
# Operations Research Coding Lab (OR-LAB)

Winter Semester 2022/23

Technische Universität Berlin
Faculty VII (Economics and Management)
Workgroup for Infrastructure Policy (WIP)

# Homework 1

*You can find two templates for this homework on the isis platform. Download the whole directory and open it in Visual Studio Code to work on the subsequent tasks.*
*To pass the homework, you must again submit the whole directory with working code via the provided tool on the isis platform.*

Task 1 to 4 should be done in *template01.jl* and the last task should be done in *template02.jl*.

## Task 1.1

Create a variable of the name and type:

- *intvar*, Int64
- *floatvar*, Float64
- *stringvar*, String
- *symbolvar*, Symbol
- *boolvar*, Bool

### Task 1.1.1

Create two variables of type *String* with the content:

- Hello
- World!

### Task 1.1.2

Concatenate these two variables with a space in between. *Hint: Take a look at the official Julia doc if you do not know how to do it* (`https://docs.julialang.org/en/v1/manual/strings/#man-concatenation`).

### Task 1.1.3

Use string interpolation to create a sentence using at least three of the variables created in task 1! *Hint: Take a look at* `https://docs.julialang.org/en/v1/manual/strings/#string-interpolation`.
Assign this sentence as a String to a variable called *mymeaningfulsentence*.

### Task 1.1.4

Print the variable *mymeaningfulsentence* to the REPL!

## Task 1.2

### Task 1.2.1

Create an array with the size 100x100 that contains only zeros of type Float64. Assign it to the variable name *mymatrix*.

### Task 1.2.2

Use the *fill!* function to replace the values of *mymatrix* with the value 1.

### Task 1.2.3

Sum up all values of *mymatrix* and print the result to the REPL.

### Task 1.2.4

Select only the last row of *mymatrix* and print it to the REPL.
*Hint: Indexing in Julia is row first, column second. You can use the colon ":" operator to select every position in a dimension of an array.If you have troube with indexing of arrays, take a look at* `https://juliabook.chkwon.net/book/basics#cid15`.

#### Task 1.2.5

Create a vector of the length 100 which contains the numbers 2 to 200 in steps of 2.

#### Task 1.2.6

Replace the last row of *mymatrix* with the vector created in task 3.4 and print *mymatrix* to the REPL.

#### Task 1.2.7

Create another vector of the length 100 containing random numbers and use the function *hcat* to concatenate this vector to *mymatrix*. This new array should be assigned to the variable name *mergedarray*.

#### Task 1.2.8

Print the size of *mergedarray* to the REPL.

#### Task 1.2.9

Sum over the first dimension of *mergedarray* and print the result to the REPL.

## Task 1.3

#### Task 1.3.1

Create a dictionary which takes the type Int64 as keys and Float64 as values. Name it *mydict. Hint: You can force element type of dictionaries using "DictT1,T2()" where T1 and T2 are the respective types you want to specify.*

#### Task 1.3.2

Add the an entry with key 1 and value 10.

#### Task 1.3.3

Iterate from 1 to 5 using a for-loop and assign the iteration counter as key with a random number as value. Afterwards, print the dictionary to the REPL.

**Task 1.3.4**

Create a Vector that containts all values from the dictionary which are greater than 0.5.

## Task 1.4 Gradient Descent on a Convex Quadratic Function

You are given a function of the following type:

$$f(x) = \frac{1}{2}x^T Q x + q^T x \tag{1.1}$$

where $x$ is the vector of decision variables, i.e. $x \in \mathbb{R}^n$,
and $q, Q$ are parameters, i.e. $q \in \mathbb{R}^n, Q \in \mathbb{R}^{n \times n}$.
We can try to find a minimum for this function by always moving in a direction of descent. If the matrix $Q$ is symmetric positive definite, repeatedly moving in a direction of descent will eventually yield a global minimum of the function $f(x)$. A direction of descent for symmetric $Q$ can analytically be determined by computing the gradient:

$$\nabla f(x) = Qx + q \tag{1.2}$$

Starting with an initial guess for x and using an appropriate step size $\alpha \in \mathbb{R}$, we can compute new iterates for our decision variables (which have a lower function value):

$$x_{k+1} = x_k - \alpha \nabla f(x) = x_k - \alpha(Qx + q) \tag{1.3}$$

i) *Write a function that implements this behavior. The function takes the following arguments:*

- *A symmetric parameter matrix $Q \in \mathbb{R}^{n \times n}$ and a parameter vector $q \in \mathbb{R}^n$*
- *An initial guess for the decision variables $x \in \mathbb{R}^n$*
- *A scalar value denoting the step size $\alpha \in \mathbb{R}$ and the number of iterations*

Use the code provided (file: *template02.jl*) and test your function. After each iteration, use the function *print_ objective_ f*. You can find documentation by hovering over the function name (or in help mode) after running the first three lines of code.

## Homework 2

A template file that containts the core structure is available. Additionally, a file *tests.jl* with four different opimization problems defined as tests is provided. To make sure that everything is set up correctly, you can open the *tests.jl* and the file as is. The first problemset should be able to pass. The other three tests will fail with the default template. In the following tasks we will expand the functionality so that in the end all tests should be able to pass.

## Task 2.1 Check if problem is unbounded

The Primal Simplex algorithm is defined as follows:

Initialize: Valid basic solution

(1) All values in $f_j > 0$? If yes $\rightarrow$ Optimal solution found! Algorithm terminates.

(2) Take the minimal value in the $f_j$ row $\rightarrow j$ becomes the pivot column

(3) All $a_ij \leq 0$ for pivot column $j$? If yes $\rightarrow$ problem is unbounded. Algorithm terminates.

(4) Take $min\frac{b_i}{a_{ij}}$ if $a_{ij}$ is positiv $\rightarrow i$ becomes the pivot row.

(5) $x_j$ becomes basic variable, $x_i$ becomes non basic variable: Row vector of the basic variables have to become unit vectors. Next iteration.

In the lecture we already implemented the Primal Simplex. However, we skipped step 3 of the algorithm descriped above. Your task is to implement this check in the given template. In the template file there are hints where to modify the existing code.
If the problem is unbounded the respective pivot column shoudl be printed and an error should be thrown that says *Every value is lower or equal zero. Problem Unbounded.* Use the *error* function that is automatically provided by Julia (`https://docs.julialang.org/en/v1/base/base/#Base.error`).

After successfully implementing the task, the test for *Tableau 3* should pass.

## Task 2.2 Dual Simplex

The Dual Simplex algorithm is defined as follows:

Initialize: No valid basic solution

(1) All values in $b_i >= 0$ ? If yes $\rightarrow$ Current solution is a basic solution! Move on the Primal Simplex.

(2) Take the minimal negative value in the $b_i$ row $\rightarrow i$ becomes the pivot row

(3) All $a_ij \geq 0$ for pivot row $i$? If yes $\rightarrow$ problem is infeasable. Algorithm terminates.

(4) Take $max\frac{f_j}{a_{ij}}$ if $a_{ij}$ is negative $\rightarrow j$ becomes the pivot column.

(5) $x_j$ becomes basic variable, $x_i$ becomes non basic variable: Row vector of the basic variables have to become unit vectors. Next iteration.

### Task 2.2.1

Your task is to implement the Dual Simplex algorithm. Fortunately, a lot of functions already exists since the operations are the same in the Primal Simplex. Only one function called *get_ dual_ pivot_ element* should be added. Take a look in the template where you will find some hints how to implement that function.
Once you completed that task, the test for *Tableau 2* should pass.

### Task 2.2.2

In a similar manner as in task 1 your function should throw an error if the problem is infeasble. In that case, the values of the pivot row should be printed and an error message should be thrown saying *Every value is greater or equal zero. Problem infeasable.*

Finally, also test *Tableau 4* should pass.