|}$$

and stop once it reaches the scale of the high-precision arithmetic. For instance, suppose that we stop when

$$\frac{\|r_k\|}{\|b\|} \leq u_{\text{high}}.$$

Let us see what this implies for the forward error. Since

$$e_k = x_\star - x_k = A^{-1}(b - Ax_k) = A^{-1}r_k,$$

we have

$$\frac{\|e_k\|}{\|x_\star\|} \leq \frac{\|A^{-1}\| \|r_k\|}{\|x_\star\|}.$$

Moreover,

$$\|b\| = \|Ax_\star\| \leq \|A\| \|x_\star\|,$$

so

$$\frac{1}{\|x_\star\|} \leq \frac{\|A\|}{\|b\|}.$$

Combining the two bounds gives

$$\boxed{\frac{\|x_k - x_\star\|}{\|x_\star\|} \leq \kappa(A) \frac{\|r_k\|}{\|b\|}.$$

Therefore, if the iteration is stopped when

$$\frac{\|r_k\|}{\|b\|} \leq u_{\text{high}},$$

then

$$\frac{\|x_k - x_\star\|}{\|x_\star\|} \leq \kappa(A) u_{\text{high}}.$$

Thus for a well-conditioned problem, driving the relative residual down to the high-precision roundoff level also gives a solution that is accurate to roughly that level. For an ill-conditioned problem, however, the forward error may be larger by a factor comparable to $\kappa(A)$. This is not a flaw of iterative refinement; it is the standard sensitivity of the linear system $Ax = b$.

2.7 Cost

The attraction of mixed-precision refinement is clearest when the expensive setup can be done much faster in low precision.

In the simplified model where B is an explicitly available approximate inverse,

- constructing B costs on the order of n^3 operations;
- applying B to a residual costs on the order of n^2 operations;
- computing the high-precision residual $b - Ax_k$ also costs on the order of n^2 operations.

Thus we replace an expensive high-precision $O(n^3)$ computation by

$$\boxed{\text{one low-precision } O(n^3) \text{ setup} + \text{a few } O(n^2) \text{ refinement steps.}}$$

If low-precision arithmetic is substantially faster, and only a small number of correction steps is needed, this can be a major win.

The same idea is especially attractive for multiple right-hand sides: the low-precision setup is reused, while each new right-hand side requires only low-cost solves and high-precision residual calculations.

2.8 Practical complications and extensions

Several issues matter in real implementations.

- **Conditioning.** If the low-precision approximation is not good enough to make $I - BA$ contractive, ordinary refinement may fail. This is the most basic limitation.
- **Scaling and dynamic range.** Low-precision formats may have limited exponent ranges, and badly scaled matrices can cause overflow, underflow, or an unnecessarily poor approximate inverse. Row and column scaling can therefore be important.
- **Residual accuracy.** The residual calculation is the high-precision “truth check” in the method. It should not be silently demoted to the same low precision as the approximate solve.
- **Data movement.** On modern hardware, the speedup is not determined by flop counts alone. Converting data between formats and moving vectors and matrices through memory can be significant costs.
- **Krylov acceleration.** If B is a useful approximate inverse but the simple stationary iteration

$$x_{k+1} = x_k + B(b - Ax_k)$$

is not sufficiently robust, one can instead use B as a preconditioner inside a Krylov method. This leads to more sophisticated forms of iterative refinement.

- **Adaptive precision.** There is no requirement that only two precisions be used. One may begin with very cheap arithmetic and increase precision only if convergence stalls.

2.9 Remark: in practice we usually do not form the inverse

The discussion above was deliberately phrased in terms of an approximate inverse B because that makes the numerical idea transparent. In practical dense linear algebra, one usually does *not* explicitly compute and store A^{-1} .

Instead, one computes a factorization in low precision—for example a QR factorization or an LU factorization—and applies the approximate inverse implicitly by solving with the factors. The refinement loop is unchanged conceptually:

high-precision residual $\longrightarrow$ low-precision correction solve $\longrightarrow$ high-precision update.

The approximate-inverse notation should therefore be viewed as a clean model for the correction operator rather than as a recommendation to form the inverse explicitly.

2.10 A second interpretation: preconditioned Richardson iteration

There is another useful way to view the same algorithm. The update

$$x_{k+1} = x_k + B(b - Ax_k)$$

is simply Richardson iteration preconditioned by B . The iteration matrix is

$$I - BA.$$

Mixed-precision iterative refinement is therefore not a mysterious new iteration. What is special is the division of labor between precisions:

- the preconditioner is constructed and applied cheaply;
- the residual is evaluated accurately;
- the iterate is stored and updated accurately.

This viewpoint will recur in many modern algorithms: low precision is used where approximation is acceptable, and high precision is reserved for error measurement and certification.

3 Ozaki-type high-precision matrix multiplication

Iterative refinement uses low precision to obtain an approximate solution and then repairs that solution. The Ozaki idea takes a different route. Its goal is to perform an intrinsically high-precision operation—matrix multiplication—by decomposing it into many low-precision matrix multiplications whose outputs can be recombined accurately.

The basic idea is easiest to understand by thinking about ordinary long multiplication.

3.1 Splitting a scalar into blocks of digits

Suppose that we wish to multiply two numbers with roughly 16 digits of accuracy, but our fast arithmetic works with only about 8 digits. Write

$$x = 10^d(x_1 + \beta x_2), \quad y = 10^e(y_1 + \beta y_2), \quad \beta = 10^{-8},$$

where x_1, x_2, y_1, y_2 each contain about 8 significant digits. Then

$$xy = 10^{d+e} [x_1 y_1 + \beta(x_1 y_2 + x_2 y_1) + \beta^2 x_2 y_2].$$

The three groups of terms contribute at successively smaller scales,

$$1, \quad 10^{-8}, \quad 10^{-16}.$$

This is simply long multiplication using blocks of digits rather than one digit at a time.

There is, however, an immediate problem. If x_1 and y_1 each contain 8 digits, then $x_1 y_1$ may contain 16 digits. Ordinary 8-digit arithmetic cannot compute that product exactly.

3.2 Make the slices narrower

Suppose instead that the fast arithmetic can represent 8 significant digits accurately, and split the original numbers into blocks containing only 4 digits. Let

$$\beta = 10^{-4},$$

and write

$$x = 10^d (x_0 + \beta x_1 + \beta^2 x_2 + \beta^3 x_3),$$

with a similar expression for y . Now each product $x_i y_j$ contains at most 8 digits and can be computed exactly in this idealized arithmetic.

Expanding gives

$$\begin{aligned}
xy = 10^{d+e} [& x_0y_0 \\
& + \beta(x_0y_1 + x_1y_0) \\
& + \beta^2(x_0y_2 + x_1y_1 + x_2y_0) \\
& + \beta^3(x_0y_3 + x_1y_2 + x_2y_1 + x_3y_0) \\
& + \beta^4(\dots)].
\end{aligned}$$

If we seek roughly 16 digits of accuracy, terms at the scale $\beta^4 = 10^{-16}$ and smaller may be discarded.

The significant terms lie on successive “anti-diagonals”:

$$x_0y_0,$$

then

$$x_0y_1 + x_1y_0,$$

then

$$x_0y_2 + x_1y_1 + x_2y_0,$$

and so forth.

Thus a high-precision multiplication has been replaced by a collection of exact low-precision multiplications followed by accurate recombination.

3.3 Integer arithmetic gives a particularly clean model

Suppose each slice is represented by a 4-digit integer. The product of two slices then contains at most 8 digits. For example,

$$5678 \times 9012 = 51190136.$$

One can therefore imagine hardware that accepts narrow integer inputs but returns the wider product exactly.

For example,

$$1.234567890123456 = 10^{-3} (1234 + 10^{-4}5678 + 10^{-8}9012 + 10^{-12}3456) .$$

The integer block products can be formed exactly, the appropriate powers of 10^{-4} restored, and the results recombined in high precision.

This cleanly separates the precision of the *inputs* from the precision of the *product*:

low-precision inputs	→	wider, exact products.
----------------------	---	------------------------

3.4 What changes for matrices?

Now consider matrix multiplication. Write, schematically,

$$A = A_1 + \beta A_2, \quad B = B_1 + \beta B_2.$$

Then

$$AB = A_1B_1 + \beta(A_1B_2 + A_2B_1) + \beta^2 A_2B_2.$$

Algebraically, this is exactly the same expansion as for scalar multiplication.

The new issue is that an entry of $A_1 B_1$ is not a single multiplication. It is a dot product:

$$(A_1 B_1)_{ij} = \sum_{k=1}^n (A_1)_{ik} (B_1)_{kj}.$$

Even if every individual product is exact, the *sum* of those products may require additional digits, in particular if n is large.

For example, let

$$A_1 = B_1 = \begin{bmatrix} 0.9999 & 0.9999 \\ 0.9999 & 0.9999 \end{bmatrix}.$$

Each scalar product is

$$0.9999^2 = 0.99980001,$$

which contains 8 significant digits. But

$$A_1 B_1 = \begin{bmatrix} 1.99960002 & 1.99960002 \\ 1.99960002 & 1.99960002 \end{bmatrix},$$

and each entry now contains 9 significant digits.

For an n -term dot product, the worst-case growth in the magnitude of a sum is a factor of n . Roughly speaking, this costs an additional

$$\log_{10} n$$

decimal digits, or

$$\log_2 n$$

binary bits, of accumulator headroom.

Thus the slice width cannot be chosen by considering scalar multiplication alone. It must also account for accumulation inside the dot products.

3.5 Low-precision inputs with a wider accumulator

Modern matrix-multiplication hardware often provides precisely the operation needed here: low-precision inputs whose products are accumulated in a wider format. Schematically,

$$\boxed{\text{low precision} \times \text{low precision} \longrightarrow \text{higher-precision accumulation.}}$$

For example, one can think of half-precision-like inputs whose products are accumulated in single precision. The wider accumulator provides room for the growth that occurs when many products are added together.

A wider accumulator is not by itself sufficient, however. The slices must be narrow enough that the slice products and their dot-product accumulation are represented accurately enough for the final recombination.

This gives the basic Ozaki strategy:

1. Split each high-precision number into several lower-precision slices.
2. Make the slices narrow enough that products of two slices, and the accumulation of those products, can be carried out accurately in the available hardware.
3. Compute matrix products between the slices using fast low-precision GEMMs.
4. Rescale and combine the resulting matrices in the desired high precision.

As the matrix dimension increases, or as the desired final precision increases, more slices may be required.

3.6 The matrix-product hierarchy

Suppose

$$A = A_0 + A_1 + A_2 + \cdots, \quad B = B_0 + B_1 + B_2 + \cdots,$$

where A_0 and B_0 contain the leading digits, A_1 and B_1 the next group of digits, and so forth. Then

$$AB = \sum_{i,j} A_i B_j.$$

The products form a natural hierarchy:

$$\begin{array}{cccc} A_0 B_0 & A_0 B_1 & A_0 B_2 & \cdots \\ A_1 B_0 & A_1 B_1 & A_1 B_2 & \cdots \\ A_2 B_0 & A_2 B_1 & A_2 B_2 & \cdots \\ \vdots & \vdots & \vdots & \ddots \end{array}$$

The leading contribution is

$$A_0 B_0.$$

The next level is

$$A_0 B_1 + A_1 B_0,$$

followed by

$$A_0 B_2 + A_1 B_1 + A_2 B_0,$$

and so forth.

The terms therefore lie on successive anti-diagonals. Terms farther down and to the right represent successively less significant information. Once an entire group is too small to affect the requested final accuracy, it can be omitted.

The basic computational transformation is

$\begin{array}{l} \text{high-precision GEMM} \\ \longrightarrow \text{several fast low-precision GEMMs} + \text{high-precision recombination} \end{array}$
--

If s slices are retained for each input and all pairwise slice products are formed, there are s^2 low-precision GEMMs. Truncating by anti-diagonals can reduce this count substantially. The number of GEMMs required in practice depends on the target precision, the input format, the accumulator precision, the inner dimension, scaling, and the amount of truncation that can safely be used. The method is attractive only when the low-precision GEMMs are sufficiently faster that this multiplication count is worthwhile.

3.7 Practical details

A real Ozaki implementation must handle several issues that the simple examples suppress.

- **Scaling.** Matrix entries can vary greatly in magnitude. Rows of A and columns of B can be scaled so that significant digits are split efficiently into low-precision slices.
- **Slice width.** Slices must be narrow enough that their products, and the accumulation of those products in a dot product, fit accurately in the available accumulator.

- **More slices when needed.** Higher target precision and longer dot products generally require more slices.
- **Blocking.** For very long dot products, the inner dimension can be divided into blocks. Each block product is computed separately, and block results are then accumulated in higher precision. This limits the growth that must be accommodated by the lower-precision accumulation.
- **Mixed-precision hardware.** Matrix units that accept low-precision inputs while accumulating in a wider format are particularly well matched to the method.
- **Integer variants.** Some implementations use narrow integer inputs together with wider integer accumulators. In that setting, the slice products can be interpreted literally as exact integer matrix products.
- **Truncation.** It is often unnecessary to compute every $A_i B_j$. Once later anti-diagonals are too small to affect the requested accuracy, they can be skipped.
- **Final accumulation.** The low-precision GEMMs perform the expensive work, but their outputs must be rescaled and combined accurately—typically in the target precision or with another accurate accumulation strategy.

The essential idea is therefore not tied to one floating-point format or one piece of hardware. It is the combination

splitting + exact or highly accurate low-precision products + accurate recombination.

4 Two different mixed-precision philosophies

It is useful to end by comparing the two methods.

	Iterative refinement	Ozaki multiplication
Expensive low-precision work	Approximate inverse / solve	Many low-precision GEMMs
High-precision work	Residual and update	Rescaling and recombination
Main mechanism	Detect and repair the remaining error	Decompose high-precision arithmetic into narrow pieces
Main limitation	Approximate inverse must be good enough to contract	Enough slice products must be computed accurately for the desired precision

The common principle is that high precision is used selectively rather than uniformly. In iterative refinement, we can summarize the strategy as

compute cheaply, measure accurately, repair.

For Ozaki multiplication, the corresponding summary is

split, compute cheaply, recombine accurately.

These are two examples of a much broader trend in numerical linear algebra: the arithmetic precision itself has become a design parameter of the algorithm.