# Challenge #4: Gauss-Jordan elimination

Bachelor in Informatics and Computing Engineering

Programming Fundamentals

Instance: 2022/2023

# Introduction

The challenge is extra work planned for more advanced Python programmers that easily solve the regular exercises.

Advice: Do not look for a solution before trying

Reference: https://en.wikibooks.org/wiki/Linear_Algebra/Gauss%27_Method

# Solving systems of linear equations

A system of $n$ linear equations on $n$ variables can be represented by a matrix of $n \times (n+1)$ coefficients; for example, the system

$$\begin{cases} x & +y & & = 0 \\ 2x & -y & +3z & = 3 \\ x & -2y & -z & = 3 \end{cases}$$

can be represented by the $3 \times 4$ matrix

$$M = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 2 & -1 & 3 & 3 \\ 1 & -2 & -1 & 3 \end{bmatrix}$$

Each row represents the coefficients of the variables on the left-hand side plus the constant term on the right-hand side.

A solution for the system is a vector of values for the variables that simultaneously make all equations hold. A system of $n$ equations on $n$ variables may be *determinate* (single solution), *indeterminate* (infinite solutions) or *impossible* (no solution).

If the system is determinate we can find its solution using an algorithm known as *Gaussian elimination*:

1. first we transform the system in an *upper triangular* form by swapping rows, adding rows and multiplying rows by non-zero constants;
2. we then solve the triangular system by *backwards substitution.*

Applying step 1 to the matrix $M$ above we get

$$M' = \begin{bmatrix} 1 & 1 & 0 & 0 \\ 0 & -3 & 3 & 3 \\ 0 & 0 & -4 & 0 \end{bmatrix}$$

Applying step 2 to the matrix $M'$ we get

$$\begin{aligned} z &= 0/(-4) && = 0 \\ y &= (3 - 3 \times z)/(-3) && = -1 \\ x &= (0 - 0 \times z - 1 \times y)/1 && = 1 \end{aligned}$$

This (single) solution of the system corresponds to the vector $(1, -1, 0)$.

## Objective

Write a Python function `gauss(matrix)` that solves a system of linear equations using this method. The argument should be a matrix (i.e. list of lists of numbers) with dimensions $n \times (n + 1)$. The result should be a solution vector (i.e. list of numbers) with dimension $n$ if the system is determinate or a `ValueError` exception otherwise.

## Suggestion

It may be helpful to split the problem into two auxiliary functions:

**`upper_triangular(matrix)`** transforms a matrix for a system into an equivalent one in upper triangular form.

**`solve_triangular(matrix)`** receives a matrix that is in upper triangular form and returns the solution vector by backwards substitution.

## The end