Homework problems that will be graded (P1 - P6, 30pts in total):

**P1.** This exercise is meant to help you understand and illustrate how block-matrix multiplication works. Let $A$ and $B$ be the two matrices defined below:

$$A = \begin{pmatrix} 1 & -1 & 0 \\ 3 & -2 & -1 \\ 0 & 2 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 0 & 1 \\ 2 & -1 \\ -1 & 2 \end{pmatrix}.$$

Compute the product $AB$ the “traditional” way, and then again by using block-matrix multiplication with the partition

$$A = \begin{pmatrix} 1 & -1 & 0 \\ 3 & -2 & -1 \\ 0 & 2 & 0 \end{pmatrix}, \quad B = \begin{pmatrix} 0 & 1 \\ 2 & -1 \\ -1 & 2 \end{pmatrix}.$$

Use the formula given in lecture (or the posted notes, or the textbook). *Show your work.*

**P2.** (This is similar to Exercise 1.2.10 from the textbook)

Consider the simple cart-spring system shown in the picture below. In the case of each little cart, a steady force is applied that pushes it to the right by the number of Newtons shown in the picture. The stiffness constants of the springs are also shown (1 N/m for the first and last, and 2 N/m for the two middle ones).

Initially, the system was in equilibrium. Once the forces were applied, a new equilibrium was reached, and each cart was displaced by some number of meters. Let the cart displacements (the difference between the initial and the final position of the cart) be denoted by $x_1$, $x_2$, and $x_3$ (from left to right). Using the balancing of forces and Hooke’s law (stiffness times displacement equals force) write out and solve a system of equations that can be used to find $x_1$, $x_2$, $x_3$.

**Hint:** For example, given the first cart, the force pulling to the left is given by the first spring, so it’s $1 \cdot x_1$, whereas the forces pulling to the right are two-fold: from the second spring we get $2 \cdot (x_2 - x_1)$, and we also have the constant force of 1 N being applied. Therefore the equation we get for it is $x_1 = 2(x_2 - x_1) + 1$. 

1
**P3.** Set up (do not solve!) a linear equation to numerically solve the following ODE:

\[ u''(x) - u(x) = 1 , \quad \text{for all } x \in (0, 1) , \quad \text{with } u(0) = u(1) = 0 , \]

using a partition of \([0, 1]\) with \(m = 5\) (and, correspondingly, \(h = 1/5\)). You will need to use the approximation formula

\[ u''(x) \approx \frac{u(x + h) - 2u(x) + u(x - h)}{h^2} . \]

**P4.** Prove that if \(L\) is an invertible lower triangular matrix, then also \(L^{-1}\) is lower triangular.

*Hint:* There are many ways to prove this. One possibility is to note that the \(j\)-th column \(c_j\) of \(L^{-1}\) satisfies

\[ L c_j = e_j, \quad j = 1, \ldots, n \]

where \(n\) is the size of the matrix and \(e_j\) is the \(j\)-th unit vector. Now use forward substitution as we did in class.

**P5.** (This is Exercise 1.3.15 from the textbook)

Consider a system

\[ U x = b , \]

where \(U\) is upper triangular.

a) Modify the solution we developed in class for lower triangular systems to upper triangular systems (in the upper triangular case, this process is called back substitution).

b) Write a MATLAB code that solves upper triangular systems.

**P6.** Using basic programming (for loops, while loops, and if statements), write two MATLAB functions that both take as an input

- \(n \times n\) matrix \(A\),
- \(n \times n\) matrix \(B\),
- \(n \times 1\) matrix \(x\).

Have the first function compute \(ABx\) through \((AB)x\) and the second through \(A(Bx)\). Have both functions output the number of flops used. Then

(a) print out and hand in the first function

(b) print out and hand in the second function

(c) apply both algorithms for \(n = 100, 200, 400, 800\). Which approach is better?
Additional problems, which will not be graded (A1 - A4):

**A1.** (This is Exercise 1.1.25 from the textbook)
Make a partition of the matrix-vector product $Ax = b$ that demonstrates that $b$ is a linear combination of the columns of $A$.

**A2.** Prove that if $A^{-1}$ exists, then there can be no nonzero $y$ for which $Ay = 0$.

**A3.** (This is Exercise 1.3.14 (a) from the textbook.)
Consider the following pseudocode algorithm (given in equation 1.3.13):

```plaintext
for $j = 1, \ldots, n$
    if $g_{jj} = 0$, set error flag (G is singular), exit
    $b_j \leftarrow b_j / g_{jj}$ (this is $y_j$)
    for $i = j + 1, \ldots, n$ (not executed when $j = n$)
        $b_i \leftarrow b_i - g_{ij} b_j$
    end
end
```

Count the operation in the algorithm above. Note that the flop count is identical to that of the row-oriented forward substitution algorithm (1.3.5).

**A4.** Explain why, for a given input, an algorithm of order $O(n^2)$ may not be faster than an algorithm of order $O(n^3)$, even if they are calculating the same thing. (Note that, for $n$ large enough, the former algorithm will eventually be faster.)