Czech University of Life Sciences Prague

Faculty of Economics and Management

Department of Systems Engineering

Bachelor Thesis

Linear Programming guided solver

Jasurbek Normakhmadov

CZECH UNIVERSITY OF LIFE SCIENCES PRAGUE

Faculty of Economics and Management

BACHELOR THESIS ASSIGNMENT

Jasurbek Normakhmadov

Systems Engineering and Informatics
Informatics

Thesis title

Linear Programming guided solver

Objectives of thesis

Solving a Linear Programming with Simplex Algorithm can be hard for students. The aim of the thesis is designing a step-by-step linaer programming solver with explanations.

Methodology

The information in the theoretical part will be extracted from the relevant sources such as monographies, scientific papers and online sources. It will contain the knowledge base for the practical part which includes mostly Linear programming and JavaScript programming.

In the practical part, a website will be created with the use of tools such as HTML and CSS. The algorithm itself will be designed in JavaScript programming language. This will involve JavaScript library React.

The proposed extent of the thesis

30-40

Keywords

JavaScript, ReactJS, React, Linear Programming, Simplex Method, Jordan Elimination

---

Recommended information sources

DANTZIG, G B. *Linear programming : 2: theory and extensions.* Madison: Springer, 2003. ISBN 978-0387986135.

FLANAGAN, D. *JavaScript : the definitive guide.* Sebastopol, CA: O'Reilly, 2002. ISBN 0-596-00048-0.

VANDERBEI, R J. *Linear programming : foundations and extensions.* New York: Springer, 2001. ISBN 978-0-387-74387-5.

Expected date of thesis defence

2019/20 SS – FEM

The Bachelor Thesis Supervisor

Ing. Robert Hlavatý, Ph.D.

Supervising department

Department of Systems Engineering

Electronic approval: 21. 2. 2020

doc. Ing. Tomáš Šubrt, Ph.D.

Head of department

Electronic approval: 24. 2. 2020

Ing. Martin Pelikán, Ph.D.

Dean

Prague on 22. 03. 2020

Declaration

I declare that I have worked on my bachelor thesis titled "Linear Programming guided solver" by myself and I have used only the sources mentioned at the end of the thesis. As the author of the bachelor thesis, I declare that the thesis does not break copyrights of any third person.

In Prague on 20.03.2020 _____

Acknowledgement

I would like to thank Robert Hlavatý, for his advice and support during my work on this thesis

Linear Programming guided solver

Abstract

The purpose of the thesis is to explain basics of Simplex Method of Linear programming problem and JavaScript programming language. There will be explained definitions, installed them through Command Prompt, showed result in a browser.

Moreover, there will be covered tools of JavaScript's frameworks and libraries such as React.js, React-Redux, React-Bootstrap. There will be a lot of typing of code in Visual Studio code editor using React.js library. A reason of using React.js library of JavaScript programming language is popularity of library among JavaScript's library, which is very easy to use and popular among Web-Developers.

At the end, is to build pseudo-calculator with guide that would solve Linear programming problem using Simplex Method. It will be run on a browser and have some input fields that a user has to type in order to solve his/her Linear programming problem. After that, it would calculate the problem by given values. During a solution, it will show pivot column, pivot row and pivot value, also, it will calculate test of optimality, test of feasibility and Jordan Elimination.

Keywords: Linear Programming Problem, Simplex Method, Jordan Elimination, JavaScript, React.js, React, React-Redux, React-Bootstrap, Node.js

Table of content

List of pictures

List of abbreviations

JS – JavaScript

HTML – Hypertext Markup Language

CSS - Cascading Style Sheets

ECMAscript – scripting language specification standardized by ECMA International, for example ES5, ES6

BOM - Browser Object Model

DOM - Document Object Model

MVC - Model View Controller

JSX – JavaScript XML

XML - Extensible Markup Language

NPM - Node package manager

NPX – Execution of NPM packages

Node.js – Environment that executes JavaScript code outside of a browser

UML - Unified Modeling Language

VS code – Visual Studio code

1 Introduction

The aim of the thesis is to explain Linear programming problem using Simplex method in coding format. Using Simplex method in order to solve Linear problem is the one of the most difficult part of students' study. Some of the students want to implement this in coding way. There are no any resources on the internet that would explain creating Simplex method in JavaScript programming language. By the coding format means typing several code of lines in order to run it in a browser, in the following pages will be explained tools that were used in order to implement it. There will be explained briefly about what Simplex Method, JavaScript, React.js, React – Redux, React – Bootstrap are in practical part. In theory part will be gone through steps of making alive Simplex Method calculator, by using above mentioned tools and all steps will be explained step by steps.

The WWII influenced a development of science such as physics, chemical, technology and so on. Simplex method that is used in Linear programming was found during that time and founder was George Dantzig (1915-2005). He developed this method, because during that period mathematicians used only two variables to solve Linear programming problem (LPP) on a two-dimension graph. There was lack of variables, because just two variables for the whole of problem was not easy. Moreover, if you wanted to add one more variable then graph had to be three dimensional. This big issue pushed G. Dantzig to create a table where it combines more than two variables.

The JavaScript programming language is on the top demanding programming languages and it is growing. It was found at the beginning of the twentieth century by the Netscape Corporation. It was started to use widely in web development, it has a good interaction with HTML and CSS. It has different libraries and frameworks, for example, Angular.js, Vue.js, React.js, Redux.js, jQuery, Node.js and so on.

2 Objectives and Methodology

2.1 Objectives

The main objective of this bachelor thesis is to build Simplex Method which is used in Linear Programming. Solving a Linear Programming with Simplex Algorithm can be hard for students. The aim of the thesis is designing a step-by-step linear programming solver with explanations. In literature review part, will be explained tools which were used in order to implement it as a Web page.

2.2 Methodology

The information in the theoretical part will be extracted from the relevant sources such as monographies, scientific papers and online sources. It will contain the knowledge base for the practical part which includes mostly Linear programming and JavaScript programming. In the following part will be explained about tools that were used in implementation of Solver for Linear Programming in React.js. These tools are mainly about JavaScript, how it works, its library React.js, how does it work, how to install it, how to work with it, what is Virtual DOM, why do we need Redux.js and what it does.

In the practical part, a website will be created with the use of tools such as HTML and CSS. The algorithm itself will be designed in JavaScript programming language. This will involve JavaScript library React.js. The reason why React.js was chosen is because of easy to use and popularity among JavaScript libraries. In this section will be discussed how React app that solves Linear programming problem. It explains how it has been done by building a bunch of functions that interacts with each other and gathered in several classes. Moreover, in order to work with values of the Linear programming what role played in Redux.js that works with a state among React App.

3 Literature Review

3.1 Simplex Method

"The simplex algorithm is always initiated with a program whose equations are in canonical form" (Dantzig, 1963)

There are steps that has to be done in order to solve a Linear Programming problem and they are:

1. Add slack variables to inequation, to make them in equation form
2. Construct an objective function with nonzero condition
3. Build a simplex tableau by creating augmented matrix
4. Test of optimality (Zj – Cj), if the objective function has to be maximized and there are any negative value, then continue further
5. Test of feasibility (Omega test), find pivot row
6. Find pivot value
7. Build new tableau by working with row operations – Jordan Elimination
8. Repeat steps from 4-7

A simple tableau looks like that:

	Cj	1	4	3	0	0	0		
Cb	BV	x1	x2	x3	s1	s2	s3	RHS	Omega
0	S1	0	3	0	1	-3	1	15	5
1	x1	1	2	0	0	5	3	14	7
3	x3	0	0	1	0	7	2	20	infinity
Zj-Cj		0	-2	0	0	26	9	74	

Figure 1 - A simple tableau of Simplex Method

BV – stands for "basic variables"

Cb – stands for "coefficients of basic variables"

Cj – stands for "coefficient of variables"

Zj-Cj – test of optimality

Omega – test of feasibilities

RHS – stands for "right hand side"

3.2 What is JavaScript?

"JavaScript is a scripting language designed to interact with web pages and is made up of the following three distinct parts" (Matt, 2019). These three parts are DOM (Document object Model) which works with content of website; BOM (Browser Object Model) interacts with browser; ECMAscript that uses a core of functionality

JavaScript works with HTML and CSS to fabricate web applications or pages. JavaScript is upheld by most current internet browsers like Google Chrome, Firefox, Safari, Microsoft Edge, Opera, and so on. Most versatile programs for Android and iPhone now support JavaScript also.

JavaScript controls the dynamic components of pages. It works in internet browsers and, all the more as of late, on web servers also. Application Programming Interfaces (API) are likewise bolstered by JavaScript, giving a greater usefulness.

3.3 How does JavaScript work?

The internet browser stacks a site page, parses the HTML, and makes what is known as a Document Object Model (DOM) from the substance. The DOM introduces a live perspective on the page to JavaScript code. The program will at that point get everything connected to the HTML, similar to pictures and CSS records. The CSS data originates from the CSS parser. The HTML and CSS are assembled by the DOM to make the website page first. At that point, the programs' JavaScript motor burdens JavaScript records and inline code however does not run the code right away. It hangs tight for the HTML and CSS to get done with stacking. When this is done, the JavaScript is executed in the request the code is composed. This outcomes in the DOM being refreshed by JavaScript code and rendered by the program. The request here is significant. On the off chance that the JavaScript didn't hang tight for the HTML and CSS to complete, it would not have the option to change the DOM components.

3.4 What is React.js?

"React (sometimes called React.js or React.js) is an open-source JavaScript library that provides a view for data rendered as HTML. Components have been used typically to render React views that contain additional components specified as custom HTML tags. React gives a trivial virtual DOM, powerful views without templates, unidirectional data

flow, and explicit mutation. It is very methodical in updating the HTML document when the data changes; and provides a clean separation of components on a modern single-page application." (Mehul & Harmeet, 2016)

React permits engineers to make enormous web applications that can change information, without reloading the page. The primary reason for React is to be quick, versatile, and straightforward. It works just on UIs in the application. This relates to the view in the Model View Controller (MVC) format. It tends to be utilized with a mix of other JavaScript libraries or structures, for example, Angular JS in MVC.

3.4.1 **Advantages of React.js**

There are such a significant number of open-source stages for making the front-end web application advancement simpler, as Angular. Let us investigate the advantages of React over other serious advances or systems. With the front-end world-changing consistently, it's difficult to dedicate time to learning another system – particularly when that structure could at last become an impasse.

1. Effortlessness

React.js is only less complex to get a handle on immediately. The part-based methodology, all around characterized lifecycle, and utilization of out and out JavaScript make React easy to learn, construct an expert web (and versatile applications), and bolster it. React utilizes an uncommon linguistic structure called JSX which permits to blend HTML in with JavaScript. This is not a necessity; Developer can in any case write in plain JavaScript yet JSX is a lot simpler to utilize.

2. Simple to learn

Anybody with a fundamental past information in programming can without much of a stretch comprehend React while Angular and Ember are alluded to as "Area explicit Language", inferring that it is hard to learn them. To respond, it simply needs essential information on CSS and HTML.

3. Local Approach

React can be utilized to make portable applications (React Native). Furthermore, React.js is a diehard aficionado of reusability, which means broad code reusability is bolstered. So simultaneously, it can be made on IOS, Android and Web applications.

4. Information Binding

React utilizes single direction information authoritative and an application engineering called Flux controls the progression of information to parts through one control point – the dispatcher. It is simpler to troubleshoot independent parts of enormous React.js applications.

5. Execution

React does not offer any idea of an inherent holder for reliance. It can utilize Browserify, Require JS, ES6 modules which it can be utilized through Babel, React.js-di to infuse conditions naturally.

6. Testability

React.js applications are too simple to test. React perspectives can be treated as elements of the state, so it can control with the state would go to the React.js view and investigate the yield and activated activities, occasions, capacities, and so on.

3.4.2 Features of React.js

1. JSX

JSX is a JavaScript extension that provides syntactic sugar (sugar-coating) for function calls and object construction, particularly React.createElement() (Azat, 2017)

Much the same as XML, JSX labels have a label name, properties, and youngsters. On the off chance that a property estimation is encased in cites, the worth is a string. Something else, envelop the incentive by props and the worth is the encased JavaScript articulation.

2. React Native

React has local libraries that were declared by Facebook in 2015, which gives the react design to local applications like IOS, Android and UPD.

The cool part of working with React Native is that program uses standard web technologies like JavaScript (JSX), CSS, and HTML, yet application is fully native. In other words, application is fast and smooth, and it is equivalent to any native application built using traditional iOS technologies like Objective-C and Swift. (Abhishek & Akshat, 2019)

3.4.3 **Installation of React.js**

To introduce and utilize React.js, it needs two things: Node.js and NPM. When the individuals realize what they are doing and why, so there should be discussed a little about these two. Node.js is a JavaScript run-time condition that permit us to execute JavaScript code like in the event that there were dealing with a server. Recollect that each web application is intended to be executed in a server (or a nearby server, in case we are running it in PCs). In the other hand NPM is a bundle administrator for JavaScript, that is, NPM permits us to introduce JavaScript libraries to make experience significantly increasingly more extravagant by growing the fundamental functionalities.

In order to install React.js, first it needs to be installed Node.js and NPM(NPM is included with Node.js). On the official webpage it can be installed by several clicks. (NodeJS, 2020)

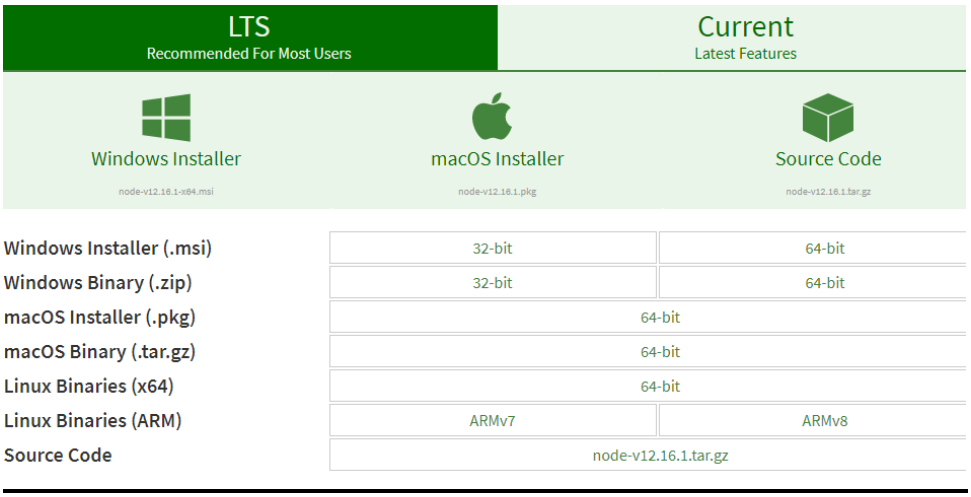

Figure 2 - Installation of Node.js

After that it can be checked the version of Node.js and NPM: **node -v** and **npm -v**. In this case, Node.js version is 13.1.0 and NPM version is 6.12.1

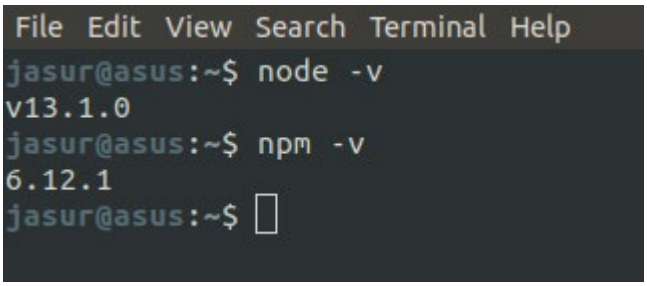

Figure 3 - Checking version of Node.js and NPM

3.4.4 Creation of React.js Application

Creating React.js app is not hard as the most people may think, they need some basic knowledge working with Command Prompt for Windows or Terminal for Linux and Mac. For that, it would call the package manager NPM and type a function that creates React.js application and at the end specify the name for the future app. However, creating the app does not stick in using NPM package manager, but also it may use Yarn and NPX. (ReactJS, 2020)

```
npx create-react-app my-app
cd my-app
npm start
```

Figure 4 - Creating ReactJS App

3.5 What is Virtual DOM?

DOM control is the core of the cutting edge, intuitive web. Shockingly, it is likewise a great deal slower than most JavaScript activities. This gradualness is exacerbated by the way that most JavaScript systems update the DOM significantly more than they need to.

A virtual DOM is a lightweight abstraction of the HTML DOM. You can think of it as a local in-memory copy of the HTML DOM. React uses it to do all computations necessary to render the state of a UI component. (Ved & Stoyan, 2017)

For instance, suppose that someone has a rundown that contains ten things. He or she scratches off the principal thing. Most JavaScript systems would reconstruct the whole

rundown. That is multiple times more work than would normally be appropriate! Just one thing changed, however the staying nine get modified precisely how they were previously.

Remaking a rundown is not a problem to an internet browser, however present-day sites can utilize tremendous measures of DOM control. Wasteful refreshing has become a difficult issue. To address this issue, the individuals at React promoted something many refer to as the virtual DOM.

In React, for each DOM object, there is a relating "virtual DOM object." A virtual DOM object is a portrayal of a DOM object, similar to a lightweight duplicate. A virtual DOM object has indistinguishable properties from a genuine DOM object, yet it comes up short on the genuine article is capacity to straightforwardly change what is on the screen. Controlling the DOM is moderate. Controlling the virtual DOM is a lot quicker, on the grounds that nothing gets drawn onscreen. Consider controlling the virtual DOM as altering a plan, rather than moving rooms in a genuine house.

At the point when it renders a JSX component, each and every virtual DOM object gets refreshed. This sounds fantastically wasteful, however the expense is unimportant in light of the fact that the virtual DOM can refresh so rapidly. When the virtual DOM has refreshed, at that point React contrasts the virtual DOM and a virtual DOM depiction that was taken just before the update. By contrasting the new virtual DOM and a pre-update rendition, React.js makes sense of precisely which virtual DOM objects have changed. This procedure is classified "diffing."

Once React knows which virtual DOM objects have changed, at that point React refreshes those articles, and just those items, on the genuine DOM. In this model from prior, React.js would be sufficiently shrewd to remake one verified rundown thing and leave the remainder of rundown alone. This has a major effect! React can refresh just the important pieces of the DOM. React.js’s notoriety for execution comes generally from this development. In rundown, this is what happens when it attempts to refresh the DOM in React:

1. The whole virtual DOM gets refreshed.
2. The virtual DOM gets contrasted with what it resembled before it refreshes itself. React makes sense of which articles have changed.

3. The changed articles, and the changed items just, get refreshed on the genuine DOM.
4. Changes on the genuine DOM cause the screen to change.

3.6 Props and state

State - is information kept up inside a component. It is neighborhood or claimed by that particular component. The part itself will refresh the state utilizing the setState work.

State is used so that a component can keep track of information in between any renders that it does. When you setState it updates the state object and then rerenders the component. (Abhishek & Akshat, 2019)

Props is simply shorthand for properties. Props are how components talk to each other and the data flow is immutable. Props are passed down the component tree from parent to children and vice versa. (Abhishek & Akshat, 2019)

The thing that matters is about which component claims the information. State is claimed locally and refreshed by the part itself. Props are possessed by a parent component and are perused as it were. Props must be refreshed if a callback work is passed to the kid to trigger an upstream change.

The condition of a parent component can be passed a prop to the youngster. They are referencing a similar worth, yet just the parent component can refresh it.

3.7 Stateful and stateless components

"Components are one of the pieces that make React, well, React! They are one of the primary ways you have for defining the visuals and interactions that make up what people see when they use your app. " (Chinnathambi, 2018)

The author wants to say that React application is consist of components that interact with each other. In React App , there are two types of components and they are stateless and stateful:

A stateless component is typically connected with how an idea is introduced to the client. It is like a capacity in that, it takes an info (props) and returns the yield (react component).

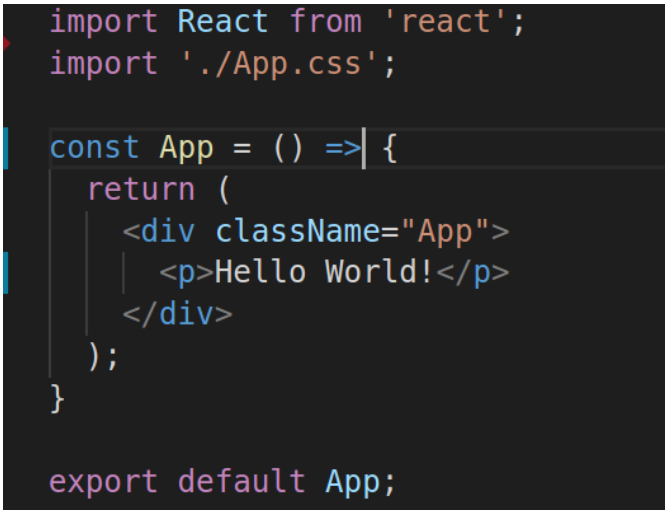

Figure 5 - Sample example of stateless component in ReactJS

A stateful component is constantly a class part. It is made by expanding the React.Component class. A stateful component is reliant on it is state question and can change it is own state. The component re-renders dependent on changes to it is state and may go down properties of it is state to kid components as properties on a props object.

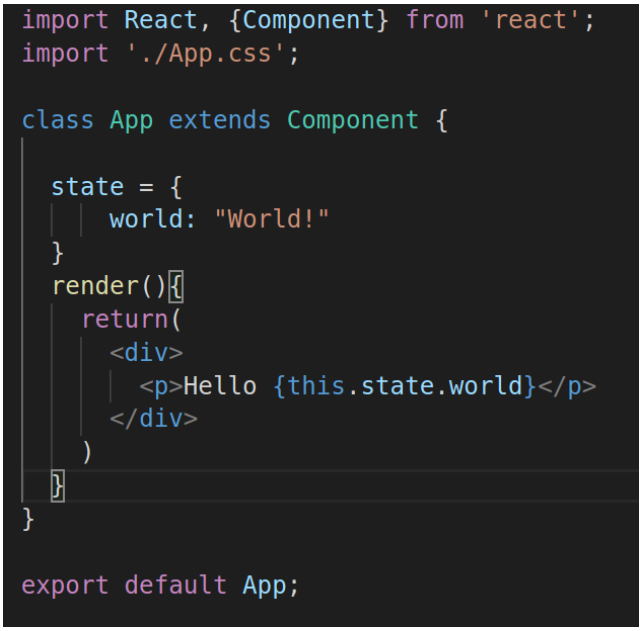

Figure 6 - Sample example of stateful component in React.js

As it be can differed in the stateless component it cannot find any state variable, any declaration of class.

3.8 What is Redux.js

Redux is an anticipated state compartment for JavaScript applications. It causes compose applications that carry on reliably, run in various situations (customer, server, and local), and are anything but difficult to test. What is more, it gives an extraordinary designer experience, for example, live code altering joined with a time traveling debugger.

"Redux is a solution that can be used to handle all kinds of state in React applications. It provides a single state tree object, which contains all application state. This is similar to what we did with the Reducer Hook in our blog application. Traditionally, Redux was also often used to store local state, which makes the state tree very complex." (Daniel, 2019)

Redux is accessible as a bundle on NPM for use with a module bundler or in a Node application (Redux, 2020)

```
npm install react-redux
```

Figure 7 - Installation of React Redux

3.8.1 Factors of Redux.js

In the Redux.js there are three main factors and they are action, reducer and store.

Action help to send information or data from React.js application to a main state which is located in a reducer. It usually imports in React App's component or class.

"Reducers handle the actions which describe the fact that something happened but managing the state of the application is the responsibility of the reducers. They store the previous **state** and **action** and return the **next state**" (Mehul & Harmeet, 2016). So, that means, the next state rewrites the previous one.

Store is the combination of these two factors such as action and reducer. It keeps the application state, it gives a chance to access state and update it.

3.9 React-Bootstrap

React-Bootstrap is the most popular Front-End framework for React library. Every part has been worked without any preparation as a genuine React component, without unneeded conditions like jQuery. As one of the most established React libraries, React-

Bootstrap has advanced and developed nearby React, settling on it a superb decision as UI establishment.

The best way of installation of React-Bootstrap is using NPM packages by typing following script. It can be implemented in Terminal (Linux and MacOS) or Command Prompt (Windows) (Bootstrap, 2020):

```
npm install react-bootstrap bootstrap
```

Figure 8 - Installation of React-Bootstrap

After installation, it has to be imported it in React App, by following script (Bootstrap, 2020):

```
import Button from 'react-bootstrap/Button';

// or less ideally
import { Button } from 'react-bootstrap';
```

Figure 9 - Importing to React App

4 Practical Part

4.1 Start to create

In practical part, would gone through process of implementing Simplex Method Algorithm in modern programming language such as JavaScript, using its library React.js, which is very popular among Web Developers. Firstly, it needs to create React Application by running Linux Terminal in this case with above mentioned procedure (Create React.js Application), how shown in below:

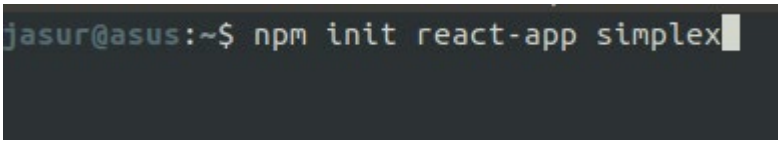

Figure 10 - Creating "Simplex" React.js App

It would install NPM packages with a specified directory. After installation it would give a hint how to run React.js application.

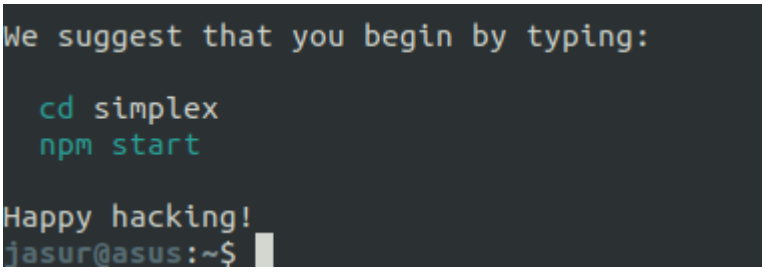

Figure 11 - Installation finished screen

Once a creation of React App is finished, it can be continued in code editor VS Code. On the editor, will be created new folder and it will contain two JavaScript files, **reducer.js** and **actions.js**. In the first file, there will be stored React state and switch statement. The switch statement contains set of actions that they will change behavior or value of the state. In the second JS file, there will be stored a set of actions that they will represent names of future functions that will change React state.

In index.js, which is located in main “simplex folder add Redux connection. In order to do that, Provider function will be imported from react-redux package, createStore from redux package and import reducer.js as reducer. Constant variable will be created which contains reducer that passes through createStore function. After all, in render function, the

Provider function will call a giving attribute store, that points to variable store and put the main function - <App/> into the Provider function. At that point, there was made a connection between Redux and the React app.

```
import {Provider} from 'react-redux';
import {createStore} from 'redux';
import reducer from './store/reducer';

const store = createStore(reducer)

ReactDOM.render(<Provider store = {store}><App />
    </Provider>, document.getElementById('root'));
```

Figure 12 - Connection of Redux.js with React.js

4.1.1 Storage

In that section will be described steps pf creating Redux State in reducer.js file. In order to do that, there must be discussed some questions such as how many numbers of variables is needed for Objective function and how many constraints will be provided.

As the first step, it is a good way to construct application's state. For that is needed several variables:

1. objective – is a storage of values of variables in objective function
2. variables – is an array where stores number of variables of objective function, for example x_1, x_2, x_3 and so on
3. constraints – is the most important object where it stores:
    a. namedVectors – is the array where values of each variable in specified constraint are stored
    b. sign – it might be $\leq$, $\geq$ and =. They are defined as a string
    c. rhs – is a value of Right-Hand Side of a specified constraint. It is defined in number
    d. bv is an object where it stores variable (x_1, x_2, x_3 …) as a string and its value. For example {x_1: 5}

4. isOptimal – is defines whether objective function is needed to be maximized or minimized

The above-mentioned specification of the objects of the state are attached with following picture:

```
const initialState = {
    variables: [],
    objective: [],
    constraints: [
        {
        namedVector: [],
        sign: '<=',
        rhs: 0,
        bv: {},
    }
    ],
    isOptimal: 'Max',
}
```

Figure 13 - Initial state of "Simplex" App

4.2 Constructor

In that section will be discussed creating forms that user will put values into input fields. Each form will have own stateless component and all of them will be gathered in one parent class, which names Constructor. All stateless component will be supplied by functions that they will change the main state, which is in reducer.js. These stateless components are:

1. Evaluation.js – is a simple form that would tell how many variables and constraints the user needs
2. ObjectiveFunction.js – is a form that builds simple objective function with given variables
3. Constraints.js - is a form that builds number of constraints with given number in the first function

There will be created Constructor.js class and keep all functions and stateless components that are related to construct forms into one class. By forms it meant input fields where user can give values to variables, choose if a goal of the problem is maximization or minimization, pick signs for constraints such as greater or equal, less than or equal and equal.

One of the main thing is getting values from the main state, to be clear from reducer.js, that is why action.js would be imported. It is like a bridge between Constructor.js and reducer.js:

```
import React, { Component } from 'react';

import { connect } from 'react-redux'
import * as actionTypes from '../store/actions'
```

Figure 14 - Main imports in Costructor class

There also be imported connect function from react-redux package in order to link mapStateProps and mapDispatchToProps to Constructor.js. The mapStateProps will be responsible for keeping the objects of the main state from reducer's state and the mapDispatchToProps will hold set of functions that will be used in the parent class and pointed to reducer function.

The first stateless component will have two input fields in order to define how many variables they need for objective function and the number of constraints, in order to build a matrix. In this case, they will use Bootstrap framework to build forms. This stateless component will get value from each input field and pass them to Redux.js. As it is shown, React packages and Form builder, Column, Button components from React-Bootstrap were imported. After that, there was created constant function where retrieves props from parent class which is Constructor one. Then, it builds a form where are three rows and they are one of them is for specifying number of objective function, to be clear, the user will specify how many variables are in objective function. The second row is saying about the number of constraints, where also user has access to specification. The last row is for a button, that submits result and goes to another step, which will be discussed later.

In the parent class will be created additional three functions that they will pass values to Redux, then Redux functions will change in the main state.

```
render() {
    return (
        <div>
            <Evaluation
                addingVar={this.addingVar}
                onAdding={this.addingConstr}
                submitting={this.onSubmitHandler} />
```

Figure 15 -Import of Evaluation component into the Constructor class

```
    addingVar = (event) => {
        event.preventDefault();
        this.props.onAddingVar(event.target.value)
    }

    addingConstr = (event) => {
        event.preventDefault();
        this.props.onAddingConst(event.target.value)
    }

    onSubmitHandler = (event) => {
        event.preventDefault();
        this.setState(prevstate => ({
            display: !prevstate.display,
        }));
    }

    render() {
        return (
            <div>
                <Evaluation
                    addingVar={this.addingVar}
                    onAdding={this.addingConstr}
                    submitting={this.onSubmitHandler} />
```

Figure 16 - Additional functions in the Constructor class

```
const mapDispatchToProps = dispatch => {
    return {
        onAddingVar: (value) => dispatch({ type: actionTypes.ADD_VAR, val: value }),
        onAddingConst: (value) => dispatch({ type: actionTypes.ADD_CONST, val: value }),
```

Figure 17 - Dispatching fuctions for Evaluation component

```
const reducer = (state = initialState, action) => {
    switch (action.type) {

        case actionTypes.ADD_VAR:
            if (state.objective.length === 0) {
                for (let i = 0; i < action.val; i++) {
                    state.objective.push(0)
                    state.variables.push(`x${i + 1}`)
                }
            } else state.objective.length = 0
            state.constraints.map((value, index) => {
                for (let i = 0; i < action.val; i++) {
                    value.namedVector.push(0)
                }
            })

            break;

        case actionTypes.ADD_CONST:
            for (let i = 0; i < action.val; i++) {
                state.constraints.push({
                    namedVector: [],
                    sign: '<=',
                    rhs: 0,
                    bv: {},
                })
            }

            break;
```

Figure 18 - Reducer function cases for the Evaluation component

```
import React from 'react';
import { Form, Col, Button} from 'react-bootstrap';


const evaluation = (props) => {
    return (
        <Form>
            <Form.Row className="justify-content-md-center">
                <Form.Label column lg={2}>Number of variables in Objective Function</Form.Label>
                <Col md ='auto'>
                    <Form.Control placeholder="number" onChange={props.addingVar}/>
                </Col>
            </Form.Row >
            <Form.Row className="justify-content-md-center" md="3">
                <Form.Label column lg={2}>Number of Constraints</Form.Label>
                <Col md ='auto'>
                    <Form.Control placeholder="number" onChange={props.onAdding}/>
                </Col>
            </Form.Row>
            <Form.Row className="justify-content-md-center">
                    <Button variant="primary" type="submit" size="sl" onClick={props.submitting}>
                        OK
                    </Button>
                </Form.Row>
        </Form>
    )
}
export default evaluation;
```

Figure 19 - Evaluation component

As it shown in browser how it looks like in a browser:

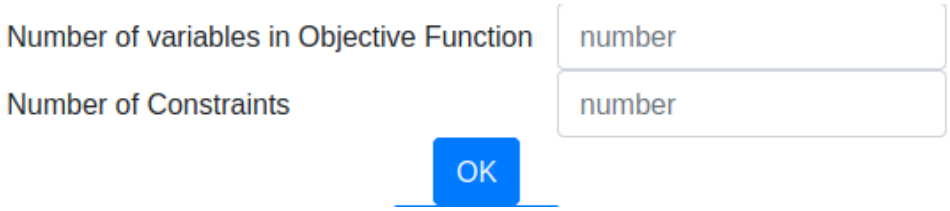

Figure 20 - The Evaluation component in a browser

The second stateless component, to be more precisely it is ObjectiveFunction.js, it will describe about constructing objective function itself as it was written before. There will be imported React from react packages and set of Bootstrap components that would help to construct Objective function. After that, by using map built-in function of JavaScript, it would take the number of variables that user specified in the Evaluation function and spread them into form. The last column describes a selection box where the user has a chance to whether objective function is needed to be maximized or minimized.

From the parent class (Constructor), it will pass one array which contains set of variables with values of them and two functions help to change values of objective array.

```
import React from 'react';
import { Form, Row, Col} from 'react-bootstrap';

const objectiveFunction = (props) => {

    return <Form>
            <Row className="justify-content-md-center">
                <p>Z =</p>
                {props.objective.map((value, index)=>{
                    return <Col md ='auto' key={index}>
                        <Form.Control
                            placeholder={`x${index + 1}`}
                            onChange={(event) => props.onChangeObjFunc(event, index)}/>
                    </Col>
                })
                }
                <Col sm lg="2">
                    <Form.Control as="select" onChange={props.onChangingOpt}>
                        <option value="select">Select an Option</option>
                        <option value="Max">Maximization</option>
                        <option value="Min">Minimization</option>
                    </Form.Control>
                </Col>
            </Row >
        </Form>
}
export default objectiveFunction;
```

Figure 21 - Objective function component

```
<ObjectiveFunction
    objective={this.props.objective}
    onChangeObjFunc={this.props.onChangeObjFunc}
    onChangingOpt={this.props.onChangeOpt} />
```

Figure 22 - Import of ObjectiveFunction component into the Constructor class

```
const mapDispatchToProps = dispatch => {
    return {
        onAddingVar: (value) => dispatch({ type: actionTypes.ADD_VAR, val: value }),
        onAddingConst: (value) => dispatch({ type: actionTypes.ADD_CONST, val: value }),

        onChangeObjFunc: (event, id) => dispatch({ type: actionTypes.CHANGE_OBJ,
            value: Number(event.target.value), index: id }),
        onChangeOpt: (event) => dispatch({ type: actionTypes.CHANGE_OPT,
            value: event.target.value }),
```

Figure 23 - Dispatching function for ObjectiveFunction component

```
case actionTypes.CHANGE_OBJ:
    state.objective[action.index] = action.value
    break;

case actionTypes.CHANGE_OPT:
    return {
        ...state,
        isOptimal: action.value
    }
```

Figure 24 - Reducer function cases for ObjectiveFunction component

One of the functions onChangeObjFunc in the Constructor class will take value and pass it to CHANGE_OBJ case in reducer function. The CHANGE_OBJ case will change value of the variable in objective function according its index. The second function will change type of optimal solution whether it is minimization or maximization.

The third stateless component (Constraints.js) is about creating forms for constraints by giving number of constraints that the user specified in the Evaluation component. Here it takes number of constraints and number of variables from the state of Redux. Then there is used built-in function which name is map(). First it maps through the amount of constraints and gives the certain number of variables.

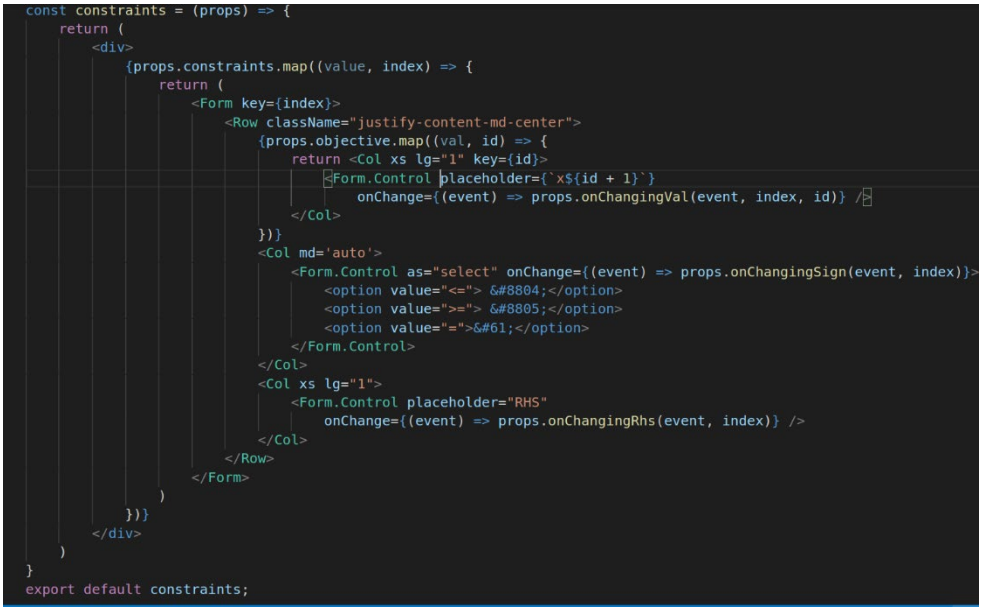

Figure 25 - Constraints component

From the parent class (Constructor), it will pass five props and they are:

- objective – array which contains a set of variables with values of them.
- constraints – is an object which almost everyhting beginning from vectors and signs.
- onChangeConstr – a function that changes the value of respective value which is user points to. In reducer function it is CHANGE_CONST
- onChangeRhs – a function that changes value of the Right-Hand Side of respective row. In reducer function it is CHANGE_RHS
- .onChangeSign – a function that changes the type of sign in respective row. In reducer function it is CHANGE_SIGN

```
    <Constraints
        objective={this.props.objective}
        constraints={this.props.constraints}
        onChangingVal={this.props.onChangeConstr}
        onChangingRhs={this.props.onChangeRhs}
        onChangingSign={this.props.onChangeSign} />
</div>
```

Figure 26 - Import of Constraints component into the Constractor class

```
    onChangeConstr: (event, id, key) => dispatch({ type: actionTypes.CHANGE_CONST,
        value: Number(event.target.value), index: id, key: key }),
    onChangeSign: (event, id) => dispatch({ type: actionTypes.CHANGE_SIGN,
        value: event.target.value, index: id }),
    onChangeRhs: (event, id) => dispatch({ type: actionTypes.CHANGE_RHS,
        value: Number(event.target.value), index: id })
}
```

Figure 27 - Dispatching function for Constraints component

```
case actionTypes.CHANGE_CONST:
    let currentConst = state.constraints[action.index].namedVector
    currentConst[action.key] = action.value
    break;

case actionTypes.CHANGE_RHS:
    state.constraints[action.index].rhs = (action.value)
    break;

case actionTypes.CHANGE_SIGN:
    state.constraints[action.index].sign = (action.value)
    break;
```

Figure 28 - Reducer function cases for Constraints component

As the last thing in the parent class Constructor.js, there will be internal state with name display. It would control which stateless component will render first. It works like that, the first component will be rendered is Evaluation component, when it finishes and user clicks on button “OK”, then will be rendered other two components. How it shown on the following picture:

```
render() {
    return (
        <div>
            {!this.state.display &&
                <Evaluation
                    addingVar={this.addingVar}
                    onAdding={this.addingConstr}
                    submitting={this.onSubmitHandler} />}

            {this.state.display && <ObjectiveFunction
                objective={this.props.objective}
                onChangeObjFunc={this.props.onChangeObjFunc}
                onChangingOpt={this.props.onChangeOpt} />}

            {this.state.display && <Constraints
                objective={this.props.objective}
                constraints={this.props.constraints}
                onChangingVal={this.props.onChangeConstr}
                onChangingRhs={this.props.onChangeRhs}
                onChangingSign={this.props.onChangeSign} />}
        </div>
    );
}
```

Figure 29 - Finished version of Constractor class

4.3 Calculation

In this section, there will be worked on calculation of the given values. A parent component Calculation.js will be created. In this file will be calculation of test of optimality, test of feasibility and Jordan elimination. Before doing that, the parent class would be connected to Redux.js. The previous section was more about creating user interface, but in this section, it will be oriented more what runs behind a calculation and what involved in it. As in before section, it needs import some necessary components, that are mentioned in a picture:

```
import React, { Component } from 'react';
import { connect } from 'react-redux'
import * as actionTypes from '../../store/actions'
```

Figure 30 – Main imports for Calculation class

After importing these packages there should fetch some objects from the main state and create four functions that would dispatch to the reducer function.

```
const mapStateToProps = state => {
    return {
        objective: state.objective,
        isOptimal: state.isOptimal,
        constraints: state.constraints,
        variables: state.variables
    };
};

const mapDispatchToProps = dispatch => {
    return {
        onSubmitHandler: (event) => dispatch({ type: actionTypes.SUBMIT, e: event }),
        onNewArr: (array, index) => dispatch({ type: actionTypes.NEW_ARRAY, arr: array, id: index }),
        onNewBasicVariable: (row, column) => dispatch({ type: actionTypes.NEW_BASIC_VARIABLE, row: row, column: column }),
        onNewRhs: (array) => dispatch({ type: actionTypes.NEW_RHS, arr: array }),

    }
}
export default connect(mapStateToProps, mapDispatchToProps)(Calculation);
```

Figure 31 - Connection between Calculation class and reducer function

Several stateless components will be created, such as:

- Test of the optimality (a calculation of Zj – Cj)
- Test of the feasibility (omega test)
- Jordan Elimination
- Building a table

The first part of Zj-Cj component will focus on the calculation of Zj-Cj. It means that it would multiply each column by values of basic variables, then sum them up, it would give

just Zj, then it would subtract from Cj which is a coefficient of column. This component is like a simple function that needs two parameters which are constraints and objective from the mapStateToProps function. All this process is implemented in the following stateless component:

```
const zjMinusCjCalc = (constraints, objective) => {
    let sumZj = [];
    let zJArr = [];
    let zjMinusCjArr =[];
    let calcRhs =0;

    constraints.map(current => {
        let calcZj = [];
        let currentBV = Object.values(current.bv);

        //Multiplication of RHS x BV
        calcRhs += currentBV*current.rhs

        //Multipication of vector x BV
        current.namedVector.map(val => {
            let calc = 0;
            calc = currentBV * val
            return calcZj.push(calc)
        })
        zJArr.push(calcZj);

        //Summing m x n matrix
        return sumZj = zJArr.reduce((r, a) => a.map((b, i) => (r[i] || 0) + b), []);
    })
    //It calculates sum of Zj - Cj
    sumZj.map((value, index) => {
        return zjMinusCjArr.push(value - objective[index]);
    })
    zjMinusCjArr.push(calcRhs)
    return zjMinusCjArr
}
export default zjMinusCjCalc;
```

Figure 32 – Calculation of Zj-Cj

Here the function is assigned to zJcJ variable, because of simplicity and it will be used in the future with other functions.

```
zJcJ = zjMinusCjCalc(this.props.constraints, this.props.objective)
```

Figure 33 - Variable of the Zj-Cj

The second part of ZJ-Cj component is about a testing of Zj-Cj for optimality. This function needs two parameters which are from mapsStateToProps function and variable zJcJ. In this function, it defines the goal of objective function, then chooses the most negative or most positive number and returns index of this number, if it does not satisfy condition then objective function is reached its optimality. Moreover, the function signed to pivotColumn variable, because it says the goal of the function which defines a pivot column.

```
const optimalityTest = (zjMinusCjArr, isOptimal) => {

    //We get rid of sum of RHS
    let zJMinusCj = [...zjMinusCjArr]
    zJMinusCj.pop();

    switch (isOptimal) {
        case 'Min':
            let valueMax = Math.max(...zJMinusCj)
            let indexMax = zJMinusCj.indexOf(valueMax)
            return (valueMax > 0
                    ? indexMax
                    : 'The solution is optimal'
            )

        case 'Max':
            let valueMin = Math.min(...zJMinusCj)
            let indexMin = zJMinusCj.indexOf(valueMin)
            return (valueMin < 0
                ? indexMin
                : 'The solution is optimal'
            )
        default:
            break;
    }
}

export default optimalityTest;
```

Figure 34 - Test of optimality function

```
zJcJ = zjMinusCjCalc(this.props.constraints, this.props.objective)
pivotColumn = optimalityTest(zJcJ, this.props.isOptimal)
```

Figure 35 - Assigning variable to the test of optimality

The second component which is test of feasibility that takes two components and they are constraints and pivotColumn variable. It will divide each Righ Hand Side value to pivot column values. After that, it will choose the lowest positive number, at the end of function it returns index of row and value of it, as shown in pictures:

```
const feasibilityTest = (constraints, id) => {
    let calc = [];
    let positive = [];
    let minPositive;

    //It takes each RHS and divides to respective pivot column
    constraints.map(current => {
        return calc.push(current.rhs / current.namedVector[id]);
    })

    //It sorts omega to positive numbers
    calc.map(value => { return value > 0 ? positive.push(value) : console.log('feasib') })

    //It takes the index of minimum positive number
    minPositive = calc.indexOf(Math.min(...positive))

    return [minPositive, calc]
}

export default feasibilityTest;
```

Figure 36 - Test of feasibility

```
zJcJ = zjMinusCjCalc(this.props.constraints, this.props.objective)
pivotColumn = optimalityTest(zJcJ, this.props.isOptimal)
pivotRow = feasibilityTest(this.props.constraints, pivotColumn)
```

Figure 37 - Assigning variable to the test of feasibility

Here comes a key point, if the objective function reached its optimality or not, there will be created if condition where checks if the variable pivotColumn is “The solution is oltimal”, then it leaves a condition and there will be rendered a table(will be discussed later). If not, then it renders the table and tries to optimize the objective function

```
if (pivotColumn === 'The solution is optimal') {
    return <div>
        {finalTable(this.props.variables, this.props.objective, this.props.constraints, zJcJ)}
    </div>
}
else {
```

Figure 38 - If condition when the solution is optimal

Else condition is more progressive and more complicated, because it involves several stateless components which they will be discussed later. However, it is more interesting to create the table that shows the result. It involves some imports such as react package, Table component from react-bootstrap and CSS styling.

```
import React from 'react';
import { Table } from 'react-bootstrap';
import './TableBuilder.css'


const tableConstructor = (props) => {

    let rowToDisplay = props.constraints[props.pivotRow].namedVector[props.pivotColumn]

    // Printing each variable of OF
    const keysOfObjFunc = props.variables.map((val, id) => { return <td key={id}>{val}</td> })

    //Printing each value of OF
    const valuesOfObjFunc = props.objective.map((val, id) => { return <td key={id}>{val}</td> }

    //Printing each value of constraint
    const constraint = props.constraints.map((current, index) => {
        return (
            <tr key={index}>
                {Object.values(current.bv)
                    .map((val, id) => { return <td key={id}>{val}</td> })
                }
                {Object.keys(current.bv)
                    .map((val, id) => { return <td key={id}>{val}</td> })
                }
                {current.namedVector.map((val, id) => {
                    if (id === props.pivotColumn) {
                        return (
                            <td key={id} className='pivot'>{val}</td>)
                    } else if (index === props.pivotRow) {
                        return (<td key={id} className='pivot'>{val}</td>)
                    } else return <td key={id}>{val}</td>
```

Figure 39 - Building a table part 1

```
                })
                }
                {index === props.pivotRow ? <td className='pivot'>{+(current.rhs).toFi
                {index === props.pivotRow ? <td className='pivot'>{+(props.omega[index

            </tr>
        )
    })

    const optimality =
        <tr>
            <td>Optimality Test</td>
            <td>Zj - Cj</td>
            {props.zjMinusCjArr
                .map((val, id) => { return <td key={id}>{+(val).toFixed(2)}</td> })}
        </tr>

    return <div className='table'>
        <Table responsive>
            <thead>
                <tr>
                    <td></td>
                    <td></td>
                    {valuesOfObjFunc}
                </tr>
                <tr>
                    <td>Values of BV</td>
                    <td>BV</td>
                    {keysOfObjFunc}
                    <td>RHS</td>
                    <td>Omega</td>
                </tr>
            </thead>
```

Figure 40 - Building a table part 2

```
                </tr>
            </thead>
            <tbody>
                {constraint}
                {optimality}
            </tbody>
        </Table>
        <p>Pivot Row is R{props.pivotRow + 1}</p>
        <p>Pivot Column is C{props.pivotColumn + 1}</p>
        <p>Pivot Value {rowToDisplay}</p>
    </div>
}
export default tableConstructor;
```

Figure 41 - Building a table part 3

After rendering the table, there will be used Jordan Elimination in order to solve a problem and it will change values in state with Jordan Elimination. For that, there is the Jordan Elimination is divided into two parts: the first part will calculate the Right-Hand Side and the second one is for calculating vectors of the Objective function.

On the first part of the Jordan Elimination function it needs additional two parameters, except constraints and they are index of pivot row and pivot column. After calculation it returns array of new Right-Hand Side values and it passes to the reducer function where it changes in the main state of the app.

```
const jordanRhs = (constraints, pivotRow, pivotColumn) => {
    let rhsArr = [];
    let calcRhs = constraints[pivotRow].rhs /
        constraints[pivotRow].namedVector[pivotColumn]
    rhsArr[pivotRow] = calcRhs

    //Calculates other rows
    constraints.map((current, index) => {
        if (index !== pivotRow) {
            rhsArr[index] = current.rhs - current.namedVector[pivotColumn] * calcRhs
        }
    })
    return rhsArr
}

export default jordanRhs;
```

Figure 42 - Calculation of new RHS

The second Jordan Elimination takes the same parameters as the first one and at the end it returns new array vectors of constraints in new table.

```
const elimination = (constraints, pivotRow, pivotColumn) => {
    let row = constraints[pivotRow].namedVector
    let rowArr = [];
    let pivotValue = row[pivotColumn]
    //Uses jordan elimination to calculate pivot row
    row.map((value, index) => {
        return row[index] = +(value / pivotValue).toFixed(2)
    })

    //Calculates other rows
    constraints.map((current, index) => {
        if (index !== pivotRow) {
            pivotValue = current.namedVector[pivotColumn]
            current.namedVector.map((val, id) => {
                return current.namedVector[id] =
                    +(val -  pivotValue * row[id]).toFixed(2)
            })
        }
        return rowArr.push(constraints[index].namedVector)
    })

    return rowArr

}

export default elimination;
```

Figure 43- Calculation of new vectors

The first two connecting functions such as onNewRhs and onNewArr or NEW_RHS and NEW_ARRAY in reducer function they are responsible for Jordan Elimination. The third function‘s job is to renew basic variables object

```
const mapDispatchToProps = dispatch => {
    return {
        onNewRhs: (array) => dispatch({ type: actionTypes.NEW_RHS, arr: array }),
        onNewArr: (array, index) => dispatch({ type: actionTypes.NEW_ARRAY,
            arr: array, id: index }),
        onNewBasicVariable: (row, column) => dispatch({ type: actionTypes.NEW_BASIC_VARIABLE,
            row: row, column: column }),
    }

}
```

Figure 44 - Connecting functions between Claculation class and reducer function

```
case actionTypes.NEW_RHS:
    action.arr.map((value, index) => {
        state.constraints[index].rhs = value
    })
    break;

case actionTypes.NEW_ARRAY:
    state.constraints[action.id].namedVector = action.arr;
    break;

case actionTypes.NEW_BASIC_VARIABLE:
    let newBasicVar = {}
    newBasicVar[state.variables[action.column]] = state.objective[action.column]
    state.constraints[action.row].bv = newBasicVar

    break;
```

Figure 45 - Reducer function‘s switch cases for new values

5 Results and Discussion

In the previous part the author mentioned the implementation of Simplex Method in React.js. They were implemented some classes and functions and the main state was kept in reducer function which were used Redux tool. The following diagram describes how it would look like in UML diagram.

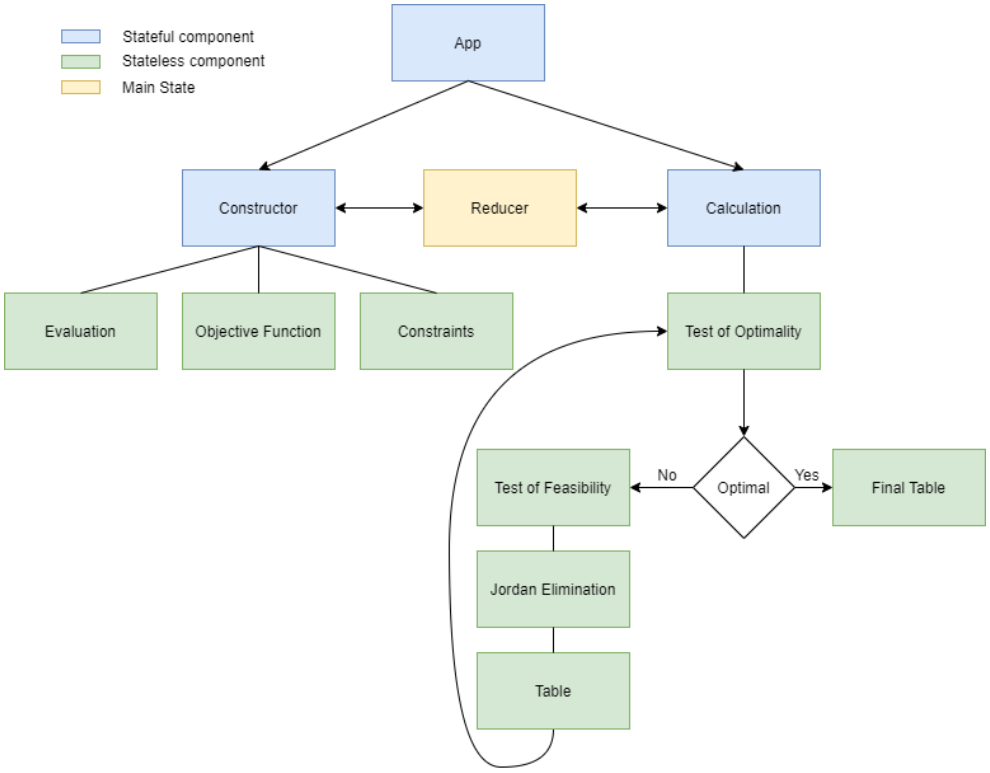

Figure 46 - UML diagram

When the user opens the Simplex Method page, he will see Constructor class where the user has to fill out number of variables in objective function and number of constraints.

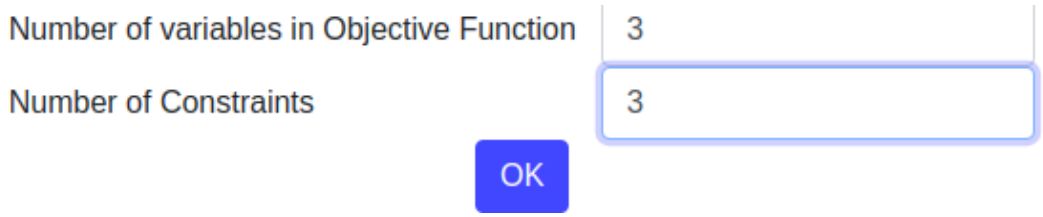

Figure 47 - Evaluation component of the Constructor class in a browser

When the user clicks on button “OK”, then comes the second step of Constructor class which are ObjectiveFunction and Constraints components. There the user will fill out coefficient of each variable and choose type of solution:

Figure 48 - ObjectiveFuction and Constraints components of Constructor class in a browser

When the user hits button “Submit”, then App class calls Calculation class and starts calculating with given values. After test of optimality, React App checks if the solution is optimal, if not then continues calculating further by calling test of feasibility, Jordan Elimination and again does test of optimality.

Now it is time to see the React application in action, let’s suppose there are 3 variables and 3 constraints following constraints:

7x1 + 8x2 + 9x3 <= 15

4x1 + 5x2 + 6x3 <= 20

1x1 + 2x2 + 3x3 <= 25

Z = 10x1 + 11x2 + 12x3 ->MAX

The Simplex React App constructs table with addition of slack variables. It tells that solution is not optimal by checking test of optimality (Zj-Cj), from this it finds pivot column. Then, it makes test of feasibility and finds pivot row. From this two information, it chooses pivot value and it continues further.

Coefficient of variables		10	11	12	0	0	0		
Coefficient of BV	BV	x1	x2	x3	s1	s2	s3	RHS	Omega
0	s1	7	8	9	1	0	0	15	1.67
0	s2	4	5	6	0	1	0	20	3.33
0	s3	1	2	3	0	0	1	25	8.33
	Zj - Cj	-10	-11	-12	0	0	0	0	

The solution is not optimal

Pivot column - **column 3**, least negative value in **test of optimality**

Pivot row - **row 1**, least positive value in **test of feasibility**

Pivot value - **9**

Figure 49 - TableBuilder component in a browser

After some steps, there pops up the result, it says that the solution is optimal. It gives some result and they are:

x1 = 2.14

s2 = 11.45

s3 = 22.86

Z = 21.4

Coefficient of variables		10	11	12	0	0	0	
Coefficient of BV	BV	x1	x2	x3	s1	s2	s3	RHS
10	x1	1	1.14	1.28	0.14	0	0	2.14
0	s2	0	0.44	0.87	-0.56	1	0	11.45
0	s3	0	0.86	1.72	-0.14	0	1	22.86
	Zj - Cj	0	0.4	0.8	1.4	0	0	21.4

The solution is optimal

x1 = 2.14

s2 = 11.45

s3 = 22.86

Value of objective function Z = 21.4

Figure 50 -FinalTable component in browser

6 Conclusion

In the theory part, the author explained tools that he used in order to implement Linear programming guided solver using JavaScript basics and combination with its library which is called React.js. In this part was mentioned how looks like Linear programming table, how JavaScript programming language works, what is React.js and additional features such as React - Redux and React – Bootstrap.

In the practical part, all above mentioned tools have been used. The author created three stateless components and eight stateful components and they cooperate with each other. The App class (stateful component) is like a parent class which keeps two following classes: Constructor and Calculation which one of them fetch data from the user, another one is calculates and builds table

In conclusion, JavaScript's library React.js is claimed that it is simple, convenient and easy for creating webpages with complex algorithm such as Linear programming problem. Creating web pages with calculation of mathematical solution could be beneficial not always for programmers that starting to develop, also it is useful for the users (students) that learn solving algebraic problems. It might be more useful when a student understands theory of Simplex Method and implement it in code format, because he/she feels feelings what other students may feel when they solve such kind of problems and it makes the React App more useful for student communities.

7 References

Abhishek, N., & Akshat, P. (2019). *React Native for Mobile Development: Harness the Power of React Native to Create Stunning iOS and Android Applications.* New York, United States: Apress.

Azat, M. (2017). *React Quickly: Painless web apps with React, JSX, Redux, and GraphQL.* Shelter Island, New York, United States: Manning Publications.

Bootstrap, R. (2020). *React Bootstrap Introduction*. Načteno z React Bootstrap: https://react-bootstrap.github.io/getting-started/introduction/

Chinnathambi, K. (2018). *Learning React: A Hands-On Guide to Building Web Applications Using React and Redux, Second edition.* Boston, Massachusetts, United States: Addison-Wesley Professional.

Daniel, B. (2019). *Learn React Hooks.* Birmingham, United Kingdom: Packt Publishing.

Matt, F. (2019). *Professional JavaScript for Web Developers, 4th Edition.* Birmingham, England: Wrox.

Mehul, B., & Harmeet, S. (2016). *Learning Web Development with React and Bootstrap.* Birmingham, United Kingdom: Packt Publishing.

Node.js. (2020). *Download Node.js*. Načteno z Node.js: https://Node.js.org/en/download/

React.js. (2020). *Create a new React App - React*. Načteno z React - A JavaScript library for building user interfaces: https://React.js.org/docs/create-a-new-react-app.html

Redux. (2020). *Getting Started with Redux*. Načteno z Redux - A predictable state container for JavaScript apps: https://redux.js.org/introduction/getting-started

Ved, A., & Stoyan, S. (2017). *Object-Oriented JavaScript - Third Edition.* Birmingham, United Kingdom: Packt Publishing.