JUSTUS-LIEBIG-UNIVERSITÄT GIESSEN

FACHBEREICH 06
Psychologie und Sportwissenschaft

Soviet Style Weightlifting Program Generator App

Lecturer:

Lea Junge-Bornholt (M.Sc)

Philipp Ewen,

Gutenbergstraße 21,

35390 Gießen

E-mail: philipp.ewen@sport.uni-giessen.de

Matriculationnumber: 8102119,

M.Sc. Human Movement Analytics: Biomechanics, Motor Control and Learning

Table of Contents

Prephrase:

The author distances himself from any political messaging. Any political symbols or regional phrases used are solely for comedic effect.

1. Introduction

This project is based the book *Fundamentals of the Soviet System: The Soviet Weightlifting System and Modern Applications of the Sport of Weightlifting* by Gwendolyn Sisto and Ivan Rojas (2017). The authors provide valuable insights into the training methodologies that produced world-class weightlifters with long careers. This system developed hundreds of athletes who dominated the sport for decades, winning Olympic gold medals, championship titles, and setting world records.

The Soviet system, developed during the mid-20th century, has profoundly influenced strength training methodologies worldwide. Experts like Yessis (1987) have played a pivotal role in translating and disseminating these techniques, making them accessible to a global audience. The concept of supercompensation, introduced by Yakovlev (1955), is central to understanding the effectiveness of these training methods. It describes the body's process of adapting to training stress by temporarily increasing performance capacity beyond the original level, provided that recovery periods are appropriately managed. Periodization, another cornerstone of Soviet training regimens, involves systematically varying training variables to optimize performance gains and prevent overtraining. Developed by Matveyev (1964), periodization has become a standard approach in athletic training worldwide.

This application and its accompanying template are designed for individuals with a solid foundation in weightlifting techniques, specifically the snatch, clean and jerk, and squat. It's essential that users can consistently perform these movements with proper form. Beginners or those with less than 2-3 years of experience should focus on technique development and may benefit from more adaptable training programs that don't heavily rely on percentage-based loads. Additionally, users should have the knowledge to intelligently incorporate accessory exercises tailored to their specific goals and address individual weaknesses.

2. User Manual

2.1. Opening window

After starting the app, the user is presented with a opening window. Here, they can either create a new training program by pressing the "Generate New Weightlifting Program" button or load an existing one by selecting "Load Existing Weightlifting Program." If the user does not yet have a program, they should not be able to load one. As a design choice, the application incorporates some comedic elements. If the user attempts to load a program without having one, an error message appears stating, "You have no program yet, comrade... put the vodka aside and generate a new program." After this message, the "Load Existing Weightlifting Program" button is disabled, ensuring that the user can only generate a new program. If a program already exists, the user is directed to the program viewer tab, where they can view and interact with their training plan. If they choose to generate a new program, they are redirected to the Program Generator.

2.2. Program generator

If the user wants to generate a new program, they are first directed to the "Program Generator" tab. There, they can enter their max weights for the Snatch, Clean & Jerk, Back Squat, and Front Squat. These are the most relevant exercises for weightlifting progress and are used to calculate the training weights for each session. The user can also select the type of program they want to follow. There are two main program types a weightlifter can use to progress:

1. A strength-based squat program – focused on building maximal leg strength and adding muscle mass through bodybuilding-style accessory work.

2. A weightlifting-focused program – centered around the Olympic lifts, allowing the user to improve their technique and performance without overly taxing the legs, so they stay fresh for the main lifts.

Squat programs last 6 weeks, while weightlifting programs run for 12 weeks. The user can also choose whether they want to train 3 or 4 days per week. Each configuration generates a tailored program based on the athlete's goals and needs. To generate the program, the user simply sets their preferences and clicks “Generate Program, Comrade.” Once completed, the program is created, and the user is redirected to the "Program Viewer" tab.

2.3. Program viewer

On the Program Viewer tab, the user can see their current training day, including all main exercises with the corresponding sets, reps, and weights. The maximum number of sets is limited to 10, and only the main lifts include specific weight, set, and rep prescriptions. There are also two text fields at the top of the view that display the current training week and day (e.g., "Week 3 – Day 2"), helping the user keep track of their progress in the program. Accessory work is intentionally left open. The athlete is expected to have a solid understanding of intelligent training and should be able to choose accessories based on their individual weaknesses. This is common among intermediate-level weightlifters—most of them are training nerds when it comes to optimizing their routines. Additionally, there is a table displaying the most important training variables for a structured program. These include:

- Volume (tonnage), calculated as weight × reps
- Total reps
- Relative intensity (Σ(Weights × Reps) / Reps) / 1RM × 100

These metrics give the athlete a clear picture of the workload for each exercise and the session as a whole. This helps them gauge whether it's a heavier or lighter training day and prepare accordingly. There is also an option to save and download the current training day as a CSV file by clicking the "Save TP for Current Day" button. This allows the athlete to keep their training plan accessible during workouts. After completing a session, the user can press the "Day Done" button. A pop-up message saying "Great work, Comrade" appears, and the next day's session—with updated training variables—is automatically loaded.

2.4. Recordboard

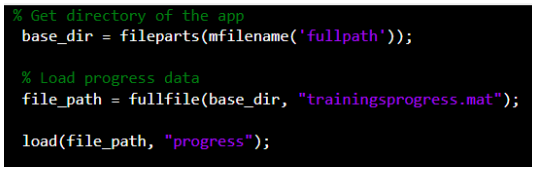
There is also an option to navigate to a Recordboard by clicking the "Recordboard" button. In this tab, the user can track their progress on a graph titled "1 RM Progress Timeline." Here, users can update their current one-rep max (1RM) values by entering new maximum weights into the respective numeric input fields and clicking the "Update Maxes" button. This action updates both the stored max values and all relevant percentage-based calculations in the Program Viewer and the 1 RM Progress Timeline. Additionally, a visual maximum graph is shown. If the newly entered max exceeds the previous value, a congratulatory pop-up message appears: "Awesome, a new 1 RM! Now back to work, Comrade!" From the Recordboard, the user can also generate a new program if needed, by clicking the "Make New Program" button, which redirects to the Program Generator tab.

To return to the current program, the user can simply click the “Back to Program Viewer” button. With all these functionalities, the Soviet-Style Weightlifting Program Generator App is designed to fulfill the core needs of an intermediate weightlifting athlete. It offers a structured Soviet-style program, enables progress tracking through essential training metrics, and supports long-term performance development through an intuitive and efficient interface.

3. Code Explanation and Decisionmaking

3.1. Loading and saving data

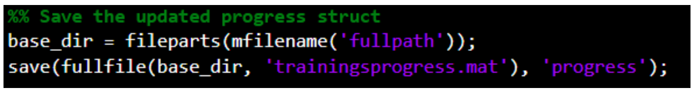
Before delving into the specifics of the code, it's essential to understand how data loading and saving are managed in this project. To efficiently preserve user data—ensuring that personal records remain accessible whether loading an existing program or initiating a new one—a MATLAB file named "Trainingsprogress.mat" is utilized. This file stores critical information such as current one-repetition maximums (1RMs), program type, training day within the week, and the history of 1RMs. Notably, the "Trainingsprogress.mat" file is created once and maintains historical data regardless of new program initiations; only the program type, day, week, and current 1RMs are updated. All data is encapsulated within a structure named "progress," which is consistently referenced throughout the project's major components. Understanding this data management approach is crucial, as it ensures seamless functionality across different systems and sessions.



```
% Get directory of the app
 base_dir = fileparts(mfilename('fullpath'));

 % Load progress data
 file_path = fullfile(base_dir, "trainingsprogress.mat");

 load(file_path, "progress");
```

Image 1: loading data

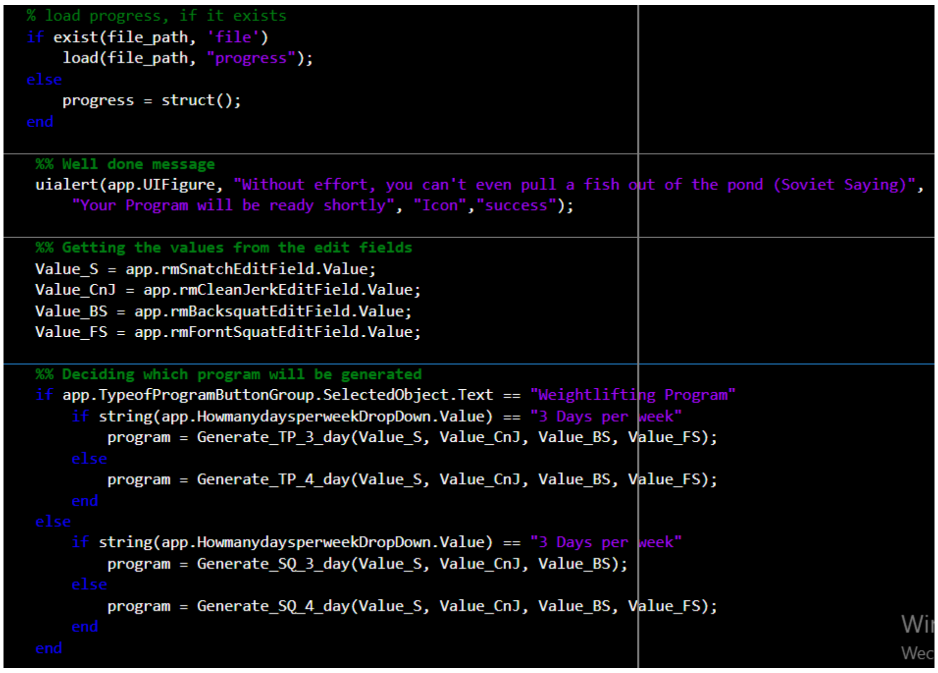
In this project, instead of using the commonly used pwd function—which returns only the current working directory—the mfilename function is utilized to obtain the full path of the directory containing the executing script. This approach ensures that file operations are correctly referenced relative to the script's location, enhancing reliability. The fileparts function is then applied to extract the directory path from the full script path. Subsequently, the fullfile function is used to construct the complete path to the "Trainingsprogress.mat" file within the application's directory. This method checks for the existence of the "Trainingsprogress.mat" file in the application's directory and retrieves its name or returns an empty value if the file is absent. By employing the same functions for both loading and saving operations, the project ensures that data is consistently managed within the application's directory.



```
%% Save the updated progress struct
base_dir = fileparts(mfilename('fullpath'));
save(fullfile(base_dir, 'trainingsprogress.mat'), 'progress');
```

Image 2 save data

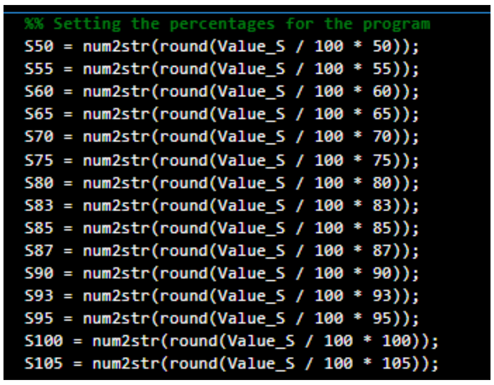
3.2. Generating the programs



```
% load progress, if it exists
if exist(file_path, 'file')
    load(file_path, "progress");
else
    progress = struct();
end

 %% Well done message
 uialert(app.UIFigure, "Without effort, you can't even pull a fish out of the pond (Soviet Saying)",
     "Your Program will be ready shortly", "Icon","success");

 %% Getting the values from the edit fields
 Value_S = app.rmSnatchEditField.Value;
 Value_CnJ = app.rmCleanJerkEditField.Value;
 Value_BS = app.rmBacksquatEditField.Value;
 Value_FS = app.rmForntSquatEditField.Value;

 %% Deciding which program will be generated
 if app.TypeofProgramButtonGroup.SelectedObject.Text == "Weightlifting Program"
     if string(app.HowmanydaysperweekDropDown.Value) == "3 Days per week"
         program = Generate_TP_3_day(Value_S, Value_CnJ, Value_BS, Value_FS);
     else
         program = Generate_TP_4_day(Value_S, Value_CnJ, Value_BS, Value_FS);
     end
 else
     if string(app.HowmanydaysperweekDropDown.Value) == "3 Days per week"
         program = Generate_SQ_3_day(Value_S, Value_CnJ, Value_BS);
     else
         program = Generate_SQ_4_day(Value_S, Value_CnJ, Value_BS, Value_FS);
     end
 end
```

Image 3 generating the programs

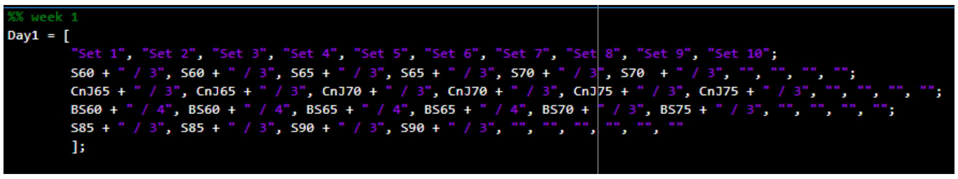
After determining the file path, the code checks for the existence of the 'Trainingsprogress.mat' file using MATLAB's exist function. If the file does not exist, a new progress structure is initialized. An alert is then displayed using the uialert function to inform the user that the process may take a short while. The user's one-repetition maximum (1RM) values are extracted from the numeric input fields in the GUI. Based on the selected program type and desired training frequency, the code generates the appropriate program using conditional statements. The generated program is stored in the program variable, utilizing a custom function that incorporates the user's 1RM values as input parameters.



```
%% Setting the percentages for the program
S50 = num2str(round(Value_S / 100 * 50));
S55 = num2str(round(Value_S / 100 * 55));
S60 = num2str(round(Value_S / 100 * 60));
S65 = num2str(round(Value_S / 100 * 65));
S70 = num2str(round(Value_S / 100 * 70));
S75 = num2str(round(Value_S / 100 * 75));
S80 = num2str(round(Value_S / 100 * 80));
S83 = num2str(round(Value_S / 100 * 83));
S85 = num2str(round(Value_S / 100 * 85));
S87 = num2str(round(Value_S / 100 * 87));
S90 = num2str(round(Value_S / 100 * 90));
S93 = num2str(round(Value_S / 100 * 93));
S95 = num2str(round(Value_S / 100 * 95));
S100 = num2str(round(Value_S / 100 * 100));
S105 = num2str(round(Value_S / 100 * 105));
```

Image 4 setting the percentages

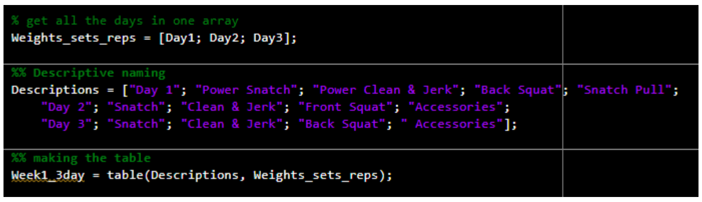
In the program generation functions, the code first calculates the required training percentages, rounds them, and converts the values into strings using MATLAB’s num2str() function. These values are then arranged into string arrays, each representing 10 sets per day. This structure ensures consistent table dimensions and allows for efficient computation. The program templates are based on the book *Fundamentals of the Soviet System* (Sisto, G. & Rojas, I., 2017) and a widely shared Russian squat program. Both have been adapted to meet the specific needs of weightlifters in Gießen—currently tailored to serve approximately one individual.



```
%% week 1
Day1 = [
       "Set 1", "Set 2", "Set 3", "Set 4", "Set 5", "Set 6", "Set 7", "Set 8", "Set 9", "Set 10";
       S60 + " / 3", S60 + " / 3", S65 + " / 3", S65 + " / 3", S70 + " / 3", S70  + " / 3", "", "", "", "";
       CnJ65 + " / 3", CnJ65 + " / 3", CnJ70 + " / 3", CnJ70 + " / 3", CnJ75 + " / 3", CnJ75 + " / 3", "", "", "", "";
       BS60 + " / 4", BS60 + " / 4", BS65 + " / 4", BS65 + " / 4", BS70 + " / 3", BS75 + " / 3", "", "", "", "";
       S85 + " / 3", S85 + " / 3", S90 + " / 3", S90 + " / 3", "", "", "", "", "", ""
       ];
```

Image 5 weights, sets & reps string array

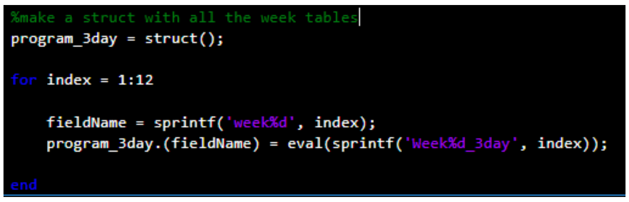
The information about weights, sets, and reps for each training day was first organized into daily string arrays. These were then concatenated and stored in a string array called Weights_set_reps. Descriptions of the training days and exercises were also stored in a separate column-based string array. Finally, both arrays were combined into a weekly training table that includes all descriptions and the corresponding Weights_set_reps data (see Image 6).



```
% get all the days in one array
Weights_sets_reps = [Day1; Day2; Day3];

%% Descriptive naming
Descriptions = ["Day 1"; "Power Snatch"; "Power Clean & Jerk"; "Back Squat"; "Snatch Pull";
    "Day 2"; "Snatch"; "Clean & Jerk"; "Front Squat"; "Accessories";
    "Day 3"; "Snatch"; "Clean & Jerk"; "Back Squat"; " Accessories"];

%% making the table
Week1_3day = table(Descriptions, Weights_sets_reps);
```

Image 6 making the week tables

Finally, all training weeks were stored in a program struct using a for loop and the sprintf() function, which incremented the week number with each iteration until all weeks were added. This struct is also the output variable returned by the program generation functions.

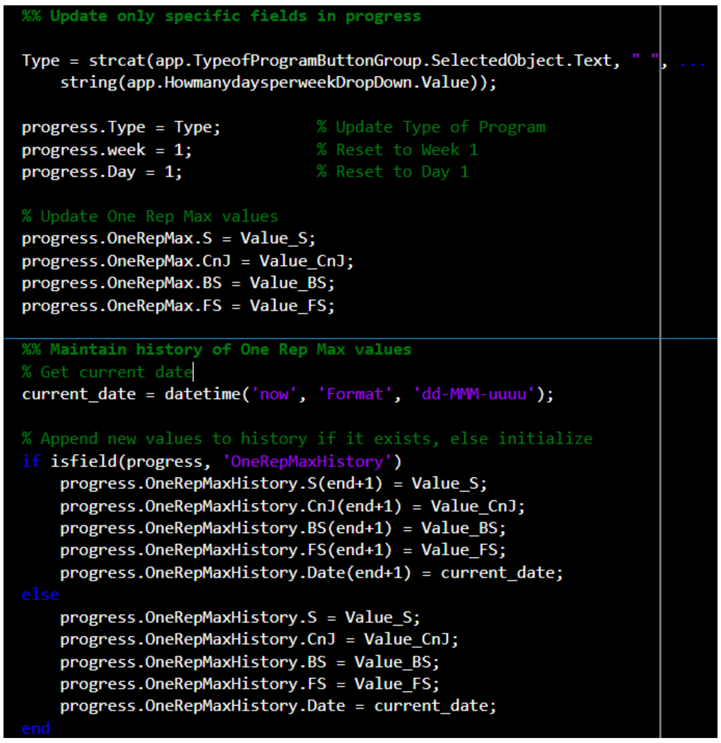


```
%make a struct with all the week tables
program_3day = struct();

for index = 1:12

    fieldName = sprintf('week%d', index);
    program_3day.(fieldName) = eval(sprintf('Week%d_3day', index));

end
```

Image 7 making the program struct

3.3. Storing the progress

Next, the metadata for the program is stored in a progress struct. First, variables such as progress.Type, progress.Day, and progress.Week are initialized. In the Type variable, a strcat() function is used to concatenate the string from the "TypeofProgramButtonGroup" with the characters from the "HowManyDaysPerWeekDropDown," which are converted into a string. The Week and Day variables store information about the specific training day. Additionally, the progress.OneRepMax variables store the current one-rep max values (image 7).



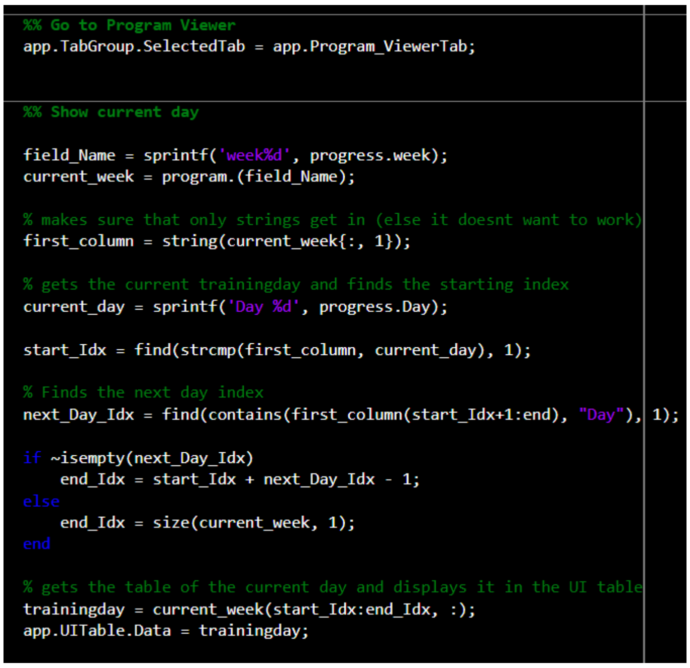
```
%% Update only specific fields in progress

Type = strcat(app.TypeofProgramButtonGroup.SelectedObject.Text, " ", ...
    string(app.HowmanydaysperweekDropDown.Value));

progress.Type = Type;          % Update Type of Program
progress.week = 1;             % Reset to Week 1
progress.Day = 1;              % Reset to Day 1

% Update One Rep Max values
progress.OneRepMax.S = Value_S;
progress.OneRepMax.CnJ = Value_CnJ;
progress.OneRepMax.BS = Value_BS;
progress.OneRepMax.FS = Value_FS;

%% Maintain history of One Rep Max values
% Get current date
current_date = datetime('now', 'Format', 'dd-MMM-uuuu');

% Append new values to history if it exists, else initialize
if isfield(progress, 'OneRepMaxHistory')
    progress.OneRepMaxHistory.S(end+1) = Value_S;
    progress.OneRepMaxHistory.CnJ(end+1) = Value_CnJ;
    progress.OneRepMaxHistory.BS(end+1) = Value_BS;
    progress.OneRepMaxHistory.FS(end+1) = Value_FS;
    progress.OneRepMaxHistory.Date(end+1) = current_date;
else
    progress.OneRepMaxHistory.S = Value_S;
    progress.OneRepMaxHistory.CnJ = Value_CnJ;
    progress.OneRepMaxHistory.BS = Value_BS;
    progress.OneRepMaxHistory.FS = Value_FS;
    progress.OneRepMaxHistory.Date = current_date;
end
```

Image 7: initializing the progress struct

To track the progress and one-rep max timeline, progress.OneRepMaxHistory is created to effectively store the new max weights. It's important to keep track of the days these maximum weights were achieved or first recorded in the program. This is done using the datetime() function, which returns the current date in the day-month-year format. The program first checks if a OneRepMaxHistory field already exists in the progress struct using the isfield() function. If the history exists, the new max values and the current date are concatenated. If not, a new variable is initialized, and the data is stored. The reason the one-rep max values are stored separately is that the previous max values are often irrelevant for most calculations. Therefore, it is easier to use the current OneRepMax variables for calculations and maintain the history only for cases when it's necessary to review past events.

3.4. Displaying the table in the program viewer tab

First, the tab is switched to the Program_ViewerTab. Then, a variable current_week is initialized to store the program for the current week. The first column of the current_week table contains information about the exercises and the current training day, which is why its position is used to determine which part of the table is displayed on the UITable. This first column is stored as a string, and a variable current_day is used to store information about the current day as a string variable. The sprintf() function is used to format this as "Day %d", with progress.Day being added to the string.



```
%% Go to Program Viewer
app.TabGroup.SelectedTab = app.Program_ViewerTab;


%% Show current day

field_Name = sprintf('week%d', progress.week);
current_week = program.(field_Name);

% makes sure that only strings get in (else it doesnt want to work)
first_column = string(current_week{:, 1});

% gets the current trainingday and finds the starting index
current_day = sprintf('Day %d', progress.Day);

start_Idx = find(strcmp(first_column, current_day), 1);

% Finds the next day index
next_Day_Idx = find(contains(first_column(start_Idx+1:end), "Day"), 1);

if ~isempty(next_Day_Idx)
    end_Idx = start_Idx + next_Day_Idx - 1;
else
    end_Idx = size(current_week, 1);
end

% gets the table of the current day and displays it in the UI table
trainingday = current_week(start_Idx:end_Idx, :);
app.UITable.Data = trainingday;
```

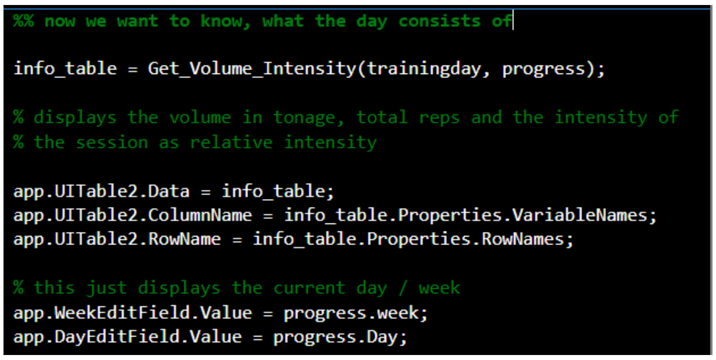
Image 8 displaying the current day in an UITable

To display the current day in the UITable, the start and end of the current week are determined based on the known position of the current training day in the current_week table. The start_idx is found as a boolean value (using find() with the second argument set to 1) by searching for the position in the first column that matches the current_day string using strcmp().

A similar approach is used to find the index of the next day, which is stored in next_Day_Idx. Here, contains() is used to search for the next day by comparing the string in the first column with "Day". This indicates the start of the next day. If a next_Day_Idx is found, the end_idx of the program to be displayed is next_Day_Idx - 1. If no next day is found, the rest of the table is displayed after the start_idx. The end_idx is determined using the size of the table, with the second argument set to 1 to get the height. Finally, the training day is declared as current_week(start_idx:end_idx), and the data is displayed in the UITable.

3.5. Displaying the training variable table

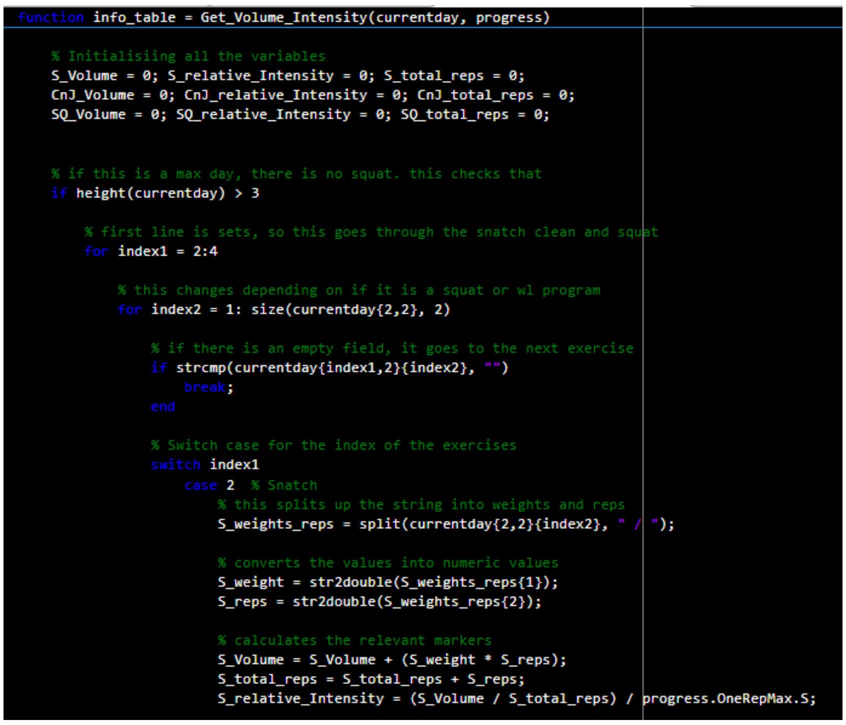
Finally, the "Generate new Weightlifting Program" button also generates a table with all the important training variables. This is done using the Get_Volume_Intensity() function, which was built to extract these variables. The data is then displayed in VariablesTable, along with row and column names, while the current week and day are shown in an uneditable edit field. This method of displaying tables and training variables is used in various functions throughout the app.



```
%% now we want to know, what the day consists of

info_table = Get_Volume_Intensity(trainingday, progress);

% displays the volume in tonage, total reps and the intensity of
% the session as relative intensity

app.UITable2.Data = info_table;
app.UITable2.ColumnName = info_table.Properties.VariableNames;
app.UITable2.RowName = info_table.Properties.RowNames;

% this just displays the current day / week
app.WeekEditField.Value = progress.week;
app.DayEditField.Value = progress.Day;
```

Image 9 displaying important training variables

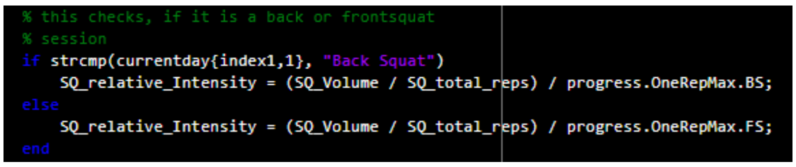
Next, the workings of the Get_Volume_Intensity() function are explained in more detail. Implementing this function was the most complex task of the entire project and posed several challenges. Only the weights for the main exercises were used in this part, as they are the most significant for the "feel" of the session and are most relevant to tracking progress.



```
function info_table = Get_Volume_Intensity(currentday, progress)

    % Initialisiing all the variables
    S_Volume = 0; S_relative_Intensity = 0; S_total_reps = 0;
    CnJ_Volume = 0; CnJ_relative_Intensity = 0; CnJ_total_reps = 0;
    SQ_Volume = 0; SQ_relative_Intensity = 0; SQ_total_reps = 0;


    % if this is a max day, there is no squat. this checks that
    if height(currentday) > 3

        % first line is sets, so this goes through the snatch clean and squat
        for index1 = 2:4

            % this changes depending on if it is a squat or wl program
            for index2 = 1: size(currentday{2,2}, 2)

                % if there is an empty field, it goes to the next exercise
                if strcmp(currentday{index1,2}{index2}, "")
                    break;
                end

                % Switch case for the index of the exercises
                switch index1
                    case 2  % Snatch
                        % this splits up the string into weights and reps
                        S_weights_reps = split(currentday{2,2}{index2}, " / ");

                        % converts the values into numeric values
                        S_weight = str2double(S_weights_reps{1});
                        S_reps = str2double(S_weights_reps{2});

                        % calculates the relevant markers
                        S_Volume = S_Volume + (S_weight * S_reps);
                        S_total_reps = S_total_reps + S_reps;
                        S_relative_Intensity = (S_Volume / S_total_reps) / progress.OneRepMax.S;
```

Image 10 calculating the Volume, total reps and relative intensity

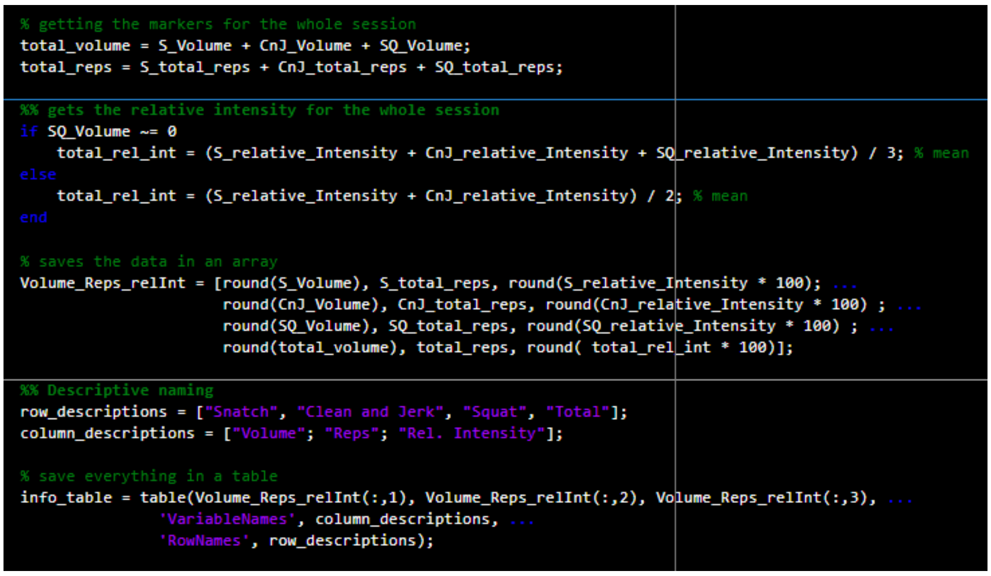
First, the necessary variables were initialized, and the height of the current day's table was examined. There are some "max-out" days, where the user is only asked to perform Snatches and Clean and Jerks. In these cases, we skip calculating the Squat volume, reps, and intensity, and set them to zero. Next, an index (ranging from 2 to 4, or 3 to 4 if there are no squats) was used in combination with a switch case to go through the different exercises. Since the rows for the Squat and Weightlifting programs differ, the length of the second index depends on the width of the weights, sets, and reps in the current day's table. The loop that iterates through the fields of the different exercise rows breaks as soon as a field is empty, which is checked using an if condition with strcmp(). The different fields in the table (weights and reps) are split into two parts using the split() function, with the "/" character as the delimiter. These parts are then converted into doubles using str2double(). Using this, the volume is calculated as the total weight (exercise or overall) multiplied by the total reps, and the relative intensity is calculated as volume / total reps / 1RM.



```
% this checks, if it is a back or frontsquat
% session
if strcmp(currentday{index1,1}, "Back Squat")
    SQ_relative_Intensity = (SQ_Volume / SQ_total_reps) / progress.OneRepMax.BS;
else
    SQ_relative_Intensity = (SQ_Volume / SQ_total_reps) / progress.OneRepMax.FS;
end
```

Image 11 special case front / back squat

There is one special case regarding the Squat: the one-rep max changes depending on the variation of the squat. The code shown in Image 11 illustrates how this case is handled. Finally, the total volume and reps are calculated by adding everything together, while the mean of all the combined relative intensities is determined. A numeric array is then built, where the volume is rounded using the round() function, and the relative intensity is converted into a percentage by rounding it and multiplying by 100.



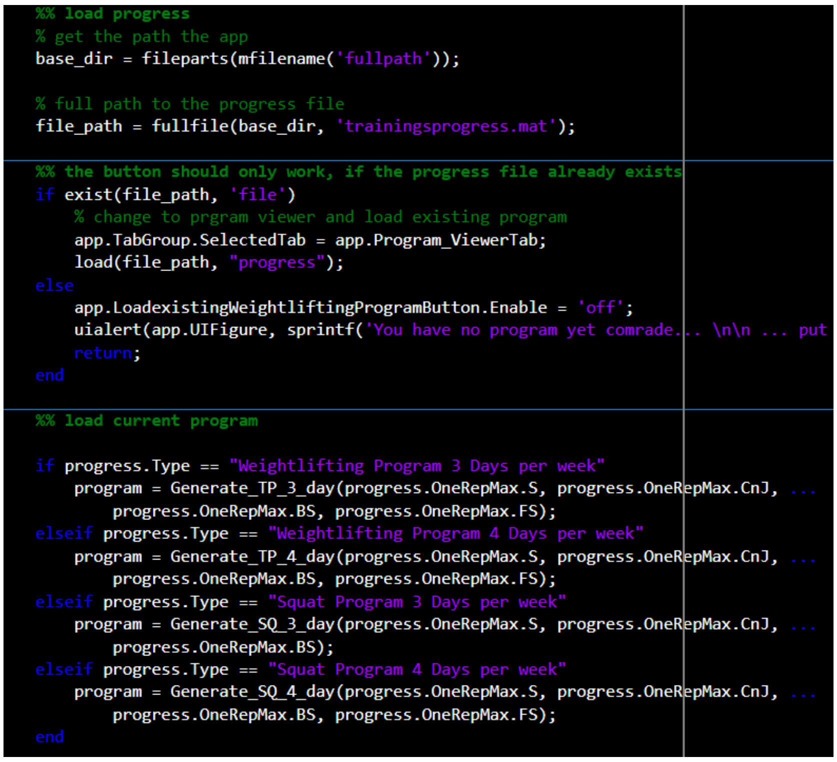
```matlab
% getting the markers for the whole session
total_volume = S_Volume + CnJ_Volume + SQ_Volume;
total_reps = S_total_reps + CnJ_total_reps + SQ_total_reps;

%% gets the relative intensity for the whole session
if SQ_Volume ~= 0
    total_rel_int = (S_relative_Intensity + CnJ_relative_Intensity + SQ_relative_Intensity) / 3; % mean
else
    total_rel_int = (S_relative_Intensity + CnJ_relative_Intensity) / 2; % mean
end

% saves the data in an array
Volume_Reps_relInt = [round(S_Volume), S_total_reps, round(S_relative_Intensity * 100); ...
                      round(CnJ_Volume), CnJ_total_reps, round(CnJ_relative_Intensity * 100) ; ...
                      round(SQ_Volume), SQ_total_reps, round(SQ_relative_Intensity * 100) ; ...
                      round(total_volume), total_reps, round( total_rel_int * 100)];

%% Descriptive naming
row_descriptions = ["Snatch", "Clean and Jerk", "Squat", "Total"];
column_descriptions = ["Volume"; "Reps"; "Rel. Intensity"];

% save everything in a table
info_table = table(Volume_Reps_relInt(:,1), Volume_Reps_relInt(:,2), Volume_Reps_relInt(:,3), ...
               'VariableNames', column_descriptions, ...
               'RowNames', row_descriptions);
```

Image 12 making the volume, reps and intensity table

Finally, variables with descriptive names are initialized to hold the descriptions for the different fields, improving readability. All of this is then packed into a table, which serves as the output variable of the Get_Volume_Intensity() function.

3.6. Loading an existing weightlifting program

After pressing the "Load Existing Weightlifting Program" button on the opening window, the application first checks if it can find a trainingsprogress.mat file in the application's directory. If the file exists, the program viewer tab is opened, and the progress struct from the file is loaded for further calculations. If the file is not found, a uialert message is displayed, the button is disabled to prevent further use, and no additional action is taken. To load the current program, the progress.Type string is compared to the available options, and the appropriate program is loaded (see image 13).

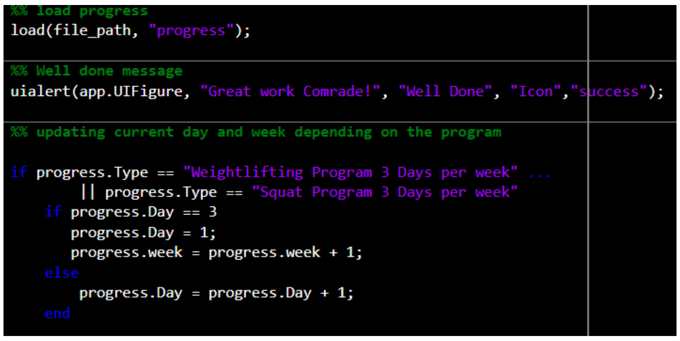


```
%% load progress
% get the path the app
base_dir = fileparts(mfilename('fullpath'));

% full path to the progress file
file_path = fullfile(base_dir, 'trainingsprogress.mat');

%% the button should only work, if the progress file already exists
if exist(file_path, 'file')
    % change to prgram viewer and load existing program
    app.TabGroup.SelectedTab = app.Program_ViewerTab;
    load(file_path, "progress");
else
    app.LoadexistingWeightliftingProgramButton.Enable = 'off';
    uialert(app.UIFigure, sprintf('You have no program yet comrade... \n\n ... put
    return;
end

%% load current program

if progress.Type == "Weightlifting Program 3 Days per week"
    program = Generate_TP_3_day(progress.OneRepMax.S, progress.OneRepMax.CnJ, ...
        progress.OneRepMax.BS, progress.OneRepMax.FS);
elseif progress.Type == "Weightlifting Program 4 Days per week"
    program = Generate_TP_4_day(progress.OneRepMax.S, progress.OneRepMax.CnJ, ...
        progress.OneRepMax.BS, progress.OneRepMax.FS);
elseif progress.Type == "Squat Program 3 Days per week"
    program = Generate_SQ_3_day(progress.OneRepMax.S, progress.OneRepMax.CnJ, ...
        progress.OneRepMax.BS);
elseif progress.Type == "Squat Program 4 Days per week"
    program = Generate_SQ_4_day(progress.OneRepMax.S, progress.OneRepMax.CnJ, ...
        progress.OneRepMax.BS, progress.OneRepMax.FS);
end
```

Image 13 checking if there is a program exists

3.7. Going from day to day

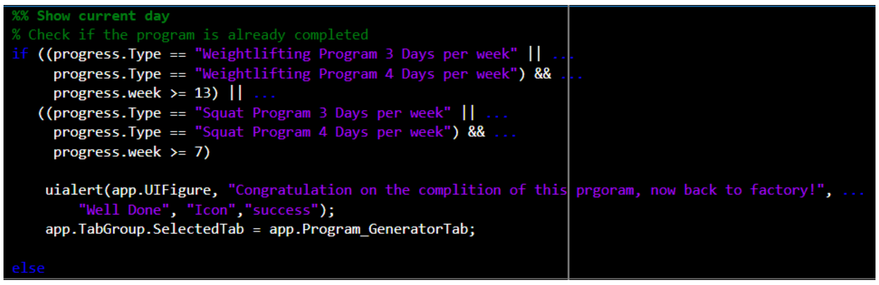
When transitioning from one training day to the next, the progress.Day and progress.Week need to be updated. This is done by checking the type of program and determining the current day the user is on. If the program is set to 3 days per week and the user is on the third day, the progress.Week is increased, and the day is reset to 1. Otherwise, if the user is not on the last day, the progress.Day is simply incremented by 1. The same logic applies for 4-day per week programs (see image 13). Afterward, the current program is loaded in the same way as when an existing program is loaded.



```
%% load progress
load(file_path, "progress");

%% Well done message
uialert(app.UIFigure, "Great work Comrade!", "Well Done", "Icon","success");

%% updating current day and week depending on the program

if progress.Type == "Weightlifting Program 3 Days per week" ...
        || progress.Type == "Squat Program 3 Days per week"
    if progress.Day == 3
        progress.Day = 1;
        progress.week = progress.week + 1;
    else
         progress.Day = progress.Day + 1;
    end
```

Image 14 updating the progress file

It is also important to check if the person has completed the last day of the current program. This is done by checking the program type and comparing the current week with the program's maximum week limit. If the person has completed the program, they are greeted with a moderately funny message and redirected to the Program Generator tab. If the program is not yet finished, the day and/or week are updated in the Edit fields, and both the ProgramTable and VariableTable are refreshed. Finally, the progress is saved back into the trainingsprogress.mat file.

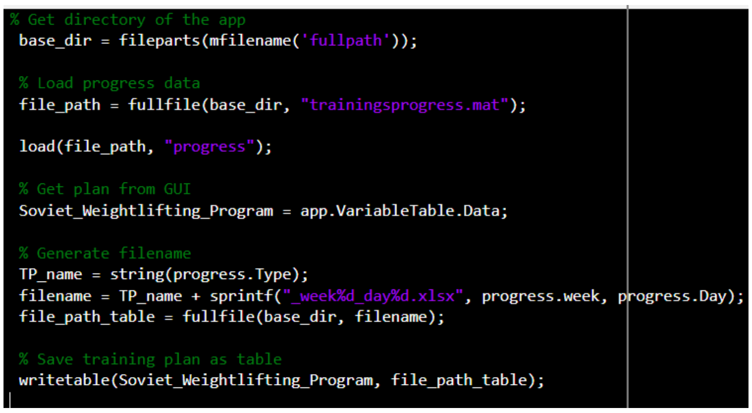


```
%% Show current day
% Check if the program is already completed
if ((progress.Type == "Weightlifting Program 3 Days per week" || ...
     progress.Type == "Weightlifting Program 4 Days per week") && ...
     progress.week >= 13) || ...
   ((progress.Type == "Squat Program 3 Days per week" || ...
     progress.Type == "Squat Program 4 Days per week") && ...
     progress.week >= 7)

    uialert(app.UIFigure, "Congratulation on the complition of this prgoram, now back to factory!", ...
        "Well Done", "Icon","success");
    app.TabGroup.SelectedTab = app.Program_GeneratorTab;

else
```

Image 15 checking, if the user completed the program

3.8. Saving the program as a csv file

To allow the user flexibility with the program and the ability to take it anywhere, an option to save the current day's training program was provided. The current day is stored directly in the Soviet_Weightlifting_Program variable from the ProgramTable data. The file name is a combination of progress.Type, concatenated with the respective week and day using sprintf(), along with progress.week and progress.Day. The fullfile function is used with the base directory (where the app is stored) to ensure the program file is saved in the same location as the app. Finally, the CSV table is written using the writetable() function, with the table data and file path as input arguments.

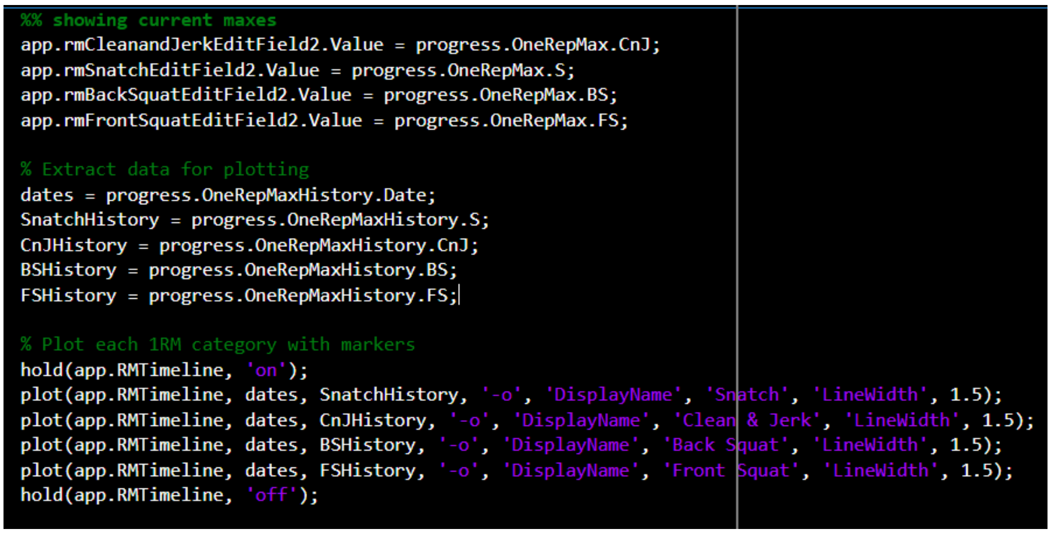


```
% Get directory of the app
 base_dir = fileparts(mfilename('fullpath'));

 % Load progress data
 file_path = fullfile(base_dir, "trainingsprogress.mat");

 load(file_path, "progress");

 % Get plan from GUI
 Soviet_Weightlifting_Program = app.VariableTable.Data;

 % Generate filename
 TP_name = string(progress.Type);
 filename = TP_name + sprintf("_week%d_day%d.xlsx", progress.week, progress.Day);
 file_path_table = fullfile(base_dir, filename);

 % Save training plan as table
 writetable(Soviet_Weightlifting_Program, file_path_table);
```

Image 16 saving the program as csv

3.9. Displaying the one rep max timeline.

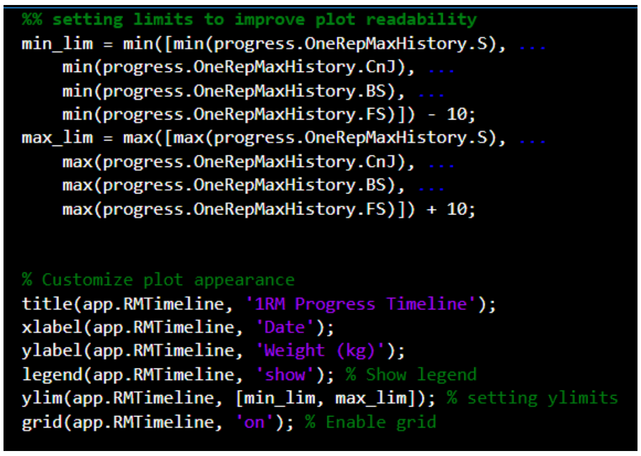
After pressing the "Recordboard" button, the user is directed to a new tab, where the progress struct is loaded. The editable One Rep Max edit fields display the current max values, and the OneRepMaxHistory values are stored in variables for easier access. Finally, the current max values are plotted on the 1 RM Timeline, with the dates shown on the x-axis and the One Rep Max values in kilograms on the y-axis (image 17).



```
%% showing current maxes
app.rmCleanandJerkEditField2.Value = progress.OneRepMax.CnJ;
app.rmSnatchEditField2.Value = progress.OneRepMax.S;
app.rmBackSquatEditField2.Value = progress.OneRepMax.BS;
app.rmFrontSquatEditField2.Value = progress.OneRepMax.FS;

% Extract data for plotting
dates = progress.OneRepMaxHistory.Date;
SnatchHistory = progress.OneRepMaxHistory.S;
CnJHistory = progress.OneRepMaxHistory.CnJ;
BSHistory = progress.OneRepMaxHistory.BS;
FSHistory = progress.OneRepMaxHistory.FS;

% Plot each 1RM category with markers
hold(app.RMTimeline, 'on');
plot(app.RMTimeline, dates, SnatchHistory, '-o', 'DisplayName', 'Snatch', 'LineWidth', 1.5);
plot(app.RMTimeline, dates, CnJHistory, '-o', 'DisplayName', 'Clean & Jerk', 'LineWidth', 1.5);
plot(app.RMTimeline, dates, BSHistory, '-o', 'DisplayName', 'Back Squat', 'LineWidth', 1.5);
plot(app.RMTimeline, dates, FSHistory, '-o', 'DisplayName', 'Front Squat', 'LineWidth', 1.5);
hold(app.RMTimeline, 'off');
```

Image 17 Dsiplaying the one rep max timeline 1

To further enhance the readability of the table, title(), xlabel(), ylabel(), legend(), and grid() were added. The ylim was also adjusted, setting the lower limit to 10 kg below the lowest recorded weight in the OneRepMaxHistory, and the upper limit to 10 kg above the highest recorded weight. This adjustment improved the clarity of the plot (image 18).

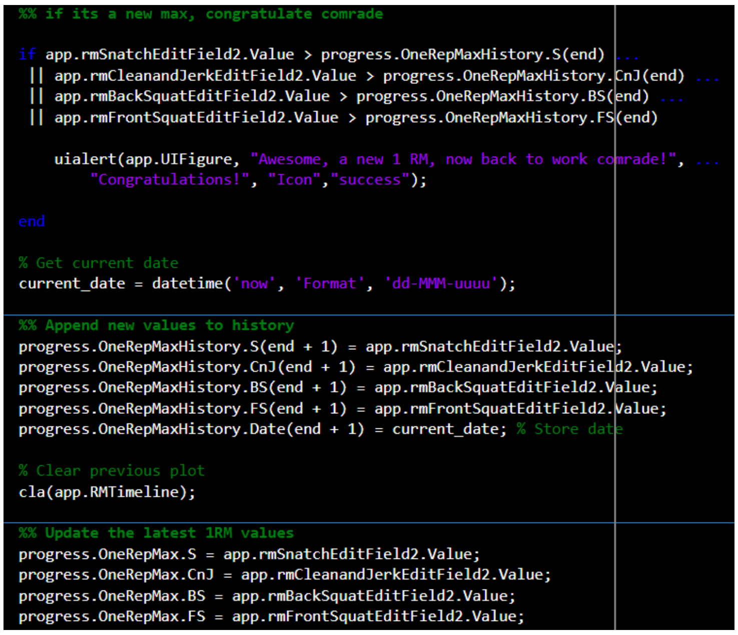


```
%% setting limits to improve plot readability
min_lim = min([min(progress.OneRepMaxHistory.S), ...
    min(progress.OneRepMaxHistory.CnJ), ...
    min(progress.OneRepMaxHistory.BS), ...
    min(progress.OneRepMaxHistory.FS)]) - 10;
max_lim = max([max(progress.OneRepMaxHistory.S), ...
    max(progress.OneRepMaxHistory.CnJ), ...
    max(progress.OneRepMaxHistory.BS), ...
    max(progress.OneRepMaxHistory.FS)]) + 10;


% Customize plot appearance
title(app.RMTimeline, '1RM Progress Timeline');
xlabel(app.RMTimeline, 'Date');
ylabel(app.RMTimeline, 'Weight (kg)');
legend(app.RMTimeline, 'show'); % Show legend
ylim(app.RMTimeline, [min_lim, max_lim]); % setting ylimits
grid(app.RMTimeline, 'on'); % Enable grid
```

Image 18 improving the plot

3.10. Updating one rep maxes and the remaining functionalities of the recordboard

When a user achieves a new personal record higher than the previous one, the app first congratulates the user with a pop-up window. Afterward, the current_date and new maximum values are added to the OneRepMaxHistory variable. The figure is cleared using cla() and the updated maxes are stored in the progress.OneRepMax struct (image 19).



```
%% if its a new max, congratulate comrade

if app.rmSnatchEditField2.Value > progress.OneRepMaxHistory.S(end) ...
 || app.rmCleanandJerkEditField2.Value > progress.OneRepMaxHistory.CnJ(end) ...
 || app.rmBackSquatEditField2.Value > progress.OneRepMaxHistory.BS(end) ...
 || app.rmFrontSquatEditField2.Value > progress.OneRepMaxHistory.FS(end)

    uialert(app.UIFigure, "Awesome, a new 1 RM, now back to work comrade!", ...
        "Congratulations!", "Icon","success");

end

% Get current date
current_date = datetime('now', 'Format', 'dd-MMM-uuuu');

%% Append new values to history
progress.OneRepMaxHistory.S(end + 1) = app.rmSnatchEditField2.Value;
progress.OneRepMaxHistory.CnJ(end + 1) = app.rmCleanandJerkEditField2.Value;
progress.OneRepMaxHistory.BS(end + 1) = app.rmBackSquatEditField2.Value;
progress.OneRepMaxHistory.FS(end + 1) = app.rmFrontSquatEditField2.Value;
progress.OneRepMaxHistory.Date(end + 1) = current_date; % Store date

% Clear previous plot
cla(app.RMTimeline);

%% Update the latest 1RM values
progress.OneRepMax.S = app.rmSnatchEditField2.Value;
progress.OneRepMax.CnJ = app.rmCleanandJerkEditField2.Value;
progress.OneRepMax.BS = app.rmBackSquatEditField2.Value;
progress.OneRepMax.FS = app.rmFrontSquatEditField2.Value;
```

Image 19 updating maxes

Afterward, the plot is generated using the same code as in the previous section. Finally, the progress is saved in the trainingsprogress.mat file, ensuring the new one-rep max values are stored. On the Recordboard tab, there is also an option to generate a new program, which simply redirects the user to the Program Generator tab. If the user wishes to return to the Program Viewer tab, the program is loaded similarly to how it is loaded when pressing the “Load existing Weightlifting Program” button, ensuring that any updated one-rep max values are reflected in the program.

4. Reflexion, Difficulties and Extension Possibilities

4.1. Reflexion

Overall, the outcome of this project is very satisfactory. All components function successfully, and the features operate as intended. The solution effectively addresses the problem that prompted this project. Previously, calculating weights and percentages, and documenting each set, was time-consuming. Now, users can effortlessly generate the program with just a few clicks. The application also tracks the exact dates and times when an athlete achieves a new maximum. After extensive peer testing, the program has proven stable and reliable. Additionally, a significant number of local weightlifters in Gießen are currently using the application. This observation is based on the knowledge that the author's cellar is near the only training facility in Gießen that meets the equipment requirements for Olympic weightlifters.

Although certain aspects of the code's structure and execution could have been optimized, such as the program generation functions returning the entire program and subsequently extracting the current day, a more efficient approach could have been to modify these functions to directly return the current training day. Incorporating the 'progress' structure as an input variable would streamline this process. This adjustment would simplify setting dimensions for exercises throughout the week. These improvements are considerations for future updates.

4.2. Difficulties

The most complex and time-consuming function to develop was the Get_Volume_Intensity function. The challenges primarily stemmed from the intricate indexing required to accurately access specific fields within the data structure. Extracting the desired values for weights and repetitions necessitated meticulous attention to detail, especially when implementing appropriate conditions and loops. These difficulties were exacerbated by the table-based storage structure, which, while efficient, introduced complexity in data retrieval.

An alternative approach that might have simplified indexing involves utilizing current_week.Weights_sets_reps, potentially streamlining data access. However, this method was not employed in the current implementation. Additionally, the existing code structure lacks flexibility in program design; it mandates a fixed exercise order, limiting customization. Addressing this rigidity is a priority for future updates to enhance user adaptability.

Aligning the dimensions of string arrays for each week presented another significant challenge. Extracting individual sets, percentages, and repetitions from the weightlifting program was labor-intensive, and alternative data structures might have alleviated this issue. For instance, employing cell arrays or structures with dynamic field names could have provided more flexibility in handling variable-length data . However, the decision was made to conform to the existing structure, ensuring consistency across the application.

Despite these challenges, the current structure meets all project requirements and operates effectively. The development process offered valuable insights into MATLAB's data handling capabilities and highlighted areas for potential improvement. Future revisions will focus on enhancing flexibility and optimizing data management to better accommodate diverse user needs.

4.3. Extension possibilities

The current program possesses basic functionalities that provide a complete experience while leaving room for future improvements. One potential enhancement is the ability to track training more effectively. This could involve logging the Rate of Perceived Exertion (RPE) to monitor the "hardness" of workouts. Additionally, recording the number of failed attempts per session could serve as an indicator of training quality. Implementing a commentary section for each day would allow users to note aspects that went well or areas needing improvement, as well as cues for specific lifts, helping users focus on refining their technique.

These variables—RPE, session quality, and daily notes—could be integrated into a training log table that updates with each completed workout. The RPE and failure data could then be used to autoregulate the weights used in subsequent sessions. This approach would enable athletes to intensify their training when progress permits and adjust loads when sessions feel particularly challenging. Alternatively, weights could be reduced for exercises where the athlete frequently encounters difficulties. Incorporating this feature would be a valuable addition in future updates.

If training logging capabilities are introduced, allowing users to modify variations of main exercises and accessories could further personalize training regimens. For instance, if an athlete needs to focus on a specific phase of a lift, providing a selection of alternative exercises to target that phase would be beneficial. Adjusting percentages for these exercises—for example, replacing the hang snatch with a decline snatch to emphasize the initial pulling phase—would also be advantageous. Including options to edit accessory exercises, specifying weights and repetitions, would help athletes monitor a broader scope of their workouts and progress.

To support these features, the Get_Volume_Intensity function might require updates to accommodate training program templates for various strength sports. This could include logging intensity and volume for exercises like the deadlift, bench press, or even sprinting metrics such as intensity and times. Such versatility would make the program applicable to a wider range of athletes seeking to enhance their performance.

Looking further ahead, implementing a machine learning algorithm could optimize training plans. By analyzing user data over time, the algorithm could recommend personalized adjustments to weights, sets, reps, and exercise variations based on individual progress and feedback. This approach aligns with current trends in sports analytics, where machine learning is utilized to derive insights from performance data. Analyzing Soviet-era weightlifting methods through this lens might also yield valuable training strategies. Overall, this project has been both enjoyable to develop and a valuable learning experience, enhancing problem-solving skills in MATLAB programming.

5. Literature

Matveyev, L. P. (1964). *Fundamentals of sports training*. Progress Publishers.

Sisto, G., & Rojas, I. (2017). *Fundamentals of the Soviet System: The Soviet Weightlifting System and Modern Applications of the Sport of Weightlifting*. Risto Sports LLC.

Yakovlev, N. N. (1955). Supercompensation theory and lifting principles. *Fitness Stack Exchange*.

Yessis, M. (1987). *Secrets of Soviet sports fitness and training*. Arbor House.

6. Images

Institut für Sportwissenschaft
Kugelberg 62
35394 Gießen

7. Decleration of Authorship

Name: Philipp Ewen
Matriculation number.: 8102119
Degree program: Human Movement Analytics: Biomechanics, Motor Control and Learning

I hereby declare to the Department 06 Psychology and Sports Science, Institute of Sports Science, that the present project with the subject:

Soviet Style Weightlifting Program Generator App

was written independently (together with the members of my project group), has not yet been submitted in whole or in part as an examination paper and was prepared exclusively with the aid of the works and documents mentioned in the list of sources and references. This also applies to the use of a text-based dialog system (such as ChatGPT) or support by another form of artificial intelligence.
AI tools (such as ChatGPT, Elicit, Paraphraser, etc.) were used in the following places for the stated purposes:

Specification of the AI used	Indication of the purpose
ChatGPT	Correction of grammar and form in the project report

I agree to the verification by anti-plagiarism software.

(Location, Date) (Signature)

15.03.2025, Fürth [illegible]