Matrices with Tunable Infinity-Norm Condition Number and No Need for Pivoting in LU Factorization

Fasi, Massimiliano and Higham, Nicholas J.

2020

MIMS EPrint: **2020.17**

Manchester Institute for Mathematical Sciences
School of Mathematics

The University of Manchester

Reports available from: http://eprints.maths.manchester.ac.uk/
And by contacting: The MIMS Secretary
School of Mathematics
The University of Manchester
Manchester, M13 9PL, UK

ISSN 1749-9097

MATRICES WITH TUNABLE INFINITY-NORM CONDITION NUMBER AND NO NEED FOR PIVOTING IN LU FACTORIZATION*

MASSIMILIANO FASI† AND NICHOLAS J. HIGHAM‡

Abstract. We propose a two-parameter family of nonsymmetric dense $n \times n$ matrices $A(\alpha, \beta)$ for which LU factorization without pivoting is numerically stable, and we show how to choose α and β to achieve any value of the ∞-norm condition number. The matrix $A(\alpha, \beta)$ can be formed from a simple formula in $O(n^2)$ flops. The matrix is suitable for use in the HPL-AI Mixed-Precision Benchmark, which requires an extreme scale test matrix (dimension $n > 10^7$) that has a controlled condition number and can be safely used in LU factorization without pivoting. It is also of interest as a general-purpose test matrix.

Key words. test matrix, ∞-norm condition number, HPL-AI Mixed-Precision Benchmark, low-precision arithmetic, LU factorization

AMS subject classifications. 65F35, 65Y05

1. Introduction.

A wide variety of test matrices has been developed and made available in software since the early days of digital computing, for example in [5], [11], [15], [36]. A matrix property of particular interest is the condition number. Ideally, we would like the condition number to be a parameter that can be freely chosen, along with the matrix dimension, but this is rarely the case. An exception is the "randsvd" matrix, constructed as the product $U\Sigma V^T$, where $U \in \mathbb{R}^{n\times n}$ and $V \in \mathbb{R}^{n\times n}$ are random orthogonal matrices from the Haar distribution and $\Sigma = \mathrm{diag}(\sigma_i)$ with $\sigma_1 \geq \cdots \geq \sigma_n > 0$. These matrices have 2-norm condition number $\kappa_2(A) = \sigma_1/\sigma_n$ and can be generated, for example, using the LAPACK test matrix generation suite [7] or the MATLAB `gallery('randsvd',...)` function. However, they require $O(n^3)$ floating-point operations (flops) to generate, which can be prohibitively expensive if n is large. By making alternative choices of U and V and possibly restricting the choice of $\sigma_2, \ldots, \sigma_{n-1}$, however, the cost of this technique can be reduced to $O(n^2)$ flops [12].

In this work we propose a new class of $n \times n$ test matrices that satisfy the following three requirements.

- R1. They can be constructed in $O(n^2)$ flops.
- R2. Their ∞-norm condition number is a parameter.
- R3. Their growth factor for LU factorization without pivoting is of order 1, that is, Gaussian elimination without pivoting is numerically stable for them.

The third requirement, which is not satisfied by the matrices of [12], is motivated by the HPL-AI benchmark, as we explain in section 2. Matrices that meet these three conditions are worth studying in their own right, as not many classes of dense matrices that satisfy R2 or R3 are known. The main families of dense matrices for which pivoting is known not to be required are symmetric positive definite matrices, totally nonnegative matrices, and matrices that are diagonally dominant by rows or columns [16, Chap. 9].

---

*Version of December 7, 2020.

Funding: This work was supported Engineering and Physical Sciences Research Council grant EP/P020720/1 and the Royal Society.

†School of Science and Technology, Örebro University 701 82, Örebro, Sweden (massimiliano.fasi@oru.se).

‡Department of Mathematics, The University of Manchester, Manchester, M13 9PL, UK (nick.higham@manchester.ac.uk).

In this work we propose a two-parameter family of matrices $A(\alpha, \beta)$ for which LU factorization without pivoting is numerically stable and for which the parameters α and β that yield a matrix with specified ∞-norm condition number $\kappa_\infty(A(\alpha, \beta))$ can be easily computed.

In section 2 we describe the HPL-AI benchmark and why it motivates our work. In section 2.1 we focus on the test matrices that have been proposed for the benchmark, and consider whether diagonally dominant matrices can satisfy our requirements. In section 3 we define the family $A(\alpha, \beta)$, explain how the entries of $A(\alpha, \beta)$ can be cheaply and accurately computed, and show that LU factorization without pivoting is stable for these matrices. In section 4 we derive explicit formulae for the entries of $A(\alpha, \beta)^{-1}$. In section 5 we derive an explicit expression for $\kappa_\infty(A(\alpha, \beta))$ that can be evaluated in a constant number of flops, and we explain how to determine α and β to achieve a given value of $\kappa_\infty(A(\alpha, \beta))$. In section 6 we discuss the practicalities of constructing the matrices $A(\alpha, \beta)$ and confirm experimentally the numerical stability of LU factorization without pivoting when applied to them. In section 7 we explain how $A(\alpha, \beta)$ can be modified to avoid the possibility of a user of the HPL-AI benchmark gaining an unfair advantage. Concluding remarks are given in section 8.

2. The HPL-AI Mixed-Precision Benchmark. The supercomputers in the TOP500 list[1] are ranked according to their performance on HPL[2] [10], [30], a portable implementation of the High-Performance Computing Linpack Benchmark for distributed-memory computers. The software measures the rate of execution, expressed in flops per second (Flop/s), that a computer achieves when solving a large dense linear system with random coefficients in IEEE binary64 arithmetic [20].

The workload of HPL is compute-bound [25], in the sense that the time needed to run the benchmark is mostly determined by the clock speed of the CPU. As the gap between the clock speeds of CPUs and RAM widens, this metric is becoming less relevant, as memory accesses now dominate the performance of HPC applications. In order to take this aspect into account and obtain a more complete view of the capabilities of a large computational facility, several benchmarking suites that provide an alternative to HPL have been proposed in recent years

The first solution to be introduced was the HPCChallenge (HPCC) Benchmark,[3] a collection of six problems meant to complement HPL with tests exhibiting more challenging memory access patterns [24]. HPCC was succeeded by the High Performance Conjugate Gradients (HPCG) Benchmark[4] [8], [9], a software package developed to test computational and data access patterns that are common in more recent scalable applications.

The HPL, HPCC, and HPCG benchmarks are all typically run in binary64 arithmetic, which is the precision level required by most mathematical codes underlying scientific simulations in traditional HPC applications. This is not representative, however, of typical computations in the emerging field of artificial intelligence, where 32 or fewer bits of precision are typically considered adequate. The HPL-AI Mixed-Precision Benchmark[5] is an attempt to consider both workloads at once, and can be regarded as the first step towards a benchmark suite that takes into account the availability of hardware accelerators for low-precision computation, such as the sys-

[1] http://www.top500.org
[2] https://www.netlib.org/benchmark/hpl/
[3] https://icl.utk.edu/hpcc/
[4] https://icl.utk.edu/hpcg/
[5] https://icl.bitbucket.io/hpl-ai/

Algorithm 2.1: Reference implementation of the HPL-AI benchmark.

Input: Matrix order n, tolerance parameter μ.
Output: "success"/"failure" depending on residual of computed solution.
Constants: Binary64 unit roundoff $u = 2^{-53}$.

1 Generate $[A, b] \in \mathbb{R}^{n\times(n+1)}$ in binary64.
2 Compute $A \approx LU$ by LU factorization without pivoting in binary32.
3 Compute $x_0 = U^{-1}(L^{-1}b)$ in binary32.
4 Convert L, U, and x_0 to binary64.
5 Solve $Ax = b$ in binary64 using preconditioned GMRES without restart with x_0 as starting guess and L and U as preconditioners.
6 **if** $\|A\widehat{x} - b\|_\infty < \mu nu(\|A\|_\infty\|\widehat{x}\|_\infty + \|b\|_\infty)$ **then**
7 | **return** success
8 **else**
9 | **return** failure

tolic arrays that equip the Google TPU v2 and v3 or the tensor cores featured by the NVIDIA V100, T4, and A100 GPUs.

The main computational steps in the reference implementation of the HPL-AI benchmark[6] are summarized in Algorithm 2.1. A $n \times n$ test matrix A and a right-hand side vector b are generated in binary64 arithmetic. The LU factorization without pivoting of A is computed in binary32 arithmetic and the linear system $Ax = b$ is solved in binary32 arithmetic using forward and back substitution. This binary32 approximation to the solution is then converted to binary64 and refined using the generalized minimal residual method (GMRES) without restart. The solver is preconditioned with the binary32 LU factors of A (converted to binary64). The benchmark is declared successfully completed if the relative backward error satisfies

$$\frac{\|A\widehat{x} - b\|_\infty}{\|A\|_\infty\|\widehat{x}\|_\infty + \|b\|_\infty} < \mu nu, \tag{2.1}$$

where μ is a threshold parameter, which for the reference implementation of HPL-AI is set to 16, and $u = 2^{-53}$ is the unit roundoff for binary64. Although for portability reasons the reference implementation uses binary32 arithmetic as the lower-precision arithmetic, binary16 or bfloat16 [21] are expected to be used in practice.

The flop rate of Algorithm 2.1 is computed as f_h/t_g, where the global time to solution t_g is the number of seconds taken to execute lines 2–5, and $f_h = 2n^3/3+3n^2/2$ is the number of low precision floating-point operations required to compute the LU factorization on line 2 and perform the two triangular solves on line 3.

The matrix A must satisfy several requirements. First, it must be cheap to form and one for which LU factorization without pivoting is numerically stable. Second, for the algorithm to work at all the condition number $\kappa_\infty(A)$ must not be too large. No bounds on $\kappa_\infty(A)$ are known that guarantee that the reference implementation will work, but for the related GMRES-IR algorithm $\kappa_\infty(A)$ must be substantially less than u^{-1} when it is used with two precisions [4], [17]. These requirements will hold if properties R1–R3 hold and a suitable value of $\kappa_\infty(A)$ is chosen.

A further requirement is that GMRES does not converge quickly when applied to

[6] https://bitbucket.org/icl/hpl-ai/

A, because if it does then it does not need the LU factors as preconditioners and the factorization on line 2 of Algorithm 2.1 can be omitted altogether.

In the November 2020 edition of the TOP500 rankings of the world's fastest supercomputers, the Fugaku computer obtained first place in the HPL-AI benchmark with a rate of 2.0 EFlop/s,[7] obtained for a matrix of size 16,957,440; it had previously achieved 1.4 EFlop/s in the June 2020 list [22]. Clearly, then, any matrix used for the benchmark must be capable of being generated for a dimension exceeding 10^7.

2.1. Matrices proposed for the benchmark. Two classes of matrices have been proposed as test problems for the HPL-AI benchmark.

In order to satisfy requirements R1–R3 it is natural to consider diagonally dominant matrices. The matrix $A \in \mathbb{R}^{n\times n}$ is diagonally dominant by rows if

$$\omega_i := |a_{ii}| - \sum_{j\neq i} |a_{ij}| \geq 0, \quad i = 1\colon n, \tag{2.2}$$

and diagonally dominant by columns if A^T is diagonally dominant by rows. The growth factor for LU factorization without pivoting on matrices diagonally dominant by rows or columns is bounded by 2 [16, Thm. 9.9], [35]. The reference implementation of version 0.1 of the HPL-AI benchmark uses a row diagonally dominant matrix A constructed as

$$a_{ij} \sim \mathrm{unif}(-0.5, 0.5), \quad i \neq j, \qquad a_{ii} = \sum_{j\neq i} |a_{ij}|, \tag{2.3}$$

where $\mathrm{unif}(x, y)$ denotes the uniform distribution over the open real interval (x, y). Gaussian elimination without pivoting will be numerically stable for this matrix.

As regards the size of $\kappa_\infty(A)$, it is known [1], [16, Prob. 8.7], [34] that (2.2) implies

$$\|A^{-1}\|_\infty \leq \frac{1}{\min_i \omega_i}.$$

The matrix in (2.3) used by the reference implementation has $\omega_i \equiv 0$, so this bound does not provide any information. In fact, numerical experiments show the matrix to be extremely well conditioned, with $\kappa_\infty(A) \approx 4$ for large n.

Moler [26] suggests using the nonsymmetric matrix $A(\mu)$ whose (i, j) element is $(i-\mu j+1/2)^{-1}$, studies experimentally its conditioning, and investigates the numerical stability of its LU factorization without pivoting. Whether this matrix can satisfy requirements R2 and R3 is unclear.

3. A two-parameter family of matrices. Let us consider the parametric upper triangular matrix $T(\theta) \in \mathbb{R}^{n\times n}$ defined by

$$(T(\theta))_{ij} = \begin{cases} 0, & i > j, \\ 1, & i = j, \\ -\theta, & i < j, \end{cases} \tag{3.1}$$

where $\theta \geq 0$ throughout this paper. This is a scalar multiple of a matrix proposed by Ostrowski [29]. The inverse of the matrix in (3.1) is known explicitly:

$$\left(T(\theta)^{-1}\right)_{ij} = \begin{cases} 0, & i > j, \\ 1, & i = j, \\ \theta(1+\theta)^{j-i-1}, & i < j. \end{cases} \tag{3.2}$$

[7] https://www.top500.org/news/top500-expands-exaflops-capacity-amidst-low-turnover/

We define the matrix

$$A(\alpha,\beta) = LU \in \mathbb{R}^{n\times n}, \quad L = L(\alpha) = T(\alpha)^T, \quad U = U(\beta) = T(\beta), \tag{3.3a}$$

$$0 \le \alpha \le 1, \quad \beta \ge 0. \tag{3.3b}$$

For $n = 4$, for example,

$$A(\alpha,\beta) = \begin{bmatrix} 1 & -\beta & -\beta & -\beta \\ -\alpha & 1+\alpha\beta & -\beta+\alpha\beta & -\beta+\alpha\beta \\ -\alpha & -\alpha+\alpha\beta & 1+2\alpha\beta & -\beta+2\alpha\beta \\ -\alpha & -\alpha+\alpha\beta & -\alpha+2\alpha\beta & 1+3\alpha\beta \end{bmatrix}. \tag{3.4}$$

Since the subdiagonal elements of L are of modulus at most 1, LU factorization with partial pivoting does not require interchanges when applied to $A = A(\alpha,\beta)$ in exact arithmetic. Note that A is not, however, diagonally dominant in general. For $\alpha = \beta$ it is symmetric positive definite, but we are mainly interested in the nonsymmetric case.

Generating the matrix A by means of the matrix multiplication in (3.3) would require $2n^3/3$ flops, which is too costly for an extreme-scale setting. However, as (3.4) suggests should be possible, we can give an explicit expression for the elements of A by exploiting the structure of the two factors in (3.3a). As L and U are lower and upper triangular, respectively, we have that

$$a_{ij} = \sum_{k=1}^{\min(i,j)} \ell_{ik} u_{kj}.$$

For the elements along the diagonal of A, this gives

$$a_{ii} = \sum_{k=1}^{i} \ell_{ik} u_{kj} = \sum_{k=1}^{i-1} \alpha\beta + 1 \cdot 1 = 1 + (i-1)\alpha\beta,$$

for those above the diagonal we have

$$a_{ij} = \sum_{k=1}^{i} \ell_{ik} u_{kj} = \sum_{k=1}^{i-1} \alpha\beta - 1 \cdot \beta = -\beta + (i-1)\alpha\beta,$$

and for those in the strictly lower triangular part we obtain

$$a_{ij} = \sum_{k=1}^{j} \ell_{ik} u_{kj} = \sum_{k=1}^{j-1} \alpha\beta - \alpha \cdot 1 = -\alpha + (j-1)\alpha\beta.$$

Therefore

$$a_{ij} = \begin{cases} -\alpha + (j-1)\alpha\beta, & i > j, \\ 1 + (i-1)\alpha\beta, & i = j, \\ -\beta + (i-1)\alpha\beta, & i < j. \end{cases} \tag{3.5}$$

For later use, we note several relations. Recalling that $\|A\|_\infty = \max_i \sum_j |a_{ij}|$ and

$\|A\|_1 = \|A^T\|_\infty$, we have

$$\|L\|_\infty = 1 + (n-1)\alpha, \quad \|U\|_\infty = 1 + (n-1)\beta \le \|A\|_\infty, \tag{3.6}$$

$$\max_{i,j} |a_{ij}| \ge \max\big(\beta, 1 + (n-1)\alpha\beta\big), \tag{3.7}$$

$$\|A\|_1 \ge 1 + (n-1)\alpha. \tag{3.8}$$

We make two observations. Recall that $\alpha, \beta \ge 0$. First, we note that from (3.3) and the structure of T, we have

$$\kappa_\infty(A) \le \kappa_\infty\big(T(\alpha)^T\big)\kappa_\infty\big(T(\beta)\big) = \kappa_\infty\big(T(\alpha)\big)\kappa_\infty\big(T(\beta)\big).$$

From (3.2), the row of $T(\theta)^{-1}$ with the largest sum is the first, so

$$\|T(\theta)^{-1}\|_\infty = 1 + \sum_{i=0}^{n-2} \theta(1+\theta)^i = (1+\theta)^{n-1}.$$

Hence

$$\kappa_\infty(A) \le (1 + (n-1)\alpha)(\alpha+1)^{n-1}(1 + (n-1)\beta)(\beta+1)^{n-1}. \tag{3.9}$$

Next, we note that the matrix $A^{-1} = U^{-1}L^{-1}$ is nonnegative with $(1,n)$ element $(A^{-1})_{1n} = \beta(1+\beta)^{n-2}$, by (3.2). Hence, using (3.7),

$$\kappa_\infty(A) > \big(1 + (n-1)\alpha\beta\big)\beta(1+\beta)^{n-2}. \tag{3.10}$$

It is clear, then, that for large n we will need $\beta \ll 1$ in order for $\kappa_\infty(A)$ to be reasonably bounded. In section 5 we derive an exact expression for $\kappa_\infty(A)$.

3.1. The effect of rounding errors. Now we consider the effects of rounding errors in forming and factorizing $A(\alpha,\beta)$ in floating-point arithmetic.

First, we consider the errors in computing A from (3.5). Using the standard model of floating-point arithmetic [16, sect. 2.2] we find that the computed $\widehat{A}$ satisfies

$$|\widehat{a}_{ij} - a_{ij}| \le \begin{cases} \gamma_3(\alpha + (j-1)\alpha\beta) & i > j, \\ \gamma_3 a_{ij} & i = j, \\ \gamma_3(\beta + (i-1)\alpha\beta) & i < j, \end{cases}$$

where $\gamma_k = ku/(1-ku)$. By (3.7), $|\widehat{a}_{ij} - a_{ij}| \le 2\gamma_3 \max_{ij} |a_{ij}|$, so we certainly have

$$\|A - \widehat{A}\|_\infty \le 2\gamma_3 n \|A\|_\infty, \tag{3.11}$$

where the factor n is pessimistic. In fact, we also have $|A - \widehat{A}| \le c\gamma_3|A|$ as long as $|-\alpha + (j-1)\alpha\beta|$ and $|-\beta + (i-1)\alpha\beta|$ do not exceed $\alpha + (j-1)\alpha\beta$ and $\beta + (i-1)\alpha\beta$, respectively, by more than a factor c, for all $i \ne j$. This latter condition holds with a modest constant c as long as neither α nor β is too close to a member of the set $\{\, 1/k : k = 1, 2, \dots, n-1 \,\}$. We conclude that A is formed as accurately as we could expect.

We also need to show that LU factorization without pivoting is numerically stable for A. Hence we need to bound the elements of all the Schur complements that arise

during the factorization. From the structure of $T_n(\alpha,\beta)$, where the subscript denotes the dimension, we have

$$A(\alpha,\beta) = T(\alpha)^T T(\beta) = \begin{bmatrix} 1 & 0 \\ -\alpha e & T_{n-1}(\alpha)^T \end{bmatrix} \begin{bmatrix} 1 & -\beta e^T \\ 0 & T_{n-1}(\beta) \end{bmatrix} = \begin{bmatrix} 1 & -\beta e^T \\ -\alpha e & T_{n-1}(\alpha)^T T_{n-1}(\beta) + \alpha\beta e e^T \end{bmatrix},$$

where e is the vector of ones. Thus after one step of elimination we have the matrix

$$\begin{bmatrix} 1 & -\beta e^T \\ 0 & T_{n-1}(\alpha)^T T_{n-1}(\beta) \end{bmatrix},$$

in which $T_{n-1}(\alpha)T_{n-1}(\beta)$ is the leading principal submatrix of order $n-1$ of $A(\alpha,\beta)$. By induction it follows that all the elements of all the Schur complements are elements of $A(\alpha,\beta)$, and hence the growth factor is

$$\rho_n(A) = \frac{\max_{i,j,k}|a_{ij}^{(k)}|}{\max_{i,j}|a_{ij}|} = 1, \tag{3.12}$$

which is ideal.

The strongest overall backward error bound can be obtained by applying the backward error analysis for LU factorization [16, Thm. 9.3], which tells us that the computed LU factors $\widehat{L}$ and $\widehat{U}$ satisfy

$$\widehat{L}\widehat{U} = A + \Delta A, \quad |\Delta A| \le \gamma_n |\widehat{L}||\widehat{U}|. \tag{3.13}$$

Now, since $\widehat{\ell}_{ij} = \ell_{ij} + O(u)$ and $\widehat{u}_{ij} = u_{ij} + O(u)$, $|\widehat{L}||\widehat{U}| = |L||U| + O(u)$, and we have

$$|L||U| = (-L + 2I)(-U + 2I) = LU - 2(L+U) + 4I = A + G,$$

In section 5 we will take $\alpha \le \beta$, so we make this assumption now. Then, using (3.6) and (3.8), we obtain

$$\|G\|_\infty \le 2(\|L\|_\infty + \|U\|_\infty) + 4 \le 2(\|A\|_\infty + \|A\|_\infty) + 4\|A\|_\infty = 8\|A\|_\infty \quad (\alpha \le \beta).$$

Hence

$$\|\Delta A\|_\infty \le 9\gamma_n \|A\|_\infty + O(u) = 9nu\|A\|_\infty + O(u) \quad (\alpha \le \beta). \tag{3.14}$$

We conclude that the factorization is numerically stable.

The bound (3.14) is a worst-case bound and we note that in practice a more realistic bound is obtained by replacing the constant $9n$ by its square root, and even this is likely to be pessimistic [18]. This point is important because we are interested in extremely large n for the HPL-AI benchmark.

4. The inverse matrix. We now compute the elements of $B = A(\alpha,\beta)^{-1}$. Note that inverting (3.3a) yields $B = \widetilde{U}\widetilde{L}$, where $\widetilde{U} = U^{-1}$ and $\widetilde{L} = L^{-1}$, and we have

$$b_{ij} = \sum_{k=\max(i,j)}^{n} \widetilde{u}_{ik}\widetilde{\ell}_{kj}.$$

As above, this gives three distinct formulae, one for the elements along the diagonal,

$$\begin{aligned} b_{ii} &= \sum_{k=i}^{n} \widetilde{u}_{ik}\widetilde{\ell}_{kj} = 1 \cdot 1 + \sum_{k=i+1}^{n} \beta(1+\beta)^{k-i-1}\alpha(1+\alpha)^{k-i-1} \\ &= 1 + \alpha\beta \sum_{k=0}^{n-i-1} (1+\alpha)^k (1+\beta)^k, \end{aligned} \tag{4.1}$$

one for those in the upper triangular part,

$$\begin{aligned} b_{ij} &= \sum_{k=j}^{n} \widetilde{u}_{ik}\widetilde{\ell}_{kj} = \beta(1+\beta)^{j-i-1} \cdot 1 + \sum_{k=j+1}^{n} \beta(1+\beta)^{k-i-1}\alpha(1+\alpha)^{k-j-1} \\ &= \beta(1+\beta)^{j-i-1} + \alpha\beta \sum_{k=0}^{n-j-1} (1+\beta)^{k-i+j}(1+\alpha)^k, \end{aligned} \tag{4.2}$$

and one for those below the diagonal,

$$\begin{aligned} b_{ij} &= \sum_{k=i}^{n} \widetilde{u}_{ik}\widetilde{\ell}_{kj} = 1 \cdot \alpha(1+\alpha)^{i-j-1} + \sum_{k=i+1}^{n} \beta(1+\beta)^{k-i-1}\alpha(1+\alpha)^{k-j-1} \\ &= \alpha(1+\alpha)^{i-j-1} + \alpha\beta \sum_{k=0}^{n-i-1} (1+\beta)^k (1+\alpha)^{k+i-j}. \end{aligned} \tag{4.3}$$

By noting that (4.1), (4.2), and (4.3) contain truncated geometric series of ratio

$$r = (1+\alpha)(1+\beta)$$

we obtain the more computationally convenient formula

$$b_{ij} = \begin{cases} \alpha(1+\alpha)^{i-j-1} + \dfrac{\alpha\beta(1+\alpha)^{i-j}(1-r^{n-i})}{1-r}, & i > j, \\ 1 + \alpha\beta\dfrac{1-r^{n-i}}{1-r}, & i = j, \\ \beta(1+\beta)^{i-j-1} + \dfrac{\alpha\beta(1+\beta)^{j-i}(1-r^{n-j})}{1-r}, & i < j. \end{cases} \tag{4.4}$$

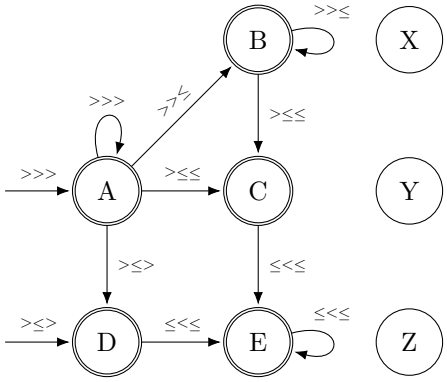
5. Generating a matrix with a prescribed condition number. We now discuss how the parameters α and β can be chosen so that $A := A(\alpha, \beta)$ has the desired condition number κ. This task can be achieved efficiently by combining an inexpensive technique for evaluating $\kappa_\infty(A) = \|A\|_\infty \|A^{-1}\|_\infty$ with a rootfinding algorithm for solving the scalar equation $\kappa_\infty(A(\alpha, \beta)) = \kappa$. In section 5.1 we derive explicit formulae for $\|A\|_\infty$ and $\|A^{-1}\|_\infty$ that can be evaluated in a constant number of operations, and in section 5.2 we discuss how these can be exploited to find a suitable value for α and β.

5.1. Computing the condition number. The sum of the absolute values of the ith row of A is, using (3.5),

$$\begin{aligned} \lambda_i &= \sum_{j=1}^{n} |a_{ij}| = \left(\sum_{j=1}^{i-1} |(j-1)\alpha\beta - \alpha| \right) + 1 + (i-1)\alpha\beta + \sum_{j=i+1}^{n} |(i-1)\alpha\beta - \beta| \\ &= \alpha \left(\sum_{j=1}^{i-1} |1 - (j-1)\beta| \right) + 1 + (i-1)\alpha\beta + (n-i)\beta|1 - (i-1)\alpha|. \end{aligned} \tag{5.1}$$

	$1-(i-1)\alpha$	$1-i\alpha$	$1-(i-1)\beta$
Ⓐ	>0	>0	>0
Ⓑ	>0	>0	≤ 0
Ⓒ	>0	≤ 0	≤ 0
Ⓓ	>0	≤ 0	>0
Ⓔ	≤ 0	<0	≤ 0
Ⓧ	≤ 0	>0	>0
Ⓨ	≤ 0	>0	≤ 0
Ⓩ	≤ 0	≤ 0	>0

(A) *States.*



(B) *Transitions.*

FIG. 5.1. *Left: sign of the argument of the absolute values in* (5.3). *Right: possible transitions among the states in the table on the left. Initial states are marked with an incoming arrow, final states have a double edge, and the states X, Y, and Z in the rightmost column are unreachable. A state transition occurs every time the value of i is incremented, and the next state is determined by the sign of the arguments of the absolute values in* (5.3). *The order of the three inequality signs is that of the columns in Figure* 5.1a.

We now explain how to compute the ∞-norm of A for

$$0 < \alpha \leq \beta.$$

In order to find the largest value of λ_i, we will look at the quantity

$$\Lambda_i := \lambda_{i+1} - \lambda_i, \qquad i = 1, \ldots, n-1, \tag{5.2}$$

which is the difference between the sum of the absolute values of two consecutive rows.

By subtracting the formula for λ_i from that for λ_{i+1}, we obtain

$$\Lambda_i = \alpha|1-(i-1)\beta| + \alpha\beta + (n-i-1)\beta|1-i\alpha| - (n-i)\beta|1-(i-1)\alpha|. \tag{5.3}$$

The value of Λ_i depends on the sign of the argument of the three absolute value operators in (5.3). In general this would give rise to eight possible combinations, but if $1-(i-1)\alpha \leq 0$ then $1-i\alpha < 0$, and $\alpha \leq \beta$ implies that also $1-(i-1)\beta \leq 0$. Therefore, only the five cases in Figure 5.1a are possible, as we now show. The automaton in Figure 5.1b shows the possible transitions, as i varies, from one state to the other. We remark that Ⓐ and Ⓓ are the only possible initial states ($i = 1$) since $\alpha \leq 1$ implies that $1-\alpha$ is nonnegative.

Our strategy is to keep track of the quantity

$$\eta_i := \max_{1 \leq k \leq i} \lambda_i$$

as we travel between states, and then use the fact that $\|A\|_\infty = \eta_n$. As long as we are in state Ⓐ we have that

$$\begin{aligned}\Lambda_i &= \alpha(1-(i-1)\beta) + \alpha\beta + (n-i-1)\beta(1-i\alpha) - (n-i)\beta(1-(i-1)\alpha) \\ &= \alpha - \beta + (2+i-n)\alpha\beta.\end{aligned}$$

If $i < n-1$ then $2+i-n \le 0$ and $\Lambda_i \le 0$, and thus $\lambda_{i+1} \le \lambda_i \le \cdots \le \lambda_1 = \eta_i$. The quantity Λ_{n-1} may be nonnegative, thus if Ⓐ is the final state we have that $\eta_n = \max(\lambda_1, \lambda_n)$.

Now we discuss the states that can be reached from Ⓐ. If we are in state Ⓑ, we have that

$$\begin{aligned}\Lambda_i &= \alpha((i-1)\beta - 1) + \alpha\beta + (n-i-1)\beta(1-i\alpha) - (n-i)\beta(1-(i-1)\alpha)\\ &= -\alpha - \beta + (3i-n)\alpha\beta.\end{aligned}$$

It is easy to see that in this state one has that $\eta_i = \max(\lambda_1, \lambda_{i+1})$. If $\Lambda_i \le 0$, then one comes from either Ⓐ, for which $\Lambda_{i-1} \le 0$, or from Ⓑ itself, in which case $\Lambda_{i-1} < \Lambda_i \le 0$, and thus $\eta_i = \lambda_1 = \max(\lambda_1, \lambda_{i+1})$. If, on the other hand, $\Lambda_i > 0$, then $\eta_i = \max(\lambda_1, \lambda_{i+1})$.

For Ⓒ, we have two distinct cases depending on the state we are coming from. When coming from Ⓐ, we have that $\eta_i = \eta_{i-1} = \lambda_1$ if $\Lambda_i \le 0$ and $\eta_i = \max(\eta_{i-1}, \lambda_{i+1})$ if $\Lambda_i > 0$. When coming from Ⓑ, we have that $\eta_i = \max(\lambda_1, \lambda_{i'}, \lambda_{i+1})$, where $i' = \min\left(\lfloor 1/\alpha \rfloor, n\right)$ is the largest row index for which one is in state Ⓑ. For state Ⓓ one can argue as for state Ⓒ that $\eta_i = \max(\lambda_1, \lambda_{i+1})$.

Finally, for state Ⓔ we have that

$$\begin{aligned}\Lambda_i &= \alpha((i-1)\beta - 1) + \alpha\beta + (n-i-1)\beta(i\alpha - 1) - (n-i)\beta((i-1)\alpha - 1)\\ &= -\alpha + \beta + (n-i)\alpha\beta > 0,\end{aligned}$$

since $n - i > 0$ for i between 1 and $n-1$. Therefore, if one is in state Ⓔ for row i, then for all subsequent rows k for k between $i+1$ and n one remains in state Ⓔ and has that $\eta_k = \max(\eta_{k-1}, \lambda_{k+1})$.

Combining the latter observation with the value of η_i from the states from which Ⓔ can be reached, we can conclude that

$$\|A\|_\infty = \max(\lambda_1, \lambda_{i'}, \lambda_n), \quad i' = \min\left(\lfloor 1/\alpha \rfloor, n\right). \tag{5.4}$$

Using (5.1), we find that

$$\lambda_1 = 1 + (n-1)\beta, \tag{5.5}$$

that

$$\begin{aligned}\lambda_{i'} &= \alpha \sum_{j=0}^{i'-2} |1 - j\beta| + 1 + (i'-1)\alpha\beta + (n-i')\beta(1-(i'-1)\alpha)\\ &= \alpha\left(\sum_{j=0}^{k-1}(1-j\beta) + \sum_{j=k}^{i'-2}(j\beta - 1)\right) + 1 + (n-i')\beta + (1-n+i')(i'-1)\alpha\beta\\ &= 1 + (2k - i' + 1)\alpha + (n-i')\beta + \left(-k^2 + k + 3\frac{i'(i'-1)}{2} - ni' + n\right)\alpha\beta, \qquad (5.6)\end{aligned}$$

where $k = \min(\lfloor (1+\beta)/\beta \rfloor, n-1)$, and similarly that

$$\begin{aligned}\lambda_n &= \alpha \sum_{j=0}^{n-2} |1 - j\beta| + 1 + (n-1)\alpha\beta \\ &= \alpha \left(\sum_{j=0}^{k-1} (1 - j\beta) + \sum_{j=0}^{n-k-2} ((j+k)\beta - 1) \right) + 1 + (n-1)\alpha\beta \\ &= 1 + (2k - n + 1)\alpha + \left(-k^2 + k + \frac{n(n-1)}{2} \right) \alpha\beta. \end{aligned} \tag{5.7}$$

Hence computing $\|A\|_\infty$ requires only a constant number of flops.

In order to compute $\|A^{-1}\|_\infty$, first we derive an expression for the sum of the absolute values of the elements in the ith row of the matrix $B = A^{-1}$ in (4.4), which we denote by δ_i. A tedious computation given in Appendix A yields

$$\delta_i = (1+\alpha)^i \left((1+\alpha)^{-1} + \frac{\beta\big(1 - r^{n-i}\big)}{1-r} \right), \tag{5.8}$$

where $r = (1+\alpha)(1+\beta)$. The ∞-norm of A^{-1} is the largest value of δ_i for i between 1 and n. We now explain how to compute it efficiently. We will argue that for $\Delta_i = \delta_{i+1} - \delta_i$, with i between 1 and $n-1$, one has that

(a) if $\Delta_i \le 0$, then $\Delta_k \le 0$ for $1 \le k < i$;
(b) if $\Delta_i \ge 0$, then $\Delta_k \ge 0$ for $i < k < n$.

We have

$$\begin{aligned}\Delta_i &= (1+\alpha)^{i+1} \left(\frac{1}{1+\alpha} + \frac{\beta\big(1 - r^{-i+n-1}\big)}{1-r} \right) - (1+\alpha)^i \left(\frac{1}{\alpha+1} + \frac{\beta\big(1 - r^{n-i}\big)}{1-r} \right) \\ &= \frac{(1+\alpha)^{i-1} \big(\alpha(1-r) + (1+\alpha)^2 \beta\big(1 - r^{-i+n-1}\big) - (1+\alpha)\beta\big(1 - r^{n-i}\big) \big)}{1-r} \\ &= \frac{(1+\alpha)^{i-1} \big(\alpha(1-r) + (1+\alpha)\alpha\beta + (1+\alpha)\beta r^{n-i-1}\big(r - 1 - \alpha\big) \big)}{1-r} \\ &= \frac{(1+\alpha)^{i-1} r^{-i-1} \big(-\alpha^2 r^{i+1} + (1+\alpha)^2 \beta^2 r^n \big)}{1-r} \\ &= \frac{(1+\alpha)^{-2} (1+\beta)^{-i-1} \big(\alpha^2 r^{i+1} - (1+\alpha)^2 \beta^2 r^n \big)}{r-1}. \end{aligned}$$

Therefore, the quantities Δ_i and $\zeta_i := \alpha^2 r^{i+1} - (1+\alpha)^2 \beta^2 r^n$ have the same sign since the other factors are necessarily positive as long as α and β are. Similarly, the signs of Δ_{i-1} and Δ_{i+1} depend on those of

$$\zeta_{i-1} = \alpha^2 r^i - (1+\alpha)^2 \beta^2 r^n$$

and

$$\zeta_{i+1} = \alpha^2 r^{i+2} - (1+\alpha)^2 \beta^2 r^n,$$

respectively. Since $r > 1$, $\Delta_i \le 0$ implies $\Delta_{i-1} < 0$ and$\Delta_i \ge 0$ implies $\Delta_{i+1} > 0$. The result follows by induction, and we can conclude that $\|A^{-1}\|_\infty = \max(\delta_1, \delta_n)$.

We summarize our findings in a theorem.

THEOREM 5.1. *Let* $A = A(\alpha,\beta) = T(\alpha)^T T(\beta) \in \mathbb{R}^{n\times n}$, *where* T *is defined in* (3.1). *For* $0 < \alpha \le 1$ *and* $\beta \ge \alpha$ *we have*

$$\|A\|_\infty = \max(\lambda_1, \lambda_{i'}, \lambda_n), \quad \|A^{-1}\|_\infty = \max(\delta_1, \delta_n), \tag{5.9}$$

where $i' = \min\big(\lfloor 1/\alpha \rfloor, n\big)$, *and* λ_i *and* δ_i *are defined in* (5.5)–(5.7) *and* (5.8), *respectively.*

Computing the condition number of $A(\alpha,\beta)$ therefore requires only a constant number of flops. What effect do rounding errors have on the condition number of the computed matrix? We know from (3.11) that the computed $\widehat{A}$ has a small normwise relative error. We also know that the condition number of the condition number is the condition number [6], [14]. It follows that $\big|\kappa_\infty(\widehat{A}) - \kappa_\infty(A)\big|/\kappa_\infty(A) \lesssim 6nu\kappa_\infty(A)$. Therefore if we choose the parameters α and β to achieve a condition number $\kappa_\infty(A) \ll u^{-1}$ then $\kappa_\infty(\widehat{A})$ will be of the same order of magnitude as $\kappa_\infty(A)$, which is entirely satisfactory.

5.2. Choosing the parameters. In order to generate a test matrix with a given condition number κ, it is necessary to determine values of α and β such that $\kappa_\infty\big(A(\alpha,\beta)\big) = \kappa$. From (5.9), we obtain that this is equivalent to finding a zero of the bivariate function

$$\begin{aligned} f(\alpha,\beta;n,\kappa) &= \|A(\alpha,\beta)\|_\infty \|A(\alpha,\beta)^{-1}\|_\infty - \kappa \\ &= \max(\lambda_1,\lambda_{i'},\lambda_n)\max(\delta_1,\delta_n) - \kappa, \end{aligned} \tag{5.10}$$

for $\alpha \in (0,1]$ and $\beta \ge \alpha$. The solution for a given κ is not unique.

The function f is continuous but not differentiable, and roots can be found by converting $f(\alpha,\beta;n,\kappa) = 0$ to a one-parameter equation by defining one of α and β as a fixed multiple of the other. For instance we can use a one-dimensional zero finder on the function

$$f_\rho(\beta;n,\kappa) = f(\rho\beta,\beta;n,\kappa), \quad \rho \in (0,1], \tag{5.11}$$

Note that it is necessary to ensure that for a solution β_* we have $\alpha = \rho\beta_* < 1$; if the latter inequality is not satisfied then a different value of ρ will have to be used.

6. Experimental evaluation. Now we discuss the practicalities of constructing $A(\alpha,\beta)$ and confirm the numerical stability of LU factorization without pivoting. Our codes are written in C and rely on the Intel Math Kernel Library (version 18.0.3) implementation of BLACS, PBLAS, and ScaLAPACK. For parallelization we use the Open MPI library (version 3.1.4), a widely-used implementation of the MPI standard (version 3.1) [27]. The codes we developed can be retrieved on GitHub,[8] and a MATLAB function for constructing the matrix is available.[9]

The experiments were run on the High Performance Computing Pool (HPC Pool), a component of the third generation of the Computational Shared Facility (CSF3) of the University of Manchester. Each compute node in the HPC Pool is equipped with two 16-core Intel Xeon Gold 6130 CPUs and 187GiB of RAM, and runs CentOS GNU/Linux release 7.4.1708 (Core).

The zeros of f_ρ in (5.11) were approximated numerically in double precision on a single node using the GNU Scientific Library (version 1.15) [13] implementation of

[8] https://github.com/mfasi/hpl-ai-matrix
[9] https://github.com/higham/hpl-ai-matrix

TABLE 6.1. *Values of β yielding $n \times n$ $A(\beta/2, \beta)$ with ∞-norm condition number κ.*

	$\kappa = 10^2$	$\kappa = 10^4$	$\kappa = 10^6$	$\kappa = 10^8$	$\kappa = 10^{10}$
$n = 10^2$	2.54×10^{-2}	5.35×10^{-2}	8.07×10^{-2}	1.09×10^{-1}	1.40×10^{-1}
$n = 10^3$	2.50×10^{-3}	5.21×10^{-3}	7.81×10^{-3}	1.05×10^{-2}	1.33×10^{-2}
$n = 10^4$	2.50×10^{-4}	5.20×10^{-4}	7.79×10^{-4}	1.04×10^{-3}	1.32×10^{-3}
$n = 10^5$	2.50×10^{-5}	5.19×10^{-5}	7.78×10^{-5}	1.04×10^{-4}	1.32×10^{-4}
$n = 10^6$	2.50×10^{-6}	5.19×10^{-6}	7.78×10^{-6}	1.04×10^{-5}	1.32×10^{-5}
$n = 10^7$	2.50×10^{-7}	5.19×10^{-7}	7.78×10^{-7}	1.04×10^{-6}	1.32×10^{-6}
$n = 10^8$	2.50×10^{-8}	5.19×10^{-8}	7.78×10^{-8}	1.04×10^{-7}	1.32×10^{-7}
$n = 10^9$	2.50×10^{-9}	5.19×10^{-9}	7.78×10^{-9}	1.04×10^{-8}	1.32×10^{-8}
$n = 10^{10}$	2.50×10^{-10}	5.19×10^{-10}	7.78×10^{-10}	1.04×10^{-9}	1.32×10^{-9}

the Brent–Dekker solver [2], [3], a fast and robust zero finding method that combines an interpolation strategy with the bisection algorithm. This bracketing technique requires two initial values $\underline{x}^{(0)}$ and $\overline{x}^{(0)}$ such that $f_\rho(\underline{x}^{(0)}; n, \kappa) < 0$ and $f_\rho(\overline{x}^{(0)}; n, \kappa) > 0$. In order to choose $\underline{x}^{(0)}$, we note that as β tends to 0, the matrix A tends to the identity matrix and $\kappa_\infty(A)$ tends to 1, thus we set $\underline{x}^{(0)}$ to the unit roundoff u. As for $\overline{x}^{(0)}$, from (3.10) we know that if $\beta = \rho^{-1} \geq 1$ (recalling (5.11)), which ensures that $\alpha = \rho\beta = 1$, then $\kappa_\infty(A)$ is bounded below by

$$\big(1 + (n-1)\rho^{-1}\big)\rho^{-1}(1 + \rho^{-1})^{n-2} \geq \big(1 + (n-1)\big)2^{n-2},$$

which for $n = 100$ is approximately 3×10^{31} and guarantees that if $\overline{x}^{(0)} = \rho^{-1}$ then $f_\rho(\overline{x}^{(0)}; n, \kappa)$ is positive for all values of n and κ of interest. In fact, this value typically overflows if f_ρ is evaluated in binary64 arithmetic. Therefore, in our code we initially set $\overline{x}^{(0)} = \rho^{-1}$ and keep dividing it by 2 until we find a value such that $f_\rho\big(\overline{x}^{(0)}; n, \kappa\big)$ is positive and representable in binary64 format. At each step, the Brent–Dekker iteration returns an approximate solution $\widehat{x}^{(i)}$ such that $f_\rho(\widehat{x}^{(i)}; n, \kappa) \approx 0$ and updates the bracketing interval producing $\underline{x}^{(i)}$ and $\overline{x}^{(i)}$ such that $\widehat{x}^{(i)} \in \big[\underline{x}^{(i)}, \overline{x}^{(i)}\big] \subset \big[\underline{x}^{(i-1)}, \overline{x}^{(i-1)}\big]$. In our code, we keep iterating until

$$|\overline{x}^{(i)} - \underline{x}^{(i)}| < 2u\underline{x}^{(i)},$$

where u is the unit roundoff of binary64 precision.

6.1. Values of α and β. To give a feeling for the typical values of α and β, we compute them for a range of matrix sizes and target condition numbers. Table 6.1 shows the values of β when $\alpha = \rho\beta$ for $\rho = 1/2$. There is little variation of β with κ, but a large variation with n on this very large range. The corresponding tables for $\rho = 1/10$ (see Table 6.3) and $\rho = 3/4$ have elements of the same orders of magnitude.

Table 6.2 reports the absolute value of the smallest element of $A(\beta/2, \beta)$. The absolute value of the largest element is between 1 and 1.32 for all the matrices we considered. We will discuss the implications of the wide dynamic range of elements for low precision in section 7.1.

6.2. Pivots and growth factor. The second and fourth columns of Table 6.3 report, for values of n between 1,000 and 200,000 and $\rho = 1/10$, values of β yielding $\kappa = 10^3$ and $\kappa = 10^6$, respectively. We used these values to generate the matrix

TABLE 6.2. *Values of* $\min_{i,j} |A(\beta/2,\beta))_{ij}|$ *for the values of* β *in Table 6.1.*

	$\kappa = 10^2$	$\kappa = 10^4$	$\kappa = 10^6$	$\kappa = 10^8$	$\kappa = 10^{10}$
$n = 10^2$	1.05×10^{-4}	4.21×10^{-4}	6.67×10^{-4}	8.13×10^{-4}	1.48×10^{-3}
$n = 10^3$	8.31×10^{-7}	5.68×10^{-7}	6.81×10^{-7}	1.64×10^{-5}	2.09×10^{-5}
$n = 10^4$	7.98×10^{-9}	5.94×10^{-9}	9.82×10^{-8}	1.09×10^{-8}	2.01×10^{-7}
$n = 10^5$	4.32×10^{-11}	2.21×10^{-10}	9.24×10^{-10}	9.98×10^{-10}	2.70×10^{-9}
$n = 10^6$	1.07×10^{-13}	2.31×10^{-12}	4.92×10^{-12}	1.03×10^{-11}	3.10×10^{-12}
$n = 10^7$	7.41×10^{-15}	2.68×10^{-14}	2.45×10^{-14}	5.19×10^{-14}	1.24×10^{-13}
$n = 10^8$	7.69×10^{-18}	7.41×10^{-17}	9.09×10^{-16}	4.16×10^{-16}	1.87×10^{-15}
$n = 10^9$	4.48×10^{-19}	4.53×10^{-19}	8.70×10^{-18}	7.97×10^{-18}	1.07×10^{-17}
$n = 10^{10}$	7.54×10^{-21}	5.46×10^{-21}	2.71×10^{-20}	1.66×10^{-20}	2.47×10^{-19}

TABLE 6.3. *Values of* β *yielding the* $n \times n$ *matrix* $A(\beta/10,\beta)$ *with* ∞*-norm condition number* κ *and values of* ϑ *in* (6.1).

	$\kappa = 10^3$		$\kappa = 10^6$	
	β	ϑ	β	ϑ
$n = 1 \times 10^3$	4.79×10^{-3}	4.44×10^{-7}	1.05×10^{-2}	2.85×10^{-4}
$n = 2 \times 10^3$	2.39×10^{-3}	3.86×10^{-7}	5.23×10^{-3}	2.26×10^{-5}
$n = 5 \times 10^3$	9.55×10^{-4}	4.06×10^{-7}	2.09×10^{-3}	9.92×10^{-5}
$n = 1 \times 10^4$	4.77×10^{-4}	2.63×10^{-7}	1.04×10^{-3}	3.90×10^{-5}
$n = 2 \times 10^4$	2.39×10^{-4}	1.19×10^{-8}	5.22×10^{-4}	1.77×10^{-4}
$n = 5 \times 10^4$	9.55×10^{-5}	2.14×10^{-7}	2.09×10^{-4}	6.66×10^{-5}
$n = 1 \times 10^5$	4.77×10^{-5}	7.35×10^{-8}	1.04×10^{-4}	2.67×10^{-5}
$n = 2 \times 10^5$	2.39×10^{-5}	1.75×10^{-7}	5.22×10^{-5}	2.63×10^{-4}

$A(\rho\beta,\beta)$ in (3.5). We computed the LU factorization $A \approx \widehat{L}\widehat{U}$ in binary32 arithmetic with the ScaLAPACK function `psgetrf`, which uses partial pivoting, and confirmed that no row interchanges were performed during the reduction to row-echelon form.

In order to gauge the stability of Gaussian elimination for these test matrices, we looked at the relative error in the computed multipliers (the elements of L), that is,

$$\vartheta = \frac{1}{\alpha} \max_{1 \le j < i \le n} |\alpha - \widehat{\ell}_{ij}|. \tag{6.1}$$

This quantity is reported in the third and fifth columns of Table 6.3. The small values of ϑ confirm that for the matrices our new algorithm generates roundoff errors do not accumulate in a harmful way. In particular, the fact that ϑ does not appear to be growing with n suggests that the accuracy of the pivots does not depend on the order of the matrix being generated, but only on the required ∞-norm condition number.

7. Adaptations for HPL-AI Mixed-Precision Benchmark. We now examine the suitability of $A(\alpha,\beta)$ as a test matrix for the HPL-AI benchmark.

7.1. Scaling for low precision. The reference implementation of the HPL-AI benchmark factorizes A in binary32, but in practice binary16 or bfloat16 might be used instead. We therefore consider whether $A(\alpha,\beta)$ can be suitably represented in these precisions. We know that $\max_{i,j} |a_{ij}|$ will be of order 1, so overflow will not occur during the factorization, since the growth factor $\rho_n = 1$ (see (3.12)).

All the elements in Table 6.2 exceed the smallest normal number 1.18×10^{-38} for binary32, and likewise for bfloat16, which has essentially the same range as binary32. In binary16, however, the smallest positive subnormal and normal numbers are approximately 6.0×10^{-8} and 6.1×10^{-5}, respectively, thus underflow or the occurrence of subnormal numbers are inevitable for larger values of n and κ when the matrix is converted to this low-precision floating-point format. This is not a problem as regards numerical stability, but subnormal numbers can incur a performance penalty, especially if handled in software [23], [28, sect. 8.1.2], [31]. The number of entries underflowing or becoming subnormal can be reduced by working with the matrix $\psi A(\alpha, \beta)$ for $\psi > 1$, as suggested in [19]. One can safely take $\psi = f_{\max}/2$, say, where $f_{\max} = 65{,}504$ is the largest finite binary16 floating-point number.

A key requirement is that throughout the factorization all row operations must be necessary at each stage, so that no amount of computation can be skipped. In other words, we need to ensure that at each stage of elimination all the elements below the diagonal are nonzero. By construction, the vector $[a_{kk}^{(k)}, a_{k+1,k}^{(k)}, \dots a_{nk}^{(k)}]$ is $[1, -\alpha, \dots, -\alpha]$ at every stage, and scaling A to ψA changes it to $[\psi, -\psi\alpha, \dots, -\psi\alpha]$. If we consider $n = 10^8$ and $\kappa = 10^4$, for example, then using the β value from Table 6.1 with $\psi = f_{\max}/2$ gives $\psi\alpha \approx 8.5 \times 10^{-4}$, which is safely above the smallest normal binary16 number.

Our conclusion is that multiplying $A(\alpha, \beta)$ by an appropriate scale factor makes the matrix suitable for LU factorization in binary16 in the HPL-AI benchmark.

7.2. Forcing computation of the LU factorization. A possible drawback is that the LU factorization of $A(\alpha, \beta)$ is explicitly known. An unscrupulous benchmarker could form the exact LU factors from (3.3) at trivial cost without carrying out an LU factorization. An obvious way to avoid this problem is to add a small random perturbation to A in order to change the LU factors. Since the multipliers are continuous functions of the matrix entries and are all equal to $\alpha < 1$, a sufficiently small perturbation of $A(\alpha, \beta)$ will retain multipliers less than 1 and growth factor of order 1.

Consider a rank-1 perturbation $\Delta A = \varepsilon e_i e_j^T$, where e_i is the ith unit vector and $\varepsilon \in \mathbb{R}$. For small enough ε, the LU factorization $A + \Delta A = (L + \Delta L)(U + \Delta U)$ exists. Since the perturbation matrices ΔL and ΔU are strictly lower triangular and upper triangular, respectively, it is easy to show that, to first order,

$$\Delta L = L\,\mathrm{tril}(L^{-1}\Delta A U^{-1}) = \varepsilon L\,\mathrm{tril}\big(L^{-1} e_i e_j^T U^{-1}\big),$$

where tril denotes the strictly lower triangular part [32], [33]. Hence we certainly have

$$|\Delta L| \le |\varepsilon||L||L^{-1}|e_i e_j^T|U^{-1}|.$$

It is easy to show using (3.2) that the elements in the ith column of $|L||L^{-1}|$ are, from the ith position onwards, 1, 2α, $2\alpha(1+\alpha)$, $\dots$, $2\alpha(1+\alpha)^{n-i-1}$, which implies that

$$(|\Delta L|)_{rs} \le |\varepsilon| 2\alpha(1+\alpha)^{n-s-1}\beta(1+\beta)^{n-r-1} \le |\varepsilon| 2\alpha\beta(1+\alpha)^{n-2}(1+\beta)^{n-2}.$$

In order for the multipliers to remain bounded by 1, we wish this bound to be less than $1 - \alpha$, which can be achieved by choosing ε such that

$$|\varepsilon| 2\alpha\beta(1+\alpha)^{n-2}(1+\beta)^{n-2} < 1 - \alpha. \tag{7.1}$$

In Table 6.1 the values of $|\varepsilon|$ satisfying this inequality as an equality range from 1.22×10^{-6} for $n = 10^2$, $\kappa_\infty(A) = 10^{10}$ to 1.51×10^{27} for $n = 10^{10}$, $\kappa_\infty(A) = 10^2$.

TABLE 7.1. *Number of GMRES iterations to solve* $\widetilde{A}(\beta/4,\beta,\xi)x = b$ *for the matrix in* (7.3) *with* $\kappa_\infty(\widetilde{A}) = 10^6$*, for different dimensions* n *and restart parameters* m*.*

	m				
	$n/3$	$n/2$	$2n/3$	$3n/4$	n
$n = 1.0 \times 10^4$	999	1,000	667	677	677
$n = 2.5 \times 10^4$	2,499	1,610	1,284	1,284	1,284
$n = 5.0 \times 10^4$	2,665	1,952	1,952	1,952	1,952
$n = 7.5 \times 10^4$	2,402	2,402	2,402	2,402	2,402
$n = 1.0 \times 10^5$	2,718	2,718	2,718	2,718	2,718

In practice, a full rank perturbation is preferable. We suggest using the matrix

$$A(\alpha,\beta,\xi) = A(\alpha,\beta) + \xi\,\mathrm{diag}(1,-1,1,-1,\dots), \tag{7.2}$$

where ξ is the smaller of $cu^{1/2}$ and the largest positive ε satisfying (7.1), where c is a some positive constant bounded by 1.

7.3. Forcing preconditioning. Is it necessary to precondition GMRES when we solve $A(\alpha,\beta)x = b$? For the matrix (7.2), our experiments indicate that unpreconditioned GMRES converges in a few tens of iterations and the number of iterations does not vary much with n. In order to make it necessary to precondition, we can carry out a two-sided diagonal scaling

$$\widetilde{A}(\alpha,\beta,\xi) = D_1 A(\alpha,\beta,\xi) D_2 = D_1 L U D_2 = D_1 L D_1^{-1} \cdot D_1 U D_2 \equiv \widetilde{L}\widetilde{U}, \tag{7.3}$$

where D_1 and D_2 are diagonal matrices with positive diagonal elements. As long as the diagonal elements of D_1 are nonincreasing, $\widetilde{L}$ has subdiagonal elements bounded by 1 and LU factorization without pivoting remains numerically stable for $\widetilde{A}(\alpha,\beta,\xi)$. We have

$$\frac{\kappa_\infty(\widetilde{A}(\alpha,\beta,\xi))}{\kappa_\infty(D_1)\kappa_\infty(D_2)} \le \kappa_\infty(A(\alpha,\beta,\xi)) \le \kappa_\infty(D_1)\kappa_\infty(D_2)\kappa_\infty(\widetilde{A}(\alpha,\beta,\xi)),$$

so we lose the ability to precisely determine the condition number.

Our experiments indicate that a mild diagonal scaling is enough to slow down the convergence of unpreconditioned GMRES. We describe an experiment in which we chose β and $\alpha = \beta/4$ so that $\kappa_\infty(A(\alpha,\beta,\xi)) = 10^6$ and used the matrix $\widetilde{A}$ in (7.3) with ξ chosen as above, with D_1 and D_2 having logarithmically equally spaced entries between 1 and 10^{-3} for D_1 and 1 and 10^{-2} for D_2. The ∞-norm condition number of the scaled matrix $\widetilde{A}$ was of order 10^6 in every case. Table 7.1 gives the number of restarted GMRES iterations needed to solve $\widetilde{A}x = b$ in double precision arithmetic, where b has random elements from the (0,1) distribution, with backward error satisfying (2.1) with $\mu = 16$. Here we are using the MATLAB `gmres` function. Several values of n and the restart parameter m are used. We can see that a significant number of iterations is required, which means that preconditioning is necessary to solve a linear system with coefficient matrix involving $\widetilde{A}$.

8. Conclusions. Constructing in $O(n^2)$ operations an $n \times n$ matrix having a specified ∞-norm condition number and such that LU factorization without pivoting is numerically stable is not a trivial task, especially when the matrix must be suitable

for extreme-scale dimensions $n > 10^7$ and for use with low-precision floating-point arithmetic. We have shown that the two-parameter family of matrices $A(\alpha,\beta)$ in (3.3) satisfies these requirements, with α and β determined by solving a scalar nonlinear equation. These matrices, with the modifications described in section 7 if necessary, are therefore suitable for use in HPL-AI Mixed-Precision Benchmark. They are also useful as general purpose dense nonsymmetric test matrices.

Acknowledgements. We acknowledge the assistance given by Research IT at the University of Manchester, and the use of the HPC Pool funded by the Research Lifecycle Programme at the University of Manchester. We also thank Sven Hammarling for comments on a draft manuscript and Jack Dongarra and Piotr Luszczek for discussions about the HPL-AI benchmark.

Appendix A. Proof of (5.8).

The elements of $B = A(\alpha,\beta)^{-1} = U^{-1}L^{-1} = T(\beta)^{-1}T(\alpha)^{-T}$ are all nonnegative, by (3.2). Hence

$$
\begin{aligned}
\delta_i &= \sum_{j=1}^{n} |b_{ij}| = \sum_{j=1}^{n} b_{ij} \\
&= \sum_{j=1}^{i-1} \left(\alpha(1+\alpha)^{i-j-1} + \frac{\alpha\beta(1+\alpha)^{i-j}\big(1-r^{n-i}\big)}{1-r} \right) + 1 + \frac{\alpha\beta\big(1-r^{n-i}\big)}{1-r} \\
&\qquad + \sum_{j=i+1}^{n} \left(\beta(1+\beta)^{-i+j-1} + \frac{\alpha\beta(1+\beta)^{j-i}\big(1-r^{n-j}\big)}{1-r} \right) \\
&= \alpha(1+\alpha)^{i-1} \left((1+\alpha)^{-1} + \frac{\beta\big(1-r^{n-i}\big)}{1-r} \right) \sum_{j=0}^{i-2} (1+\alpha)^{-j} + 1 + \frac{\alpha\beta\big(1-r^{n-i}\big)}{1-r} \\
&\qquad + \beta \sum_{j=0}^{n-i-1} (1+\beta)^{j} \left(1 + \frac{\alpha(1+\beta)\big(1-r^{n-i-j-1}\big)}{1-r} \right) \\
&= \left((1+\alpha)^{-1} + \frac{\beta\big(1-r^{n-i}\big)}{1-r} \right) (1+\alpha) \left((1+\alpha)^{i-1} - 1 \right) + 1 + \frac{\alpha\beta\big(1-r^{n-i}\big)}{1-r} \\
&\qquad - \big(1-(1+\beta)^{n-i}\big) \left(1 + \frac{\alpha(1+\beta)}{1-r} \right) - \frac{\beta\big(r^{n-i} - (1+\beta)^{n-i}\big)}{1-r} \\
&= \left((1+\alpha)^{-1} + \frac{\beta\big(1-r^{n-i}\big)}{1-r} \right) (1+\alpha)\big((1+\alpha)^{i-1} - 1\big) + 1 + \frac{\alpha\beta\big(1-r^{n-i}\big)}{1-r} \\
&\qquad + \frac{\beta\big(1-r^{n-i}\big)}{1-r} \\
&= (1+\alpha)^{i} \left((1+\alpha)^{-1} + \frac{\beta\big(1-r^{n-i}\big)}{1-r} \right).
\end{aligned}
$$

REFERENCES

[1] J. H. Ahlberg and E. N. Nilson, *Convergence properties of the spline fit*, J. Soc. Indust. Appl. Math., 11 (1963), pp. 95–104.

[2] R. P. Brent, *An algorithm with guaranteed convergence for finding a zero of a function*, Comput. J., 14 (1971), pp. 422–425.

[3] J. C. P. Bus and T. J. Dekker, *Two efficient algorithms with guaranteed convergence for finding a zero of a function*, ACM Trans. Math. Software, 1 (1975), pp. 330–345.

[4] E. Carson and N. J. Higham, *Accelerating the solution of linear systems by iterative refinement in three precisions*, SIAM J. Sci. Comput., 40 (2018), pp. A817–A847.

[5] T. A. Davis and Y. Hu, *The University of Florida sparse matrix collection*, ACM Trans. Math. Software, 38 (2011), pp. 1:1–1:25.

[6] J. W. Demmel, *On condition numbers and the distance to the nearest ill-posed problem*, Numer. Math., 51 (1987), pp. 251–289.

[7] J. W. Demmel and A. McKenney, *A test matrix generation suite*, Preprint MCS-P69-0389, Mathematics and Computer Science Division, Argonne National Laboratory, Argonne, IL, USA, Mar. 1989. LAPACK Working Note 9.

[8] J. Dongarra, M. A. Heroux, and P. Luszczek, *High-performance conjugate-gradient benchmark: A new metric for ranking high-performance computing systems*, Int. J. High Performance Computing Applications, 30 (2016), pp. 3–10.

[9] J. Dongarra, M. A. Heroux, and P. Luszczek, *A new metric for ranking high-performance computing systems*, National Science Review, 3 (2016), pp. 30–35.

[10] J. J. Dongarra, P. Luszczek, and A. Petitet, *The LINPACK benchmark: Past, present and future*, Concurrency Computat.: Pract. Exper., 15 (2003), pp. 803–820.

[11] I. S. Duff, R. G. Grimes, and J. G. Lewis, *Sparse matrix test problems*, ACM Trans. Math. Software, 15 (1989), pp. 1–14.

[12] M. Fasi and N. J. Higham, *Generating extreme-scale matrices with specified singular values or condition numbers*, MIMS EPrint 2020.8, Manchester Institute for Mathematical Sciences, The University of Manchester, UK, Mar. 2020. Revised October 2020. To appear in SIAM J. Sci. Comput.

[13] M. Galassi, J. Davies, J. Theiler, B. Gough, G. Jungman, P. Alken, M. Booth, and F. Rossi, *GNU Scientific Library Reference Manual*, Network Theory, 3rd ed., 2009.

[14] D. J. Higham, *Condition numbers and their condition numbers*, Linear Algebra Appl., 214 (1995), pp. 193–213.

[15] N. J. Higham, *Algorithm 694: A collection of test matrices in MATLAB*, ACM Trans. Math. Software, 17 (1991), pp. 289–305.

[16] N. J. Higham, *Accuracy and Stability of Numerical Algorithms*, Society for Industrial and Applied Mathematics, Philadelphia, PA, USA, second ed., 2002.

[17] N. J. Higham, *Error analysis for standard and GMRES-based iterative refinement in two and three-precisions*, MIMS EPrint 2019.19, Manchester Institute for Mathematical Sciences, The University of Manchester, UK, Nov. 2019.

[18] N. J. Higham and T. Mary, *A new approach to probabilistic rounding error analysis*, SIAM J. Sci. Comput., 41 (2019), pp. A2815–A2835.

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

[20] Institute of Electrical and Electronics Engineers, *IEEE Standard for Floating-Point Arithmetic, IEEE Std 754-2019 (revision of IEEE Std 754-2008)*, Institute of Electrical and Electronics Engineers, Piscataway, NJ, USA, July 2019.

[21] Intel Corporation, *BFLOAT16—hardware numerics definition*. White paper. Document number 338302-001US, Nov. 2018.

[22] S. Kudo, K. Nitadori, T. Ina, and T. Imamura, *Implementation and numerical techniques for one EFlop/s HPL-AI benchmark on Fugaku*, in Proceedings of the 11th IEEE/ACM Workshop on Latest Advances in Scalable Algorithms for Large-Scale, 2020, pp. 69–76.

[23] O. Lawlor, H. Govind, I. Dooley, M. Breitenfeld, and L. Kale, *Performance degradation in the presence of subnormal floating-point values*, in Proceedings of the International Workshop on Operating System Interference in High Performance Applications, 2005.

[24] P. Luszczek, J. J. Dongarra, D. Koester, R. Rabenseifner, B. Lucas, J. Kepner, J. McCalpin, D. Bailey, and D. Takahashi, *Introduction to the HPC Challenge benchmark suite*, Technical Report ICL-UT-05-01, Innovative Computing Laboratory, The University of Tennessee, Knoxville, TN, 2005.

[25] V. Marjanović, J. Gracia, and C. W. Glass, *Performance modeling of the HPCG benchmark*, High Performance Computing Systems. Performance Modeling, Benchmarking, and

Simulation, (2015), pp. 172–192.
[26] C. B. Moler, *A matrix for the new HPL-AI benchmark*. https://blogs.mathworks.com/cleve/2019/12/04/a-matrix-for-the-new-hpl-ai-benchmark/, Dec. 2019.
[27] MPI Forum, *MPI: A Message-Passing Interface Standard, Version 3.1*, High Performance Computing Center Stuttgart (HLRS), 2015.
[28] J.-M. Muller, N. Brunie, F. de Dinechin, C.-P. Jeannerod, M. Joldes, V. Lefèvre, G. Melquiond, N. Revol, and S. Torres, *Handbook of Floating-Point Arithmetic*, Birkhäuser, 2nd ed., 2018.
[29] A. M. Ostrowski, *On the spectrum of a one-parametric family of matrices*, J. Reine Angew. Math., 193 (1954), pp. 143–160.
[30] A. Petitet, R. C. Whaley, J. Dongarra, and A. Cleary, *HPL: A portable implementation of the High-Performance Linpack benchmark for distributed-memory computers, Version 2.3*, 2018.
[31] E. M. Schwarz, M. Schmookler, and S. D. Trong, *FPU implementations with denormalized numbers*, IEEE Trans. Comput., 54 (2005), p. 825–836.
[32] G. W. Stewart, *On the perturbation of LU and Cholesky factors*, IMA J. Numer. Anal., 17 (1997), pp. 1–6.
[33] J. Sun, *Componentwise perturbation bounds for some matrix decompositions*, BIT, 32 (1992), pp. 702–714.
[34] J. M. Varah, *A lower bound for the smallest singular value of a matrix*, Linear Algebra Appl., 11 (1975), pp. 3–5.
[35] J. H. Wilkinson, *Error analysis of direct methods of matrix inversion*, J. Assoc. Comput. Mach., 8 (1961), pp. 281–330.
[36] W. Zhang and N. J. Higham, *Matrix Depot: An extensible test matrix collection for Julia*, PeerJ Comput. Sci., 2 (2016), p. e58.