18.065 Pset 1

Due Friday 2/17 at 1pm. Submit in PDF format: a decent-quality scan/image of any handwritten solutions (e.g. get a scanner app on your phone or use a tablet), and a PDF printout of your Jupyter notebook showing your code and (clearly labeled) results.

Problem 1 (4+4+4+4+4 points)

Recall from class that multiplying an $m \times p$ by a $p \times n$ matrix costs mnp scalar multiplications (and a similar number of additions) by the standard (practical) algorithms.

Matrix multiplication is **not commutative** ($AB \neq BA$ in general), but it **is associative**: $(AB)C = A(BC)$. It turns out that where you put the parentheses (i.e. in *what order* you do the multiplications) can make a *huge* difference in computational cost.

(a) If $x \in \mathbb{R}^n$ and A, B are $n \times n$ matrices, compare the scalar multiplication counts of $(AB)x$ vs. $A(Bx)$, i.e. if we do the multiplications in the order indicated by the parentheses.

(b) If $x, b \in \mathbb{R}^n$, **how many scalar multiplications** does the computation

$$p = (I - (xx^T)/(x^T x))b$$

take if we *do it in the order indicated by the parentheses*? (Note that dividing by a scalar α is equivalent to multiplying by α^{-1} at the negligible cost of one scalar division.)

(c) Explain how to compute the *same* p as in part (b) using as *few multiplications as possible*. Outline the sequence of computational steps, and give the count of multiplications.

(d) $p^T x =$ what?

(e) Implement your algorithm from (c) in Julia, filling in the code below, and time it for $n = 1000$ using the `@btime` macro from the BenchmarkTools package, along with the algorithm from part (b), following the outline below. How does the ratio of the two times compare to your ratio of multiplication counts?

In [32]:
```julia
using LinearAlgebra, BenchmarkTools

# algorithm from part (b)
function part_b(x, b)
    return (I - (x*x')*(x'*x)^-1) * b
end

# algorithm from part (c)
function part_c(x, b)
    return b - (x*((x'*(x'*x)^-1)*b))
end
```

```julia
# test and benchmark on random vectors:
n = 1000
x, b = rand(n), rand(n)

# test it first — should give same answer up to roundoff error
if part_c(x, b) ≈ part_b(x, b)
    println("Hooray, part (c) and part (b) agree!")
else
    error("You made a mistake: part (c) and part (b) do not agree!")
end

# benchmark it:

println("\npart (b): ")
@btime part_b($x, $b);

println("\npart (c): ")
@btime part_c($x, $b);
```

```
Hooray, part (c) and part (b) agree!

part (b):
  5.651 ms (7 allocations: 22.90 MiB)

part (c):
  2.044 μs (3 allocations: 23.81 KiB)
```

Problem 2 (8+4 points)

(a) Describe the four fundamental subspaces of the **rank-1 matrix** $A = uv^T$ where $u \in \mathbb{R}^m$ and $n \in \mathbb{R}^n$.

(b) For any column vectors $u, v \in \mathbb{R}^3$, the matrix uv^T is rank 1, except when _______, in which case uv^T has rank ____.

Problem 3 (5+4+4+4 points)

(a) Pick the choices that makes this statement correct for arbitrary matrices A and B: $C(AB)$ (*contains / is contained in*) the column space of (*A / B*). Briefly justify your answer.

(b) Suppose that A is a 1000×1000 matrix of rank < 10. Suppose we multiply it by 10 random vectors $x_1, x_2, \ldots, x_{10}$, e.g. generated by `randn(1000)`. How could we use the results to get a 10×10 matrix C whose rank (almost certainly) matches A's?

(c) Suppose we instead make 1000×10 matrix X whose columns are $x_1, x_2, \ldots, x_{10}$. Give a formula for the *same* matrix C in terms of matrix products involving A and X.

(d) Fill in the code for C below, and compare the biggest 10 singular values of A (chosen to be rank ≈ 4 in this case) to the corresponding 10 singular values of C. Does it match what you expect?

In [30]:

```julia
using LinearAlgebra

# random 1000x1000 matrix of rank 4
A = randn(1000, 4) * randn(4, 1000)
@show svdvals(A)[1:10]

X = randn(1000, 10)
C = X' * A * X
@show svdvals(C)
```

```
(svdvals(A))[1:10] = [1033.437033249335, 1011.3839299777054, 967.3012768531058, 917.088555589723, 1.0221088602277808e-12, 8.667944633290469e-13, 7.802321942478929e-13, 7.392352479625353e-13, 7.36152674855863e-13, 7.208168891076064e-13]
svdvals(C) = [18923.711378553286, 10128.014679660939, 5240.844393768108, 4083.3439333669785, 3.5949835513878783e-12, 3.0672795174495276e-12, 2.5772864697684613e-12, 2.1910666302997e-12, 7.131643512506856e-13, 2.910189199817205e-13]
```

Out[30]:

```
10-element Vector{Float64}:
 18923.711378553286
 10128.014679660939
  5240.844393768108
  4083.3439333669785
     3.5949835513878783e-12
     3.0672795174495276e-12
     2.5772864697684613e-12
     2.1910666302997e-12
     7.131643512506856e-13
     2.910189199817205e-13
```

Problem 4 (4+5+5 points)

The famous Hadamard matrices are filled with ±1 and have orthogonal columns (orthonormal if we divide H_n by $1/\sqrt{n}$). The first few are:

$$H_1 = (\,1\,)\,, \tag{1}$$

$$H_2 = \begin{pmatrix} 1 & 1 \\ 1 & -1 \end{pmatrix}, \tag{2}$$

$$H_4 = \begin{pmatrix} H_2 & H_2 \\ H_2 & -H_2 \end{pmatrix} = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & -1 & 1 & -1 \\ 1 & 1 & -1 & -1 \\ 1 & -1 & -1 & 1 \end{pmatrix}. \tag{3}$$

Notice that (for power-of-2 sizes), they are built up "recursively" out of smaller Hadamard matrices. Multiplying a vector by a Hadamard matrix requires no multiplications at all, only additions/subtractions.

(a) If you multiply $H_4 x$ for some $x \in \mathbb{R}^4$ by the normal "rows-times-columns" method (*without* exploiting any special patterns), exactly how many scalar additions/subtractions are required?

(b) Let's break x into two blocks: $x = \begin{pmatrix} x_1 \\ x_2 \end{pmatrix}$ for $x_1, x_2 \in \mathbb{R}^2$. Write out $H_4 x$ in terms of a sequence of 2×2 block multiplications with $\pm H_2$. You'll notice that some of these 2×2 multiplications are repeated. If we re-use these repeated multiplications rather than doing them twice, we can save a bunch of arithmetic — what is the new count of scalar additions/subtractions if you do this?

(c) Similarly, the 8×8 Hadamard matrix $H_8 = \begin{pmatrix} H_4 & H_4 \\ H_4 & -H_4 \end{pmatrix}$ is made out of H_4 matrices. To multiply it by a vector $y \in \mathbb{R}^8$, the naive rows-times columns method would require ____ scalar additions/subtractions, whereas if you broke them up first into blocks of 4, used your solution from (b), and then re-used any repeated H_4 products, it would only require ____ scalar additions/subtractions.

Problem 5 (5+5 points)

The famous "discrete Fourier transform" matrix F has columns that are actually eigenvectors of the (unitary) permutation matrix:

$$P = \begin{pmatrix} & 1 & & \\ & & 1 & \\ & & & 1 \\ 1 & & & \end{pmatrix}$$

for the 4×4 case, and similarly for larger matrices.

(a) One way of saying way Fourier transforms are practically important is that they *diagonalize* (are eigenvectors of) matrices that *commute* with P. If A is a 4×4 matrix whose first row is $(a \, b \, c \, d)$

$$A = \begin{pmatrix} a & b & c & d \\ ? & ? & ? & ? \\ ? & ? & ? & ? \\ ? & ? & ? & ? \end{pmatrix}$$

that commutes with P (i.e. $AP = PA$), what must be true of the other ("?") entries of A?

(b) Fill in the matrix `A` in Julia below and fill in and run the code to check that it commutes with P and is diagonalized by F:

In [35]:
```
a, b, c, d = 1, 7, 3, 2    # 4 arbitrarily chosen values

P = [0 1 0 0
     0 0 1 0
     0 0 0 1
     1 0 0 0]
```

```julia
F = im .^ ((0:3) .* (0:3)') # the 4×4 Fourier matrix

# fill in:
A = [a b c d
     d a b c
     c d a b
     b c d a]
```

Out[35]:
```
4×4 Matrix{Int64}:
 1  7  3  2
 2  1  7  3
 3  2  1  7
 7  3  2  1
```

In [36]:
```julia
# check:
P * A == A * P
```

Out[36]:
```
true
```

In [37]:
```julia
# check that F diagonalizes A.  (How?)

@show F'*A*F
```

```
F' * A * F = Complex{Int64}[52 + 0im 0 + 0im 0 + 0im 0 + 0im; 0 + 0im -8 + 20im 0
+ 0im 0 + 0im; 0 + 0im 0 + 0im -20 + 0im 0 + 0im; 0 + 0im 0 + 0im 0 + 0im -8 - 20i
m]
```

Out[37]:
```
4×4 Matrix{Complex{Int64}}:
 52+0im   0+0im     0+0im   0+0im
  0+0im  -8+20im    0+0im   0+0im
  0+0im   0+0im   -20+0im   0+0im
  0+0im   0+0im     0+0im  -8-20im
```