INTERPRECISION TRANSFERS IN ITERATIVE REFINEMENT

C. T. KELLEY*

Abstract. We make the interprecision transfers explicit in an algorithmic description of iterative refinement and obtain new insights into the algorithm. One example is the classic variant of iterative refinement where the matrix and the factorization are stored in a working precision and the residual is evaluated in a higher precision. In that case we make the observation that this algorithm will solve a promoted form of the original problem and thereby characterize the limiting behavior in a novel way and obtain a different version of the classic convergence analysis. We also discuss two approaches for interprecision transfer in the triangular solves.

Key words. Iterative refinement, Interprecision transfers, Mixed-precision arithmetic, Linear systems

AMS subject classifications. 65F05, 65F10,

1. Introduction.

Iterative refinement (IR) is a way to lower factorization costs in the numerical solution of a linear system $\mathbf{A}\mathbf{x} = \mathbf{b}$ by performing the factorization in a lower precision.

IR is also used to address ill-conditioning by storing the data for the problem in one precision and evaluating the residual in a higher precision. IR was invented [22] with that application in mind.

Algorithm **IR-V0** is a simple formulation using Gaussian elimination. In this formulation the factorization is done in a lower precision that the residual computation, but the precisions of the other components of the algorithm are not specified. The iteration terminates on a small residual norm. Another approach [6,9] is to terminate when the normwise backward error is small, *i. e.*

$$\|\mathbf{r}\| < C_e u_r(\|\mathbf{b}\| + \|\mathbf{A}\|\|\mathbf{x}\|). \tag{1.1}$$

```
IR-V0($\mathbf{A}, \mathbf{b}$)
  $\mathbf{x} = 0$
  $\mathbf{r} = \mathbf{b}$
  Factor $\mathbf{A} = \mathbf{LU}$ in a lower precision
  while $\|\mathbf{r}\|$ too large do
    $\mathbf{d} = \mathbf{U}^{-1}\mathbf{L}^{-1}\mathbf{r}$
    $\mathbf{x} \leftarrow \mathbf{x} + \mathbf{d}$
    $\mathbf{r} = \mathbf{b} - \mathbf{A}\mathbf{x}$
  end while
```

Algorithm **IR-V0** leaves out many implementation details. Some recent papers [1,6,7,9] have made the algorithmic details explicit. The purpose of this paper is to build upon that work by explicitly including the interprecision transfers in the algorithmic description.

One consequence of this, which we discuss in § 2.2 and § 3.1 is a novel interpretation of the classic form of the method [22] where the residual is evaluated in an extended precision.

1.1. Notation.

We use the terminology from [1,9] and consider several precisions. We will use Julia-like notation for data types.

- The matrix **A** and right side **b** are stored in the **working precision** TW.
- **A** is factored in the **factorization precision** TF.
- The residual is computed in the **residual precision** TR.
- The triangular solves for $\mathbf{LUd} = \mathbf{r}$ are done in the **solver precision** TS.

If TH is a higher precision than TL, we will write $TL < TH$. fl_X will be the rounding operation to precision TX. We will let $\mathcal{F}_X$ be the floating point numbers with precision TX. We will assume that a low

*North Carolina State University, Department of Mathematics, Box 8205, Raleigh, NC 27695-8205, USA (Tim_Kelley@ncsu.edu). This work was partially supported by the Center for Exascale Monte-Carlo Neutron Transport (CEMeNT) a PSAAP-III project funded by Department of Energy grant number DE-NA003967.

precision number can be exactly represented in any higher precision. So

$$x \in \mathcal{F}_X \subset \mathcal{F}_Y \text{ if } TX \le TY. \tag{1.2}$$

We will let u_X denote the unit roundoff for precision TX and let I_A^B denote the interprecision transfer from precision TA to precision TB. Interprecision transfer is more than rounding and can, in some cases, include data allocation. If $TA < TB$ ($TA > TB$) then we will call the transfer I_A^B *upcasting* (*downcasting*).

Upcasting is simpler than downcasting because if $TA < TB$ and $x \in \mathcal{F}_A$, then (1.2) implies that

$$I_A^B x = x. \tag{1.3}$$

However, upcasting is not linear. In fact if $y \in \mathcal{F}_A$ there is no reason to expect that

$$I_A^B(xy) = I_A^B(x) I_A^B(y)$$

because the multiplication on the left is done in a lower precision than the one on the right. Downcasting is more subtle because not only is it nonlinear but also if $TA > TB$ and $x \in \mathcal{F}_A$, then (1.3) holds only if $x \in \mathcal{F}_B \subset \mathcal{F}_A$.

The nonlinearity of interprecision transfers should be made explicit in analysis, especially for the triangular solves (see § 2.3).

2. Interprecision transfers in IR.

Algorithm **IR-V0** includes many implicit interprecision transfers. We will make the assumption, which holds in IEEE [14] arithmetic, that when a binary operation $\circ$ is performed between floating point numbers with different precisions, then the lower precision number is promoted before the operation is performed.

So, if $TH > TL$, $u \in \mathcal{F}_H$, $v \in \mathcal{F}_L$, then

$$fl_H(u \circ v) = fl_H(u \circ (I_L^H v)) \text{ and } fl_H(v \circ u) = fl_H((I_L^H v) \circ u). \tag{2.1}$$

We will use (2.1) throughout this paper when we need to make the implicit interprecision transfers explicit.

2.1. The low precision factorization.

We will begin with the low precision factorization. The line in Algorithm **IR-V0** for this is

- Factor $\mathbf{A} = \mathbf{LU}$ in a lower precision.

However, $\mathbf{A}$ is stored in precision TW and the factorization is in precision TF. Hence one must make a copy of $\mathbf{A}$. So to make this interprecision transfer explicit we should express this as

- Make a low precision copy of $\mathbf{A}$. $\mathbf{A}_F = I_W^F \mathbf{A}$.
- Compute an LU factorization $\mathbf{LU}$ of $\mathbf{A}_F$, overwriting $\mathbf{A}_F$.

So in this way the storage costs of IR become clear and we can see the time/storage tradeoff. If, for example, $TW = Float64$ and $TF = Float32$, one must allocate storage for $\mathbf{A}_F$ in addition to the allocation for $\mathbf{A}$, which is needed to compute the residual. Hence IR requires a 50% increase in matrix storage.

2.2. The residual precision.

For the remainder of this paper we use the ℓ^∞ norm, so $\| \cdot \| = \| \cdot \|_\infty$.

The algorithm **IR-V0** does not make it clear how the residual precision affects the iteration. The description in [9] carefully explains what one must do.

- Store $\mathbf{x}$ in precision TR.
- Solve the triangular system in precision $TS = TR$.
- Store $\mathbf{d}$ in precision TR.

The most important consequence of this and (2.1) is that the residual is

$$\mathbf{r} = I_W^R(\mathbf{b} - \mathbf{A}\mathbf{x}) = I_W^R \mathbf{b} - (I_W^R \mathbf{A})\mathbf{x}.$$

While $I_W^R \mathbf{A} = \mathbf{A}$ by our assumptions that $TR \ge TW$, the residual is computed in precision TR and is therefore the residual of a promoted problem and the iteration is approximating the solution of that promoted problem

$$\mathbf{x}_P^* = (I_W^R \mathbf{A})^{-1} \mathbf{b} = (I_W^R \mathbf{A})^{-1} I_W^R \mathbf{b}, \tag{2.2}$$

which is posed in precision TR. We make a distinction between $\mathbf{x}_P^*$ and $\mathbf{x}^* = \mathbf{A}^{-1}\mathbf{b}$ only in those cases, such as (2.2) where we are talking about the computed solution in the residual precision. In exact arithmetic, of course, $\mathbf{x}_P^* = \mathbf{x}^*$.

All of these interprecision transfers are implicit and need not be done within the iteration. For example, when one computes $\mathbf{r}$ in precision TR, (2.1) implies that the matrix-vector product $\mathbf{Ax}$ automatically promotes the elements of $\mathbf{A}$ to precision TR because $\mathbf{x}$ is stored in precision TR.

We can use the fact that convergence is to the solution of the promoted problem to get a simple error estimate for the classic special case [22]. Here $TR = TS > TW \geq TF$. If the iteration terminates with

$$\|\mathbf{r}\|/\|\mathbf{b}\| \leq \tau$$

then the standard estimates [10] imply that

$$\frac{\|\mathbf{x} - \mathbf{x}^*\|}{\|\mathbf{x}^*\|} = \frac{\|\mathbf{x} - \mathbf{x}_P^*\|}{\|\mathbf{x}_P^*\|} \leq \frac{\kappa(I_W^R \mathbf{A})\|\mathbf{r}\|}{\|I_W^R(\mathbf{b})\|} = \frac{\kappa(\mathbf{A})\|\mathbf{r}\|}{\|\mathbf{b}\|} \leq \tau\kappa(\mathbf{A}). \tag{2.3}$$

In (2.3) we use the fact that $\|I_W^R(\mathbf{b})\| = \|\mathbf{b}\|$ in the ℓ^∞ norm and use the exact value of $\kappa(\mathbf{A})$, which is the same as $\kappa(I_W^R\mathbf{A})$ by (1.2). The estimate (2.3) is also true if $TR = TW$, but then there is no need to consider a promoted problem.

2.3. The triangular solve. The choice of TS affects the number of interprecision transfers and the storage cost in the triangular solve. The line in Algorithm **IR-V0** for this is

- $\mathbf{d} = \mathbf{U}^{-1}\mathbf{L}^{-1}\mathbf{r}$

If $TS = TR$, then the LAPACK's default behavior is to do the interprecision transfers as needed using (2.1). The subtle consequence of this is that

$$(\mathbf{LU})^{-1}\mathbf{r} = ((I_F^R\mathbf{L})(I_F^R\mathbf{U}))^{-1}\mathbf{r}.$$

Hence, one is implicitly doing the triangular solves with the factors promoted to the residual precision. We will refer to this approach as **on-the-fly** interprecision transfers.

Combining the results with § 2.2 we can expose all the interprecision transfers in the transition from a current iteration $\mathbf{x}_c$ to a new one $\mathbf{x}_+$. In the case $TS = TR$ we have a linear stationary iterative method. The computation is done entirely in TR.

$$\mathbf{x}_+ = \mathbf{x}_c + \mathbf{d} = \mathbf{x}_c + (\mathbf{LU})^{-1}I_W^R(\mathbf{b} - \mathbf{A}\mathbf{x}_c) = \mathbf{M}\mathbf{x}_c + (\mathbf{LU})^{-1}I_W^R\mathbf{b}, \tag{2.4}$$

where the iteration matrix is

$$\mathbf{M} = \mathbf{I} - ((I_F^R\mathbf{L})(I_F^R\mathbf{U}))^{-1}(I_W^R\mathbf{A}). \tag{2.5}$$

The residual update is

$$\mathbf{r}_+ = \mathbf{M}_r\mathbf{r}_c \tag{2.6}$$

where

$$\mathbf{M}_r = \mathbf{I} - (I_W^R\mathbf{A})((I_F^R\mathbf{L})(I_F^R\mathbf{U}))^{-1}. \tag{2.7}$$

One must remember that if $TR > TW$ and $\mathbf{x} \in \mathcal{F}_W^N$ then $(I_W^R\mathbf{A})\mathbf{x} \neq \mathbf{Ax}$ because the matrices are in different precisions and matrix-vector products produce different results.

All the interprecision transfers in (2.5) and (2.7) are implicit and the promoted matrices are not actually stored. However, the promotions matter because they can help avoid underflows and overflows and influence the limit of the iteration.

If $TS = TF < TW$, then the interprecision transfer is done before the triangular solves and the number of interprecision transfers is N rather than N^2. We will refer to this as interprecision transfer **in-place** to distinguish it from on-the-fly. For in-place interprecision transfers we copy $\mathbf{r}$ from the residual precision

TR to the factorization precision before the solve and then upcast the output of the solve back to the residual precision. So, one must store the low precision copy of $\mathbf{r}$. In this case one should scale $\mathbf{r}$ before the downcasting transfer I_R^F [13]. One reason for this is that the absolute size of $\mathbf{r}$ could be very small, as would be the case in the terminal phase of IR, and one could underflow before the iteration is complete. So the iteration in this case is

$$\mathbf{x}_+ = \mathbf{x}_c + \mathbf{d} = \mathbf{x}_c + \|\mathbf{r}\| I_F^R \left[(\mathbf{LU})^{-1} \frac{I_R^F \mathbf{r}}{\|\mathbf{r}\|} \right]. \tag{2.8}$$

This is not a stationary linear iterative method because the map $\mathbf{r} \to \frac{I_R^F \mathbf{r}}{\|\mathbf{r}\|}$ is nonlinear.

Even though downcasting $\mathbf{r}$ reduces the interprecision transfer cost, one should do interprecision transfers on-the-fly if TF is half precision, if $TR > TW$, or if one is using the low-precision factorization as a preconditioner [1, 7].

3. Explicit Interprecision Transfers.

We apply the results from § 2 Algorithm **IR-V0** and obtain Algorithm **IR-V1**, where all the interprecision transfers are explicit.

```
IR-V1($\mathbf{A}, \mathbf{b}, TF, TW, TR$)
  $\mathbf{x} = 0 \in \mathcal{F}_R^N$
  $\mathbf{r} = I_W^R(\mathbf{b})$
  $\mathbf{A}_F = I_W^F(\mathbf{A})$.
  Factor $\mathbf{A}_F = \mathbf{LU}$ in precision TF
  while $\|\mathbf{r}\|$ too large do
    $\mathbf{d} = ((I_F^R \mathbf{L})(I_F^R \mathbf{U}))^{-1} \mathbf{r}$ in precision TR
    $\mathbf{x} \leftarrow \mathbf{x} + \mathbf{d}$
    $\mathbf{r} = (I_W^R \mathbf{b}) - (I_W^R \mathbf{A})\mathbf{x}$
  end while
```

In the remainder of this section we look at some consequences of this formulation of IR.

3.1. The case $TS = TR > TW$.

In this section we will assume that the triangular solves are done in the residual precision (TS = TR). In this case, one can see from Algorithm **IR-V1** that no computations are done in the working precision at all. The working precision is only used to store $\mathbf{A}$ and $\mathbf{b}$, but residual computations are done in the residual precision with promotion on-the-fly. We state this observation as a theorem:

THEOREM 3.1. *If $TS = TR > TW$, then the three precision algorithm*

$$\mathbf{IR\text{-}V1}(\mathbf{A}, \mathbf{b}, TF, TW, TR)$$

produces the same computed results as the two precision algorithm

$$\mathbf{IR\text{-}V1}(I_W^R \mathbf{A}, I_W^R \mathbf{b}, TF, TR, TR).$$

The theorem makes it clear that the iteration is reducing the residual of the promoted problem. So we can apply the classical ideas for **IR-V0** [11, 12] and understand the case $TR > TW$ in that way. For example, if $\mathbf{A}$ is not highly ill-conditioned, the LU factorization is stable, and the norm of iteration matrix for IR $\|\mathbf{M}\| < 1$, then we can use equation (4.9) from [12] to obtain

$$\|\mathbf{r}_+\| \le G\|\mathbf{r}_c\| + g, \text{ where } g = O(u_R[\|I_W^R \mathbf{A}\| \|\mathbf{x}_P^*\| + \|I_W^R \mathbf{b}\|]). \tag{3.1}$$

where $G < 1$. Hence we will be able to reduce $\|\mathbf{r}\|$ until the iteration saturates with $\|\mathbf{r}\| \approx g$.

Since one has no *a priori* knowledge of g, one must manage the iteration in a way to detect stagnation. The recommendation from [12] is to terminate the iteration when

1. $\|\mathbf{r}\| \le C u_R(\|\mathbf{A}\|\|\mathbf{x}\| + \|\mathbf{b}\|)$,
2. $\|\mathbf{r}_+\| \ge \alpha \|\mathbf{r}_c\|$, or

3. too many iterations have been performed.

The first item in the list is successful convergence where we approximate the norms of the promoted objects $I_W^R \mathbf{A}$ and $I_W^R \mathbf{b}$ with the ones in the working precision which have stored. The two failure modes are insufficient decrease in the residual and slow convergence in the terminal phase. The recommendation in [12] was to set $C = 1$, $\alpha = .5$, and to limit IR to five iterations.

One problem with this approach is that the computation of $\|\mathbf{A}\|$ can be as much as two or three IR iterations. Our Julia code [18] has the options of terminating on small normwise backward as above or terminating on small relative residuals. We refer the reader to the documentation [18] for the details. For this work, we terminate on small relative residuals to avoid the costly computation of $\|\mathbf{A}\|$ We used $\alpha = .9$ and did not put a limit on the iterations. The reason for these choices is to give the IR iteration a better chance to terminate successfully.

So, we terminate the iteration when

1. $\|\mathbf{r}\| \leq C u_R \|\mathbf{b}\|$, or
2. $\|\mathbf{r}_+\| \geq \alpha \|\mathbf{r}_c\|$.

So, if we couple the termination strategy with (2.3) we see that if the iteration terminates successfully then

$$\frac{\|\mathbf{x} - \mathbf{x}_P^*\|}{\|\mathbf{x}_P^*\|} \leq u_R \frac{C\kappa(\mathbf{A})\|\mathbf{b}\|}{\|I_W^R(\mathbf{b})\|}.$$

When $TR = TS > TW$ one can also attempt to estimate the convergence rate G from (3.1) and then estimate the error $\|\mathbf{x} - \mathbf{x}_P^*\|$. This is a common strategy in the nonlinear solver literature, especially for stiff initial value problems [3–5, 15, 20]. The idea is that as the iteration progresses

$$G \approx \sigma = \|\mathbf{x}_{n+1} - \mathbf{x}_n\| / \|\mathbf{x}_n - \mathbf{x}_{n-1}\|$$

is a very good estimate if G is small enough. In that case

$$\|\mathbf{x}_n - \mathbf{x}_P^*\| \leq \frac{\|\mathbf{x}_{n+1} - \mathbf{x}_n\|}{1 - \sigma}$$

and one can terminate the iteration when one predicts that

$$\|\mathbf{x}_{n+1} - \mathbf{x}_P^*\| \leq \frac{\|\mathbf{x}_{n+1} - \mathbf{x}_n\|\sigma}{1 - \sigma}$$

is sufficiently small. The algorithm in [9] does this and terminates when the predicted error is less than u_W.

3.2. Cost of Interprecision Transfers. One case where setting $TS = TF$ may be useful is if $TS = Float32$ and $TW = TR = Float64$. In this case, unlike $TF = Float16$, tools such as LAPACK and BLAS have been compiled to work efficiently. The cases of interest in this section are medium-sized problems where the $O(N^3)$ cost of the LU factorization is a few times more than the cost of the triangular solves, but not orders of magnitude more. In these cases the $O(N^2)$ cost of interprecision transfers on the fly is noticeable and could make a difference in cases where many triangular solves are done for each factorization. One example is for nonlinear solvers [15, 17] where the factorization of the Jacobian can be reused for many Newton iterations (or even time steps when solving stiff initial value problems [20, 21]).

We illustrate this with some cpu timings. The computations in this section were done on an Apple Macintosh Mini Pro with a M2 processor and eight performance cores. We used OpenBlas, which satisfies (2.1), rather than the AppleAccelerate Framework, which does not. We used Julia [2] v1.11.0-beta2 with the author's **MultiPrecisionArrays.jl** [18, 19] Julia package. We made this choice because Julia v1.11.0 has faster matrix-vector products than the current version v1.10.4.

We used the Julia package **BenchmarkTools.jl** [8] to get the timings we report in Table 3.1. This is the standard way to obtain timings in Julia. BenchmarkTools repeats computations and can obtain accurate results even if the compute time per run is very small.

We have put the Julia codes that generate Table 3.1 from § 3.2.1 in a GitHub repository
https://github.com/ctkelley/IR_Precision_Transfers

3.2.1. Integral Equation Example. We will use a concrete example rather than generating random problems. For a given dimension N let $\mathbf{G}$ be the matrix corresponding to the composite trapezoid rule discretization of the Greens operator $\mathcal{G}$ for $-d^2/dx^2$ on $[0,1]$

$$\mathcal{G}u(x) = \int_0^1 g(x,y)u(y)\,dy \text{ where } g(x,y) = \begin{cases} y(1-x); & x > y \\ x(1-y); & x \le y \end{cases}$$

The eigenvalues of $\mathcal{G}$ are $1/(n^2\pi^2)$ for $n = 1, 2, \dots$.

We use $\mathbf{A} = \mathbf{I} - 800.0 * \mathbf{G}$ in this example. The conditioning of $\mathbf{A}$ is somewhat poor with an ℓ^∞ condition number of roughly $\kappa_\infty(\mathbf{A}) \approx 18,253$. for the dimensions we consider in this section.

We terminate the IR iteration with the residual condition from § 3.1 and tabulate the dimension, the time (LU) for copying $\mathbf{A}$ from TW to TF and performing the LU factorization in TF (column 2 of Table 3.1), the timings for the two variants of the triangular solves (OTF = on-the-fly: column 3, IP = in-place: column 4), and the times and iteration counts for the IR loop with the two variations of the triangular solves (columns 5–8).

Table 3.1
Cost of OTF Triangular Solve

N	LU	OTF	IP	OTF-IR	its	IP-IR	its
200	1.6e-04	1.5e-05	5.7e-06	4.5e-05	3	3.8e-05	3
400	4.9e-04	5.4e-05	1.9e-05	2.3e-04	4	2.3e-04	5
800	1.8e-03	2.4e-04	6.6e-05	9.9e-04	5	7.3e-04	5
1600	7.8e-03	1.4e-03	2.5e-04	2.8e-03	4	2.1e-03	4
3200	4.2e-02	9.2e-03	1.3e-03	2.0e-02	5	1.7e-02	5
6400	2.9e-01	3.6e-02	6.1e-03	7.2e-02	5	5.8e-02	5

Table 3.1 shows that the factorization time is between 6 and 10 times that of the on-the-fly triangular solve indicating that the triangular solves could be significant if the matrix-vector products were fast or one had to solve for many right hand sides. We saw that effect in the nonlinear examples in [16, 17] where the nonlinear residual could be evaluated in $O(NlogN)$ work. One can also see that the cpu time for the in-place triangular solves is 2–5 times less than for the on-the-fly version.

In the final four columns we see that, while the version with in-place triangular solves is somewhat faster, the difference is not compelling. This is no surprise because the matrix-vector product in the residual precision (double) takes $O(N^2)$ work and is therefore a significant part of the cost for each IR iteration. The number of iterations is the same in all but one case, with in-place triangular solves taking one more iteration in that case. That is consistent with the prediction in [7].

4. Conclusions.

We expose the interprecision transfers in iterative refinement and obtain new insights into this classical algorithm. In particular we show that the version in which the residual is evaluated in an extended precision is equivalent to solving a promoted problem and show how interprecision transfers affect the triangular solves.

5. Acknowledgments.

The author is very grateful to Ilse Ipsen for listening to him as he worked through the ideas for this paper.

REFERENCES

[1] P. Amestoy, A. Buttari, N. J. Higham, J.-Y. L'Excellent, T. Mary, and B. Vieublé, *Five-precision GMRES-based iterative refinement*, SIAM Journal on Matrix Analysis and Applications, 45 (2024), pp. 529–552, https://doi.org/10.1137/23M1549079.

[2] J. Bezanson, A. Edelman, S. Karpinski, and V. B. Shah, *Julia: A fresh approach to numerical computing*, SIAM Review, 59 (2017), pp. 65–98.

[3] K. E. Brenan, S. L. Campbell, and L. R. Petzold, *The Numerical Solution of Initial Value Problems in Differential-Algebraic Equations*, no. 14 in Classics in Applied Mathematics, SIAM, Philadelphia, 1996.

[4] P. N. Brown, G. D. Byrne, and A. C. Hindmarsh, *VODE: A variable coefficient ode solver*, SIAM J. Sci. Statist. Comput., 10 (1989), pp. 1038–1051.

[5] P. N. Brown, A. C. Hindmarsh, and L. R. Petzold, *Using Krylov methods in the solution of large-scale differential-algebraic systems*, SIAM J. Sci. Comput., 15 (1994), pp. 1467–1488.

[6] E. Carson and N. J. Higham, *A new analysis of iterative refinement and its application of accurate solution of ill-conditioned sparse linear systems*, SIAM Journal on Scientific Computing, 39 (2017), pp. A2834–A2856, https://doi.org/10.1137/17M112291.

[7] E. Carson and N. J. Higham, *Accelerating the solution of linear systems by iterative refinement in three precisions*, SIAM Journal on Scientific Computing, 40 (2018), pp. A817–A847, https://doi.org/10.1137/17M1140819.

[8] J. Chen and J. Revels, *Robust benchmarking in noisy environments*, 2016, https://arxiv.org/abs/1608.04295.

[9] J. Demmel, Y. Hida, W. Kahan, X. S. Li, S. Mukherjee, and E. J. Riedy, *Error bounds from extra-precise iterative refinement*, ACM Trans. Math. Soft., (2006), pp. 325–351.

[10] J. W. Demmel, *Applied Numerical Linear Algebra*, SIAM, Philadelphia, 1997.

[11] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, 1996, http://www.ma.man.ac.uk/~higham/asna.html.

[12] N. J. Higham, *Iterative refinement for linear systems and LAPACK*, IMA J. Numer. Anal., 17 (1997), pp. 495–509.

[13] N. J. Higham, S. Pranesh, and M. Zounon, *Squeezing a matrix into half precision, with an application to solving linear systems*, SIAM J. Sci. Comp., 41 (2019), pp. A2536–A2551.

[14] IEEE Computer Society, *IEEE standard for floating-point arithmetic, IEEE Std 754–2019*, July 2019.

[15] C. T. Kelley, *Iterative Methods for Linear and Nonlinear Equations*, no. 16 in Frontiers in Applied Mathematics, SIAM, Philadelphia, 1995.

[16] C. T. Kelley, *Newton's method in mixed precision*, SIAM Review, 64 (2022), pp. 191–211, https://doi.org/10.1137/20M1342902.

[17] C. T. Kelley, *Solving Nonlinear Equations with Iterative Methods: Solvers and Examples in Julia*, no. 20 in Fundamentals of Algorithms, SIAM, Philadelphia, 2022.

[18] C. T. Kelley, *MultiPrecisionArrays.jl*, 2023, https://doi.org/10.5281/zenodo.7521427, https://github.com/ctkelley/MultiPrecisionArrays.jl. Julia Package.

[19] C. T. Kelley, *Using MultiPrecisonArrays.jl: Iterative refinement in Julia*, 2024, https://arxiv.org/abs/2311.14616.

[20] L. R. Petzold, *A description of DASSL: a differential/algebraic system solver*, in Scientific Computing, R. S. Stepleman et al., ed., North Holland, Amsterdam, 1983, pp. 65–68.

[21] L. F. Shampine, *Numerical Solution of Ordinary Differential Equations*, Chapman and Hall, New York, 1994.

[22] J. H. Wilkinson, *Progress report on the automatic computing engine*, Tech. Report MA/17/1024, Mathematics Division, Department of Scientific and Industrial Research, National Physical Laboratory, Teddington, UK, 1948, http://www.alanturing.net/turing_archive/archive/l/l10/l10.php.