Homework 4

Math 5424

Numerical Linear Algebra

Alexander Novotny
Zuriah Quinton

November 4, 2023

1. Let $\boldsymbol{L}$ be a lower triangular matrix and solve $\boldsymbol{L}\vec{x} = \vec{b}$ by forward substitution. Show that barring overflow or underflow, the computed solution $\hat{x}$ satisfies $(\boldsymbol{L} + \delta\boldsymbol{L})\hat{x} = \vec{b}$, where $|\delta l_{ij}| \leq n\varepsilon |l_{ij}|$, where ε is the machine precision. This means that forward substitution is backward stable. Argue that backward substitution for solving upper triangular systems satisfies the same bound.

   Proof. We have, for the true solution $\vec{x}$,

$$x_i = \left(b_i - \sum_{k=1}^{i-1} l_{ik}x_k\right)/l_{ii}.$$

   For the solution $\hat{x}$ computed via backwards substitution,

$$\hat{x}_i = \left(b_i - \sum_{k=1}^{i-1}\left[l_{ik}\hat{x}_k(1+\varepsilon_k^*)\prod_{j=k}^{i-1}(1+\varepsilon_k^+)\right]\right)(1+\varepsilon^-)(1+\varepsilon^/)/l_{ii}$$

   (where $|\varepsilon_k^*|, |\varepsilon_k^+|, |\varepsilon^-|, |\varepsilon^/| < eps$)

$$= \left(b_i - \sum_{k=1}^{i-1}\left[l_{ik}(1+\varepsilon_k^*)\prod_{j=k}^{i-1}(1+\varepsilon_j^+)\right]\hat{x}_k\right)\Big/\left(\frac{l_{ii}}{(1+\varepsilon^-)(1+\varepsilon^/)}\right)$$

$$= \left(b_i - \sum_{k=1}^{i-1}\left[l_{ik}\left(1+\varepsilon_k^* + \sum_{j=k}^{i-1}\varepsilon_j^+ + \mathcal{O}(eps^2)\right)\right]\hat{x}_k\right)\Big/\left(l_{ii}(1-\varepsilon^- - \varepsilon^/ + \mathcal{O}(eps^2))\right).$$

   Then we have that $(\boldsymbol{L} + \delta\boldsymbol{L})\hat{x} = \vec{b}$ for

$$\delta l_{ij} = \begin{cases} l_{ij}(-\varepsilon^- - \varepsilon^/), & i = j \\ l_{ij}\left(\varepsilon_j^* + \sum_{k=j}^{i-1}\varepsilon_k^+\right), & i \neq j \end{cases}$$

$$= l_{ij}\begin{cases} -\varepsilon^- - \varepsilon^/, & i = j \\ \varepsilon_j^* + \sum_{k=j}^{i-1}\varepsilon_k^+, & i \neq j \end{cases}$$

$$\Longrightarrow |\delta l_{ij}| = |l_{ij}|\begin{cases} |(-\varepsilon^- - \varepsilon^/)|, & i = j \\ \left|\varepsilon_j^* + \sum_{k=j}^{i-1}\varepsilon_k^+\right|, & i \neq j \end{cases}$$

   (note that for $i = 1$, $\varepsilon^- = 0$, since we subtract by 0 - which will never round)

$$\leq |l_{ij}|\begin{cases} |\varepsilon^/|, & i = j = 1 \\ |\varepsilon^-| + |\varepsilon^/|, & i = j \neq 1 \\ |\varepsilon_j^*| + \sum_{k=j}^{i-1}|\varepsilon_k^+|, & i \neq j \end{cases}$$

$$\leq |l_{ij}|\begin{cases} eps, & i = j = 1 \\ 2eps, & i = j \neq 1 \\ (i - j + 1)eps, & i \neq j \end{cases}$$

$$\leq n \cdot eps|l_{ij}|$$

2. Matrix $\boldsymbol{A}$ is called *strictly column diagonally dominant*, or diagonally dominant for short, if

$$|a_{ii}| > \sum_{j=1, j\neq i}^{n} |a_{ji}| \tag{1}$$

Show that Gaussian elimination with partial pivoting does not actually permute any rows, i.e., that it is identical to Gaussian elimination without pivoting. Hint: Show that after one step of Gaussian elimination, the trailing $(n-1)$-by-$(n-1)$ submatrix, the *Schur complement* of a_{11} in $\boldsymbol{A}$, is still diagonally dominant.

Proof. For a diagonally dominant matrix $\boldsymbol{A}$, the first step of Gaussian elimination with partial pivoting does not permute any rows, since $|\alpha_{ii}| > \sum_{j=1,j\neq i}^{n} |\alpha_{ji}|$ and thus $|\alpha_{ii}| > |\alpha_{ji}|$ for any $j \neq i$. Then, we have the decomposition for $\boldsymbol{A}$ given by the first step of Gaussian elimination as

$$\begin{aligned}
\boldsymbol{A} &= \begin{bmatrix} \alpha_{11} & \vec{c}^{\top} \\ \vec{a} & \hat{\boldsymbol{A}} \end{bmatrix} \\
&= \begin{bmatrix} 1 & 0 \\ \vec{l} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \alpha_{11}\vec{c}^{\top} & \\ 0 & \hat{\boldsymbol{A}} - \vec{l}\vec{c}^{\top} \end{bmatrix} \\
&= \begin{bmatrix} 1 & 0 \\ \vec{l} & \boldsymbol{I} \end{bmatrix} \begin{bmatrix} \alpha_{11}\vec{c}^{\top} & \\ 0 & \boldsymbol{B} \end{bmatrix}
\end{aligned}$$

where $\vec{l} = \frac{\vec{a}}{\alpha_{11}}$. Note that $c_i = \alpha_{1,i+1},\ l_i = \frac{\alpha_{i+1,1}}{\alpha_{11}}$. Then, denote the submatrix $\boldsymbol{B} = \hat{\boldsymbol{A}} - \vec{l}\vec{c}^{\top}$. Now, it is sufficient to show that $\boldsymbol{B}$ is diagonally dominant, since then we could continue each step of Gaussian elimination recursively always getting a diagonally dominant submatrix. First note

$$\beta_{ji} = \alpha_{j+1,i+1} - \frac{\alpha_{j+1,1}\alpha_{1,i+1}}{\alpha_{11}} \tag{2}$$

is the expression for the elements in $\boldsymbol{B}$ by definition. Further, from eq. (1) we can split one of the terms out of the sum, and we have

$$|\alpha_{ii}| - \sum_{\substack{j=1 \\ j\neq i,k}} |\alpha_{ji}| > |\alpha_{ki}|. \tag{3}$$

We have

$$\begin{aligned}
|\beta_{ii}| &\overset{(2)}{=} \left| \alpha_{j+1,i+1} - \frac{\alpha_{i+1,1}\alpha_{1,i+1}}{\alpha_{11}} \right| \\
&\geq |\alpha_{j+1,i+1}| - \left| \frac{\alpha_{i+1,1}\alpha_{1,i+1}}{\alpha_{11}} \right| \qquad \text{by reverse triangle inequality} \\
&\overset{(1)}{>} \sum_{\substack{j=1 \\ j\neq i+1}}^{n} |\alpha_{j,i+1}| - \left| \frac{\alpha_{i+1,1}\alpha_{1,i+1}}{\alpha_{11}} \right| \\
&\overset{???}{>} \sum_{\substack{j=1 \\ j\neq i+1}}^{n} |\alpha_{j,i+1}| - \left| \frac{\alpha_{1,i+1}}{\alpha_{11}} \right| \left(|\alpha_{11}| - \sum_{\substack{j=1 \\ j\neq 1,i+1}}^{n} |\alpha_{j,1}| \right) \\
&= \sum_{\substack{j=1 \\ j\neq i+1}}^{n} |a_{j,i+1}|
\end{aligned}$$

3. Let $\boldsymbol{A}$, $\boldsymbol{B}$, and $\boldsymbol{C}$ be matrices with dimensions such that the product $\boldsymbol{A}^{\top}\boldsymbol{C}\boldsymbol{B}^{\top}$ is well defined. Let $\mathcal{X}$ be the set of matrices $\boldsymbol{X}$ minimizing $\|\boldsymbol{A}\boldsymbol{X}\boldsymbol{B} - \boldsymbol{C}\|_F$, and let $\boldsymbol{X}_0$ be the unique member of $\mathcal{X}$ minimizing $\|\boldsymbol{X}\|_F$. Show that $\boldsymbol{X}_0 = \boldsymbol{A}^{+}C\boldsymbol{B}^{+}$. Hint: Use the SVDs of $\boldsymbol{A}$ and $\boldsymbol{B}$.

Proof.

4. Show that the Moore—Penrose pseudoinverse of $\boldsymbol{A}$ satisfies the following identities:

$$\begin{aligned}\boldsymbol{A}\boldsymbol{A}^+\boldsymbol{A} &= \boldsymbol{A},\\ \boldsymbol{A}^+\boldsymbol{A}\boldsymbol{A}^+ &= \boldsymbol{A}^+,\\ \boldsymbol{A}^+\boldsymbol{A} &= (\boldsymbol{A}^+\boldsymbol{A})^\top,\\ \boldsymbol{A}\boldsymbol{A}^+ &= (\boldsymbol{A}\boldsymbol{A}^+)^\top.\end{aligned}$$

Proof. We have

$$\begin{aligned}\boldsymbol{A}\boldsymbol{A}^+\boldsymbol{A} &= \boldsymbol{A}(\boldsymbol{A}^\top\boldsymbol{A})^{-1}\boldsymbol{A}^\top\boldsymbol{A}\\ &= \boldsymbol{A}\cancel{(\boldsymbol{A}^\top\boldsymbol{A})^{-1}}\cancel{(\boldsymbol{A}^\top\boldsymbol{A})}\\ &= \boldsymbol{A}.\end{aligned}$$

Then,

$$\begin{aligned}\boldsymbol{A}^+\boldsymbol{A}\boldsymbol{A}^+ &= (\boldsymbol{A}^\top\boldsymbol{A})^{-1}\boldsymbol{A}^\top\boldsymbol{A}\boldsymbol{A}^+\\ &= \cancel{(\boldsymbol{A}^\top\boldsymbol{A})^{-1}}\cancel{(\boldsymbol{A}^\top\boldsymbol{A})}\boldsymbol{A}^+\\ &= \boldsymbol{A}^+.\end{aligned}$$

Further,

$$\begin{aligned}\boldsymbol{A}^+\boldsymbol{A} &= (\boldsymbol{A}^\top\boldsymbol{A})^{-1}\boldsymbol{A}^\top\boldsymbol{A}\\ &= \cancel{(\boldsymbol{A}^\top\boldsymbol{A})^{-1}}\cancel{(\boldsymbol{A}^\top\boldsymbol{A})}\\ &= \boldsymbol{I},\end{aligned}$$

and

$$\begin{aligned}(\boldsymbol{A}^+\boldsymbol{A})^\top &= \boldsymbol{A}^\top(\boldsymbol{A}^+)^\top\\ &= \boldsymbol{A}^\top((\boldsymbol{A}^\top\boldsymbol{A})^{-1}\boldsymbol{A}^\top)^\top\\ &= \boldsymbol{A}^\top\boldsymbol{A}\Big((\boldsymbol{A}^\top\boldsymbol{A})^{-1}\Big)^\top\\ &= \boldsymbol{A}^\top\boldsymbol{A}\Big((\boldsymbol{A}^\top\boldsymbol{A})^\top\Big)^{-1}\\ &= \cancel{(\boldsymbol{A}^\top\boldsymbol{A})}\cancel{(\boldsymbol{A}^\top\boldsymbol{A})^{-1}}\\ &= \boldsymbol{I},\end{aligned}$$

so $\boldsymbol{A}^+\boldsymbol{A} = (\boldsymbol{A}^+\boldsymbol{A})^\top$. Finally,

$$\begin{aligned}(\boldsymbol{A}\boldsymbol{A}^+)^\top &= (\boldsymbol{A}^+)^\top\boldsymbol{A}^\top\\ &= ((\boldsymbol{A}^\top\boldsymbol{A})^{-1}\boldsymbol{A}^\top)^\top\boldsymbol{A}^\top\\ &= \boldsymbol{A}\Big((\boldsymbol{A}^\top\boldsymbol{A})^{-1}\Big)^\top\boldsymbol{A}^\top\\ &= \boldsymbol{A}\Big((\boldsymbol{A}^\top\boldsymbol{A})^\top\Big)^{-1}\boldsymbol{A}^\top\\ &= \boldsymbol{A}(\boldsymbol{A}^\top\boldsymbol{A})^{-1}\boldsymbol{A}^\top\\ &= \boldsymbol{A}\boldsymbol{A}^+.\end{aligned}$$

5. (a) Describe a variant of Gaussian elimination that introduces zeros into the columns of $\boldsymbol{A}$ in the order n : -1 : 2 and which produces the factorization $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{L}$ where $\boldsymbol{U}$ is the unit upper triangular and $\boldsymbol{L}$ is lower triangular.

Answer. Note that for a 1×1 matrix $\boldsymbol{A}$, we have $\boldsymbol{A} = \boldsymbol{I}\boldsymbol{A}$, where $\boldsymbol{I}$ is unit upper triangular, and $\boldsymbol{A}$ is lower triangular. Then for an $n \times n$ matrix $\boldsymbol{A}$, we have

$$\boldsymbol{A} = \begin{bmatrix} \hat{\boldsymbol{A}} & \vec{a} \\ \vec{c}^\top & \alpha_{nn} \end{bmatrix} \tag{4}$$

$$= \begin{bmatrix} \boldsymbol{I} & \vec{l} \\ \vec{0}^\top & 1 \end{bmatrix} \begin{bmatrix} \hat{\boldsymbol{A}} - \vec{l}\vec{c}^\top & \vec{0} \\ \vec{c}^\top & \alpha_{nn} \end{bmatrix}, \tag{5}$$

for $\vec{l} = \frac{\vec{a}}{\alpha_{11}}$. Assume, by induction, that $\hat{\boldsymbol{A}} - \vec{l}\vec{c}^\top = \boldsymbol{U}_1\boldsymbol{L}_1$ for some $n-1 \times n-1$ unit upper triangular matrix $\boldsymbol{U}_1$ and lower triangular matrix $\boldsymbol{L}_1$. Then we have

$$\begin{aligned}\boldsymbol{A} &= \begin{bmatrix} \boldsymbol{I} & \vec{l} \\ \vec{0}^\top & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{U}_1\boldsymbol{L}_1 & \vec{0} \\ \vec{c}^\top & \alpha_{nn} \end{bmatrix} \\ &= \begin{bmatrix} \boldsymbol{U}_1 & \vec{l} \\ \vec{0}^\top & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{L}_1 & \vec{0} \\ \vec{c}^\top & \alpha_{nn} \end{bmatrix} \\ &= \boldsymbol{U}\boldsymbol{L},\end{aligned}$$

where $\boldsymbol{U}$ is the unit upper triangular and $\boldsymbol{L}$ is lower triangular. This produces a recursive algorithm, which can be seen in algorithm 1.

Algorithm 1: A recursive algorithm to factorize $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{L}$ where $\boldsymbol{U}$ is unit upper triangular and $\boldsymbol{L}$ is lower triangular.

```
Data: $\boldsymbol{A}$
Result: $\boldsymbol{U}, \boldsymbol{L}$
1 $\vec{l} \leftarrow \vec{a}/\alpha_{11}$;
2 Factorize $\hat{\boldsymbol{A}} - \vec{l}\vec{c}^\top = \boldsymbol{U}_1\boldsymbol{L}_1$;
3 $\boldsymbol{U} \leftarrow \begin{bmatrix} \boldsymbol{U}_1 & \vec{l} \\ \vec{0}^\top & 1 \end{bmatrix}$;
4 $\boldsymbol{L} \leftarrow \begin{bmatrix} \boldsymbol{L}_1 & \vec{0} \\ \vec{c}^\top & \alpha_{nn} \end{bmatrix}$;
```

If we pre-allocate space for the entirety of $\boldsymbol{U}, \boldsymbol{L}$, this algorithm is tail-recursive, and can be de-recursed. As well, only the entries above the diagonal of $\boldsymbol{U}$ are modified, and only the entries on and below the diagonal of $\boldsymbol{L}$ are modified - so they can be stored in the same matrix. This leads to algorithm 2.

Algorithm 2: An iterative algorithm to factorize $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{L}$ where $\boldsymbol{U}$ is unit upper triangular and $\boldsymbol{L}$ is lower triangular.

```
Data: $\boldsymbol{A}$
Result: $\boldsymbol{UL}$ - a single matrix which stores the entries of $\boldsymbol{U}$ above the diagonal and the entries of $\boldsymbol{L}$ below
        the diagonal.
1 $\boldsymbol{UL} \leftarrow \boldsymbol{A}$;
2 for $i \leftarrow n$ to 2 do
3     $\boldsymbol{UL}[1:i-1,\ i] \leftarrow \boldsymbol{UL}[1:i-1,\ i]/\boldsymbol{UL}_{ii}$;
      /* Note that $\boldsymbol{UL}[1:i-1,\ 1:i-1]$ is unused for the solution so far - we can use
         this to store $\hat{\boldsymbol{A}} - \vec{l}\vec{c}^\top$.                                   */
4     $\boldsymbol{UL}[1:i-1,\ 1:i-1] \leftarrow \boldsymbol{UL}[1:i-1,\ 1:i-1] - \boldsymbol{UL}[1:i-1,\ i] \cdot \boldsymbol{UL}[i,\ 1:i-1]$;
5 end
```

Note that in the case where it is not necessary to preserve $\boldsymbol{A}$, we can use $\boldsymbol{UL} = \boldsymbol{A}$

(b) Based on your algorithm, prove/provide the necessary and sufficient determinant conditions for the existence of the UL decomposition.

Answer.

Theorem 1. *The following two statements are equivalent:*

i. There exist a unique unit upper triangular matrix $\boldsymbol{U}$ and nonsingular lower triangular matrix $\boldsymbol{L}$ such that $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{L}$.

ii. All trailing principal submatrices of $\boldsymbol{A}$ are nonsingular.

Proof.

$\Longrightarrow$ For a trailing submatrix $\boldsymbol{A}_{ii}$, we have

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{A}_{11} & \boldsymbol{A}_{1i} \\ \boldsymbol{A}_{i1} & \boldsymbol{A}_{ii} \end{bmatrix} = \begin{bmatrix} \boldsymbol{U}_{11} & \boldsymbol{U}_{1i} \\ 0 & \boldsymbol{U}_{ii} \end{bmatrix} \begin{bmatrix} \boldsymbol{L}_{11} & 0 \\ \boldsymbol{L}_{i1} & \boldsymbol{L}_{ii} \end{bmatrix} = \boldsymbol{U}\boldsymbol{L}.$$

Then $\det(\boldsymbol{A}_{ii}) = \det(\boldsymbol{U}_{ii}\boldsymbol{L}_{ii}) = \det(\boldsymbol{U}_{ii})\det(\boldsymbol{L}_{ii}) = 1 \cdot \prod_{k=1}^{i} (\boldsymbol{L}_{ii})_{kk} \neq 0$, since $\boldsymbol{U}$ is unit upper triangular and $\boldsymbol{L}$ is nonsingular lower triangular.

$\Longleftarrow$ Note that this is trivially true for a 1×1 matrix. Then, by induction, assume there exist a unique unit upper triangular matrix $\boldsymbol{U}$ and nonsingular lower triangular matrix $\boldsymbol{L}$ such that $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{L}$ for all $n-1 \times n-1$ matrices with all trailing principal submatrices nonsingular. For

$$\boldsymbol{A} = \begin{bmatrix} \hat{\boldsymbol{A}} & \vec{a} \\ \vec{c}^{\top} & \alpha_{nn} \end{bmatrix},$$

and

$$\hat{\boldsymbol{A}} = \boldsymbol{U}_1 \boldsymbol{L}_1,$$

which exist by assumption, we have

$$\boldsymbol{A} = \begin{bmatrix} \boldsymbol{U}_1 & \vec{l} \\ \vec{0}^{\top} & 1 \end{bmatrix} \begin{bmatrix} \boldsymbol{L}_1 & \vec{0} \\ \vec{c}^{\top} & \alpha_{nn} \end{bmatrix},$$

where $\vec{l} = \vec{a}/\alpha_{nn}$. Note, then that $0 \neq \det(\boldsymbol{A}) = \det(\boldsymbol{L}_1)\alpha_{nn}$, so $\alpha_{nn} \neq 0$, and the lower triangular matrix is nonsingular.

(c) Write a Matlab code to implement the UL decomposition and apply it to

$$\begin{bmatrix} 1 & 0 & 2 & 1 \\ -4 & 5 & 3 & -1 \\ -1 & 3 & 1 & 1 \\ 0 & 2 & 0 & 1 \end{bmatrix}$$

to verify that your code generates the required decomposition $\boldsymbol{A} = \boldsymbol{U}\boldsymbol{L}$.

Answer. The output for this matrix can be found below.

```
U=
  ⎡            ⎤
  ⎢ 1 -1  2  1 ⎥
  ⎢ 0  1  3 -1 ⎥
  ⎢ 0  0  1  1 ⎥
  ⎢ 0  0  0  1 ⎥
  ⎣            ⎦


L=
  ⎡            ⎤
  ⎢ 2  0  0  0 ⎥
  ⎢-1  4  0  0 ⎥
  ⎢-1  1  1  0 ⎥
  ⎢ 0  2  0  1 ⎥
  ⎣            ⎦
```

```
UL=

  ┌             ┐
  │  1  0  2  1 │
  │ -4  5  3 -1 │
  │ -1  3  1  1 │
  │  0  2  0  1 │
  └             ┘
```

The code used to implement the algorithm and verify the given test matrix can be found below.

```rust
use std::{
    error::Error,
    fs::{create_dir_all, File},
    io::Write,
};

use nalgebra::{Dim, Matrix4, OMatrix};

fn main() -> Result<(), Box<dyn Error>> {
    // fill A with given values, column major order
    let a = Matrix4::from_iterator([
        1.0, -4.0, -1.0, 0.0, 0.0, 5.0, 3.0, 2.0, 2.0, 3.0, 1.0, 0.0, 1.0, -1.0, 1.0, 1.0,
    ]);

    // compute UL decompositon
    let ul = ul_decomp(a);

    // extract the upper triangular part
    let mut upper = ul.upper_triangle();
    // enforce upper is unit triangular
    upper.fill_diagonal(1.0);
    // extract lower triangular part
    let lower = ul.lower_triangle();

    create_dir_all("./out/")?;
    let f = File::create("out/output5.txt")?;
    writeln!(&f, "U={upper}\nL={lower}\nUL={}", upper * lower)?;

    Ok(()) // :)
}

/// Computes UL decomposition in place, note input matrix is overridden
fn ul_decomp<R: Dim, C: Dim>(mut ul: OMatrix<f64, R, C>) -> OMatrix<f64, R, C>
where
    nalgebra::DefaultAllocator: nalgebra::allocator::Allocator<f64, R, C>,
{
    // rust is 0 indexed
    for i in (1..4).rev() {
        let alphaii = ul[(i, i)];
        let mut a_vec = ul.view_mut((0, i), (i, 1));
        a_vec /= alphaii;

        let lct = ul.view((0, i), (i, 1)) * ul.view((i, 0), (1, i));
        // start (row, col), size (nrows, ncols)
        let mut a_hat = ul.view_mut((0, 0), (i, i));
        a_hat -= lct;
    }
    ul
}
```

6. Even though we rarely need to compute the inverse of a matrix, let us think about it in this problem. Let $\boldsymbol{A} \in \mathbb{R}^{n\times n}$ be an invertible matrix. Describe an algorithm (based on the LU Decomposition/Gaussian Elimination) that computes $\boldsymbol{A}^{-1}$ with an operation count of $8n^3/3$ flops (ignoring the lower order terms).

Answer.

7. Inspired by the presentation in [Trefethen and Bau, SIAM Press, 1997], in this problem, we will numerically investigate the growth factor in LU with partial pivoting. In the class (see October 13th notes), we showed that the growth factor ρ_{pp} could be as large as $\rho_{pp} = 2n - 1$. Indeed we had found an example where this upper bound is attained. However, as we mentioned, the algorithm behaves much better in practice. Here, we will try LU with partial pivoting on random matrices with varying dimensions and plot the observations.

   Use the command `n = ceil(logspace(1, 3, 1000))` to create (approximately) logarithmically spaced matrix dimensions between 10 and 1000. Some of the dimensions will be repeated. The variable `n` is a vector of size 1000 with entries ranging from 10 to 1000. Then, for every entry `n(i)` of `n`, i.e., for $i = 1, 2, \dots, 1000$, create a random matrix $\boldsymbol{A}$ using `A= randn(n(i), n(i))/sqrt(n(i))`. So, we are creating a random matrix of varying dimensions with normally distributed entries having mean zero and standard deviation $\sqrt{n(i)}$. Then, compute the growth factor ρ_{pp} for every $\boldsymbol{A}$. At the end you will have a vector of size 1000 whose entries corresponding to the growth factor for every random $\boldsymbol{A}$. Using the `loglog` command (logarithmic scale both in the horizontal and 2 vertical axes), plot the growth factor vs the matrix dimensions `n`. On the same plot, by using the `hold on` command, plot the growth rate of $\sqrt{n}$. How is the observed/numerical growth behaving with respect to the theoretical upper bound 2^{n-1} and with respect to $\sqrt{n}$? Comment on your observations.

   Answer. The growth rate of the calculated ρ_{pp} v.s. $\sqrt{n}$ can be found in fig. 1. Note that $\sqrt{n}$ has been scaled to more easily observe the difference in growth rates. It can be seen that ρ_{pp} grows slightly faster than $\sqrt{n}$ on average, as expected (the average growth rate should be somewhere between $\sqrt{n}$ and $n^{3/4}$).

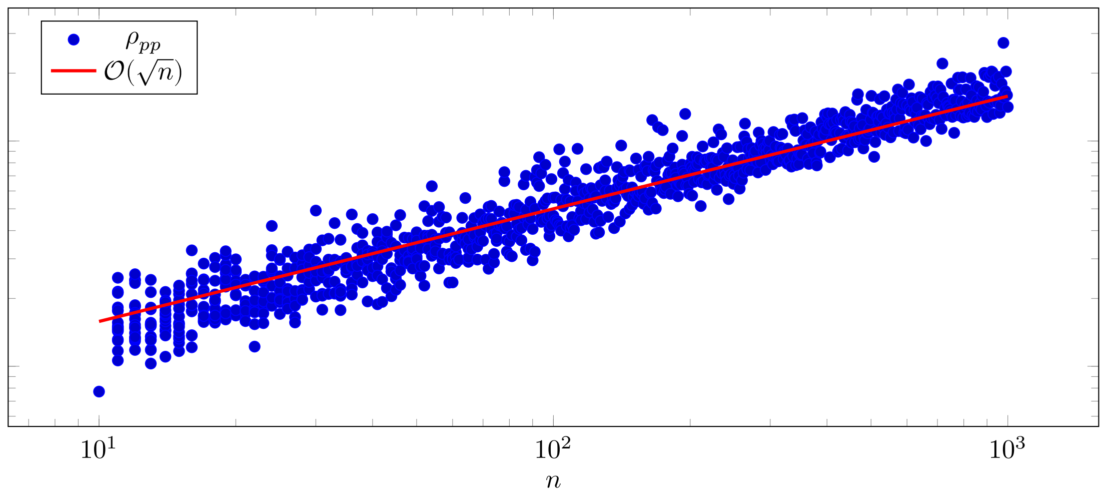

Figure 1: Growth rate of ρ_{pp} vs. $\sqrt{n}$.

The code used for this problem can be found below.

```rust
use indicatif::ProgressIterator;
use nalgebra::{DMatrix, LU};
use rand_distr::StandardNormal;
use std::{
    error::Error,
    fs::{create_dir_all, File},
    io::Write,
};

fn main() -> Result<(), Box<dyn Error>> {
    create_dir_all("./out/")?;
    let f = File::create("out/output7.txt")?;
    for n in ceil_logspace(1., 3., 1000).progress() {
        // A= randn(n, n)/sqrt(n)
        let a = DMatrix::<f64>::from_distribution(n, n, &StandardNormal, &mut rand::thread_rng())
            / (n as f64).sqrt();

        // $\|A\|_{max} = \max_{i,j} |a_{ij}|$
        let a_max = a.amax();
        let u_max = LU::new(a).u().amax();

        // $\rho_{pp} = \|U\|_{max} / \|A\|_{max}$
        let rho_pp = u_max / a_max;

        writeln!(&f, "{n} {rho_pp}")?;
    }

    Ok(()) // :)
}

/// Recreates Matlab ceil(logspace(a, b, n)) generates n points between decades 10^a and 10^b.
fn ceil_logspace(a: f64, b: f64, n: usize) -> impl ExactSizeIterator<Item = usize> {
    let temp = (b - a) / (n as f64 - 1.);
    (0..n).map(move |i| 10_f64.powf((i as f64) * temp + a).ceil() as usize)
}
```