18.330 Problem set 7 (spring 2020)

Submission deadline: 11:59pm on Tuesday, April 14

Exercise 1: LU factorization

Suppos you are given an upper-triangular $m \times m$ matrix U, i.e. all elements below the main diagonal are known to be 0. We wish to solve U**x** $=$ **b** for the vector **x**.

1. Find the analytical solution for the components x_i of **x** in terms of the $U_{i,j}$ and b_i using backsubstitution. (Hint: Where should you start?)
2. Write a function `backsubstitution(U, b)` that implements this.
3. What is the approximate operation count to perform backsubstitution?
4. Repeat [1]–[3] for forward substitution to solve L**x** $=$ **b** for a lower-triangular matrix L.
5. Write down what happens at a general step in the Gaussian elimination process for a matrix A, in terms of the old coefficients $A_{i,j}$ and the new coefficients $A'_{i,j}$.
6. Implement Gaussian elimination for an $m \times m$ square matrix A to calculate the LU factorization *without* pivoting. Your function `LU_factorization` should return L and U.
7. What is the operation count of this algorithm?
8. Write a function `solve_linear_system(A, b)` that uses the functions you have written to solve the linear system A**x** $=$ **b**.
9. Consider the matrix `A` built by the following Julia code. Please use exactly this code so that the results are consistent:

   ```
   using Random
   rng = MersenneTwister(1234)
   A = randn(rng, Float64, (10, 10))
   b = randn(rng, Float64, 10)
   ```

   Solve A**x** $=$ **b** for this `A` and `b`.

   Print out the full `x` vector using `jl show(stdout, "text/plain", x)` (This prevents the ellipses, `...`, that can occur in compressed output.)

   Compare this to the result from Julia's `\` (backslash) operator and comment.
10. Find the size of the residual A**x** $-$ **b** using the maximum norm (i.e. find the largest entry of the vector) for the `x` you found using your `solve_linear_system(A, b)` function.

11. Plot the residual as a function of m for matrices of sizes between $m = 10$ and $m = 1000$ (or however high your computer can manage) on suitable axes.

    How does the residual grow as the size grows?

    Note that Gaussian elimination is exact in exact arithmetic, so the residual must be due to round-off error.

Exercise 2: Exploiting structure in matrices 1 : Banded LU

In this exercise we will exploit **structure** in a matrix, i.e. the location of zero elements. If we know in advance that there are zeros in a certain part of the matrix ("structural zeros"), we can exploit that information to define *more efficient operations* on matrices of that type.

The simplest example is diagonal matrices, where we know that all off-diagonal elements are 0; thus calculations that would use those off-diagonal elements do *not need to be carried out*. (Note that there *may* also be zeros on the diagonal, but we don't know that in advance and can't use that information.)

We define a matrix A to be (p, q)**-banded** if its nonzero elements lie within p columns to the left of the main diagonal and q columns to the right:

$$A_{ij} \neq 0 \implies i - p \leq j \leq i + q$$

For example, a diagonal matrix is $(0, 0)$-banded (only the **main diagonal** is present) and a tridiagonal matrix is $(1, 1)$-banded.

As an example, think of a system of n particles in a line, each of which interacts only with its nearest neighbour on each side. This could lead to an interaction matrix that is known in advance to be tridiagonal, hence leading to the possibility of more efficiently solving the dynamics. Finite difference matrices are also banded. The matrices without structure that we discussed in the lectures on LU and QR are called **dense** matrices.

We will define a new type `Banded` as follows:

```
struct Banded{T} <: AbstractMatrix{T}
    bands::Matrix{T}
    p::Int
    q::Int
end
```

Here, `bands` is a matrix that stores only the **bands** (diagonals) of the original matrix, numbered $(-p, -(p-1), \ldots, 0, 1, \ldots q)$, where 0 corresponds to the main diagonal.

Note that there are more terms in `bands` than we need to store. We will store the lower bands justified to the bottom and the upper bands justified to the top. For example, consider the general (3, 2) banded matrix, i.e. with 3 diagonals below and 2 diagonals above the main diagonal.

$$B = \begin{bmatrix} b_{11} & b_{12} & b_{13} & & & \\ b_{21} & b_{22} & b_{23} & b_{24} & & \\ b_{31} & b_{32} & b_{33} & b_{34} & b_{35} & \\ b_{41} & b_{42} & b_{43} & b_{44} & b_{45} & b_{46} \\ & b_{52} & b_{53} & b_{54} & b_{55} & b_{56} \\ & & b_{63} & b_{64} & b_{65} & b_{66} \end{bmatrix}$$

This would be stored as follows in `bands`:

$$B = \begin{bmatrix} b_{41} & b_{31} & b_{21} & b_{11} & 0 & 0 \\ b_{52} & b_{42} & b_{32} & b_{22} & b_{12} & 0 \\ b_{63} & b_{53} & b_{43} & b_{33} & b_{23} & b_{13} \\ 0 & b_{64} & b_{54} & b_{44} & b_{34} & b_{24} \\ 0 & 0 & b_{65} & b_{55} & b_{45} & b_{35} \\ 0 & 0 & 0 & b_{66} & b_{56} & b_{46} \end{bmatrix}$$

The columns of the new matrix correspond to the diagonals of the old matrix.

Although this seems not to be more efficient in terms of storage, it will be for a larger matrix.

Note that when stored in this way the (i, j)th element in `B` is in band $(j - i)$ and the element is the jth element of that band. This means that `B[i, j] == B.bands[j, p + 1 + j - i]` or is 0 if the term is not in `bands`. You can use the code in `banded.jl` on the course website as the implementation for the `Banded` type. This should work just like an `Array` in Julia. You are encouraged to read and understand the code. Similar code would allow you to implement your own matrix types. This may be useful for your final projects!

1. Write a function `mymul(B::Banded, v::Vector)` that returns the matrix–vector product `B*v`. The function should be optimized in the sense that terms we know are 0 are not calculated. Compare the result to the built-in dense matrix multiply using the code `Array(B) * v`. (`Array(B)` produces a dense matrix with the same bands as `B`).

2. Consider the LU factorization. Show, using the formula for matrix multiplication, that if L is $(p, 0)$-banded and U is $(0, q)$-banded then A $=$ LU is (p, q)-banded.

3. Now show, by thinking about how the elimination algorithm works, that if A is (p, q)-banded then the factor L is $(p, 0)$-banded and U is $(0, q)$-banded.

4. We have seen that the structure of bandedness is maintained during an LU factorization. We saw in Exercise 1 that the operation count for a

dense LU factorization is $\mathcal{O}(n^3)$. What happens to the operation count when we exploit the banded structure of a (p, q)-banded matrix of size $m \times m$? Find the approximate operation count in terms of m, p and q.

5. Implement LU factorization for a banded matrix, `LU_factorization(B::Banded)`, making sure to operate only on non-zero entries.

6. Given the new operation count, is the naive back/forward substitution or the LU factorization the bottleneck when solving a banded system? If LU is no longer the bottleneck, add methods for your substitution functions so that they are specialized for banded matrices.

   (Hint for parts 5 & 6: if you fully understand what is happening here, modifying your LU / substitution codes from Exercise 1 should be simple for banded matrices, changing at most 1 or two lines of code.)

7. As an example of how necessary it is to exploit structure for efficiency, create a random tridiagonal matrix A of size $n \times n$ and a random vector **b** of length n, taking $n = 10,000$ (or whatever your computer can safely handle. How many times faster does your structured LU code run than if you were to use dense (standard) linear algebra (after you convert your banded matrix to a dense matrix)? (Remember to run your code once before timing it.)

8. Check numerically the dependence on m that you found in question [4] by solving random tridiagonal systems of sizes $m \times m$ for m between 10 and 10,000 and plotting the time taken as a function of m. (If the times are too fast, use larger values of m!)

   [You can use `@elapsed` (or `@belapsed` from `BenchmarkTools.jl`) to capture the time taken by a Julia operation, so that you can automate this in a loop.]

Exercise 3: Exploiting structure in matrices 2 : Tridiagonal QR

In this exercise we will see how to exploit structure (zeros again) to make a more efficient QR algorithm. (However, we do not ask you to code the algorithm this time.)

Consider a general tridiagonal matrix (i.e. a (1,1)-banded matrix):

$$T = \begin{bmatrix} b_1 & c_1 & & & & \\ a_2 & b_2 & c_2 & & & \\ & a_3 & b_3 & c_3 & & \\ & & \ddots & \ddots & \ddots & \\ & & & a_{N-1} & b_{N-1} & c_{N-1} \\ & & & & a_N & b_N \end{bmatrix}$$

Since there is only a single subdiagonal, it should be easier to make the matrix upper-triangular by operating on it.

1. Consider the following rotation matrix:

$$R(\theta) = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}$$

   How should you choose θ so that

$$R(\theta)\begin{bmatrix} p \\ q \end{bmatrix} = \begin{bmatrix} r \\ 0 \end{bmatrix}$$

   where $r^2 = p^2 + q^2$?

2. Consider the matrix

$$P_0 = \begin{bmatrix} R(\theta) & 0 \\ 0 & I_{N-2} \end{bmatrix}$$

   where I_N is the $N \times N$ identity matrix, and where θ is chosen as above for the terms $p = b_1$ and $q = a_2$. What does the resulting matrix $\tilde{A}_1 = P_0 A$ look like? Show that the first column is now upper-triangular.

   To be more concrete consider the matrix

$$M = \begin{bmatrix} 1 & 4 & 0 & 0 \\ 2 & 1 & 3 & 0 \\ 0 & 5 & 6 & 7 \\ 0 & 0 & 1 & 2 \end{bmatrix}.$$

   Show the resulting matrix when you multiply by P_0 – is it what you expected?

3. What matrix should you multiply $\tilde{A}_1$ by to make the second column upper-triangular? (Hint: it will have a similar structure to P_0) Check that the first column is still upper triangular. Call this matrix P_1. Multiply the result of $P_0 M$ by P_1 and show the result.

4. Generalize to the rotation matrix that makes the nth column of a tridiagonal matrix upper-triangular. Call this matrix P_{n-1}. Show that P_{n-1} is orthogonal, i.e. $P_n P_n^\top = I$.

5. We can now write a tridiagonal matrix T in upper-triangular form by forming the product $P_{N-2} P_{N-3} \cdots P_1 P_0 T$. From the above we see that the result of this will be an upper-triangular matrix R.

   Show that the product of orthogonal matrices is orthogonal, and hence that this procedure gives a QR decomposition of T.

6. What is the approximate operation count for a full QR factorization on a dense matrix A?

7. What is the approximate operation count for this reduced tridiagonal QR?
8. **Extra credit** Implement this algorithm in an efficient way.