**Модуль StormGlass.**

Создадим модуль для получения метеорологических данных для точки разлива. Для этого будем использовать *fetch* запросы, которые были приняты в новом стандарте ECMAScript6 ES6 Языка JavaScript. Особенность *fetch* запросов является то, что результат работы будут *промисы*. При удачном выполнении это *resolve* ответ, при возникновении какой-то ошибки reject ответ. В данном модуле не будет ничего кроме данной функции, и поэтому пропишем, чтобы функция экспортировалась по умолчанию. На вход функция будет принимать координаты аварии и время для расчета.Рис.1

export default async function StormGlass(lat,lng,time){

Параметр Async указывает, что данная функция является асинхронной и является частью async/await комплекса. Асинхронность функций является вспомогательный инструментом и указывает на то, что данная функция должна возвращать результат, время расчета которого не известно. И прописывает варианты действия при удачном, и неудачном действии. Внутри функции Async есть возможность вызывать await функцию. Обратим внимание, что вызов функции await возможен исключительно внутри функции async.

Для получения данных будет отправляться запрос Get запрос на API сайта StormGlass.

Образец запроса выглядит таким образом.

let response = await fetch(`https://api.stormglass.io/v2/weather/point?lat=${lng}&lng=${lat}&params=${params}&start=${time}&end=${time}`, {

      headers: {

        'Authorization': ApiKey

      }

    });

Объявляем переменную с именем response, в которой запишем *промис* результата действия *fetch* запроса. Конструкция ${}, позволяет нам передавать переменную в строку. Обратим внимание, что конструкция передачи переменной в строку возможно исключительно внутри `` кавычек и только в новом стандарте ES6.

В параметр *lat* передается координаты X, которые будут передаваться на вход функции. В параметр *lng* координата Y, которые будут передаваться на вход функции. В параметр params будем передавать массив необходимых нам значений в виде строки.

const paramsArray = ['airTemperature','waterTemperature','windSpeed',

'windDirection','currentDirection','currentSpeed'];

Преобразуем данный массив в строку при помощи метода массивов ***join***.

const params = paramsArray.join();

Параметры start и end принимают начальное и конечное время для получения данных. В данном запросе будет передаваться входной параметр, и эти два параметра в данном случае будут совпадать. На вход данный параметр принимает время в формате ***UNIX Timestamp,*** ***URL Encoded ISO Formatted Timestamp.***

В headers заголовка fetch запрос будет принимать специальный API ключ, который получен при регистрации на сайте *StormGlass*. В соответствии тарифа у каждого ключа есть ограниченное число запросов.

В результате действия fetch запроса будет возвращен промис, который необходимо распарсить в формат JSON и записать результат в переменную с название json.

let json = await response.json();

В переменной json будет хранится массив данных, отсортированный от начального времени до конечного. В данном элементе массива хранится объект со всеми параметрами, которые мы указывали при запросе Рис.1.

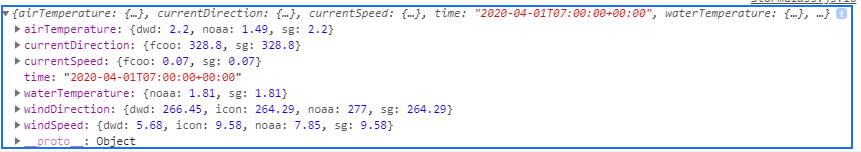


Рис. 1

Для различных параметров данные отправляются из разных источников, таких как *noaa(National Oceanic and Atmospheric Administration),dwd(Deutsche Wetter Dienst)* и для дальнейшего удобства получения значения из данного объекта просуммируем значения из различных источников и разделим на общее количество источник. Для этого пройдемся по объекту при помощи цикла for ..in и получим длину объекта при помощи метода Object.keys и узнаем полученную длину(length).

for(const j in paramsArray){

        let sum = 0;

        for (const z in weatherDate[paramsArray[j]]){

          sum = sum + weatherDate[paramsArray[j]][z];

        }

        let object\_length = Object.keys(weatherDate[paramsArray[j]]).length;

        newParams[paramsArray[j]] = sum/object\_length;

    }

В результате выполнения данного цикла получаем усреднённый массив значений в котором удобно получать параметры при обращении к свойству. В дальнейшем данный модуль возможно использовать для получения всех различных данных и не быть привязанным к их источнику. Результат работы цикла представлен ниже. Рис 2.

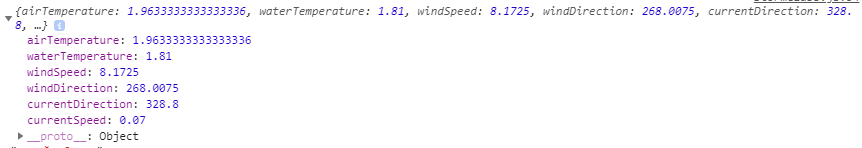


Рис. 2

**Модуль OilsAlias.**

В модуле OilsAlias будет производиться расчет больших и малых осей эллипса нефтяного пятна, на основе входных координат, времени разлива и массы разлива. И данная функция так же будет асинхронной и по умолчанию экспортироваться.

Импортируем в данный модуль функцию написанную в предыдущем модуле.

import StormGlass from './StormGlass';

export default async function OilAlias (Xcoord,Ycoord,emergencyTime,OilWeigth){

Добавим в библиотеку Math для математических расчетов метод перевода из градусов в радианы.

  Math.radians = function(degrees) {

        return degrees \* Math.PI / 180;

    };

Объявление переменной OilAliasDate, в которую будет записан результат расчетов в виде 10 массивов OilAliasIter[id, Xcoord, Ycoord, Rx, Ry, gradus\_veter] со значениями id, координат больших и малых осей, направление ветра. Объявление постоянных плотность воды, плотность нефти, температура Кельвина.

let OilAliasDate = [];

const plot\_vod = 1003.0;

const plot\_neft = 832;

const KelvinTemp = 273;

Объявление переменной IsoFormate в которой хранится время аварии в виде объекта времени с различными методами.

 let IsoFormate = new Date(emergencyTime);

Создание цикл на 10 прохождений. Объяление массива OilAliasIter и заполнение значением итерации, координатами X,Y. Знак + перед значениями координат, означает перевод в цифровой тип данных. Язык javascript является динамически типизированным языком и знак плюс перед переменными обеспечит стабильную типизацию.

for (let i = 1; i<11; i++){

let OilAliasIter = [];

OilAliasIter.push(i);

OilAliasIter.push(+Xcoord);

OilAliasIter.push(+Ycoord);

Объявление переменной time для вычисления времени с момента аварии.

*let time = 60\*i;*

Создание переменной *WeatherDate,* в которую будет записан результат действия

асинхронной функции *StormGlass*, при входных значениях координат и времени расчета в формате *ISO*.

*let WeatherDate = await StormGlass(Xcoord,Ycoord,IsoFormate.toISOString());*

Обратите внимание, что данная функция является асинхронной и конструкция await может быть выполнена исключительно внутри функций async.

В переменной WeatherDate*хранится* объект с параметрами метеорологических данных. Распарсим их и создадим аналоги значений для нашего расчета

let scor\_veter = WeatherDate.windSpeed;

let scor\_tech = WeatherDate.currentSpeed;

let gradus\_tech = WeatherDate.currentDirection;

let gradus\_veter = WeatherDate.windDirection;

Преобразуем температуры в кельвины.

let tmpVod = WeatherDate.waterTemperature + KelvinTemp;

let tmpOkruzh = WeatherDate.airTemperature + KelvinTemp;

Произведем расчет больших и малых осей Rx,Ry на основе метеорологических данных, массы нефти и времени разлива. На основе формулы(!).

let Ry=42.5\*(((plot\_vod-plot\_neft)/plot\_vod)\*\*(1.0/3.0))\*(OilWeigth\*\*(1.0/3.0))\*((time)\*\*0.25);

let Rx=Ry+0.15\*(scor\_veter\*\*(4.0/3.0))\*((time\*60)\*\*(3.0/4.0));

Расчет толщины нефти в данный момент времени с начало разлива let At=(Math.pi\*Rx\*Ry)/4.0;

Добавлние рассчитанных значений больших и малых осей эллипса, направления ветра в массив *OilAliasIter.*

OilAliasIter.push(Rx);

OilAliasIter.push(Ry);

OilAliasIter.push(gradus\_veter);

Если толщина пленки достигнет значения меньшего чем 10-6 м, то пятно разделится на многие микропятна и модель станет не актуальной. Поэтому нужно ввести условие if для выхода из цикла

if (At < 10e-6) break;

Теперь, когда мы рассчитали площадь, мы можем определить по формулам, приведенным выше кол-во испарившейся нефти за один час. Наша модель подразумевает расчет для 4 фракций с различными температурами кипения(*t)*. Рассчитаем молярные массы(*mol*) для каждой фракции по формуле (10) и молярную долю компонента (*molDolya*) от суммарной молярной массы(*molSumma*);

  const t1=373;

        const t2=413;

        const t3=453;

        const t4=493;

        let mol1=60+0.3\*(t1-273.15)+0.001\*(t1-273.15);

        let mol2=60+0.3\*(t2-273.15)+0.001\*(t2-273.15);

        let mol3=60+0.3\*(t3-273.15)+0.001\*(t3-273.15);

        let mol4=60+0.3\*(t4-273.15)+0.001\*(t4-273.15);

        let molSumma=mol1+mol2+mol3+mol4;

        let molDolya1=mol1/molSumma;

        let molDolya2=mol2/molSumma;

        let molDolya3=mol3/molSumma;

        let molDolya4=mol4/molSumma;

Для получения значения давление насыщенного пара каждой фракции *P* по формуле (10). А для этого сперва найдем коэффициенты *A* и *Используя* для их нахождения коэффициенты *alfa*.

  let alfa1=0.0125;

        let alfa2=0.058;

        let alfa3=0.0000482;

        let alfa4=0.0000005;

        let A1=2.9+alfa1\*t1-alfa2\*mol1+alfa3\*t1\*mol1;

        let A2=2.9+alfa1\*t2-alfa2\*mol2+alfa3\*t2\*mol2;

        let A3=2.9+alfa1\*t3-alfa2\*mol3+alfa3\*t3\*mol3;

        let A4=2.9+alfa1\*t4-alfa2\*mol4+alfa3\*t4\*mol4;

        let B1=10+(alfa4/t1)+(tmpVod/(mol1\*\*2));

        let B2=10+(alfa4/t2)+(tmpVod/(mol2\*\*2));

        let B3=10+(alfa4/t3)+(tmpVod/(mol3\*\*2));

        let B4=10+(alfa4/t4)+(tmpVod/(mol4\*\*2));

        let P1=2.72\*\*(A1\*(1-(((t1\*Math.log(t1))/(tmpVod\*Math.log(tmpVod)))\*Math.log(B1))));

        let P2=2.72\*\*(A2\*(1-(((t1\*Math.log(t2))/(tmpVod\*Math.log(tmpVod)))\*Math.log(B2))));

        let P3=2.72\*\*(A3\*(1-(((t1\*Math.log(t3))/(tmpVod\*Math.log(tmpVod)))\*Math.log(B3))));

        let P4=2.72\*\*(A4\*(1-(((t1\*Math.log(t4))/(tmpVod\*Math.log(tmpVod)))\*Math.log(B4))));

Последний коэффициент необходимый для расчета массы испарившейся фракции – это *коэффициент массопереноса*(*K*) который мы находим по формуле (9). И подставив все полученные выше значения найдем *массу испарившегося вещества*(*N*) каждой фракции и суммировав их найдем *суммарное количество испарившихся фракций(Nsum)* за час.

 let N1=((K\*molDolya1\*P1\*At\*3600)/(8.314\*tmpOkruzh));

        let N2=((K\*molDolya2\*P2\*At\*3600)/(8.314\*tmpOkruzh));

        let N3=((K\*molDolya3\*P3\*At\*3600)/(8.314\*tmpOkruzh));

        let N4=((K\*molDolya4\*P4\*At\*3600)/(8.314\*tmpOkruzh))

        let Nsum=(N1\*mol1+N2\*mol2+N3\*mol3+N4\*mol4)/plot\_neft;

Аналогично найдем суммарное количество растворенной нефти *(Rsum).* И единственный коэффициент, который необходимо рассчитать это *растворимость компонента(S).* В итоге количество нефти, оставшееся после испарения и растворения будет равняться, вычитанию из *начальной массы*(*massaRozliva*) суммарное количество испарившихся(*Nsum)* ирастворенное количество нефти *(Rsum).*

let S=(100e-4)\*2.72\*\*(0.1\*(i\*3600));

        let R1=3e-6\*molDolya1\*S\*At\*3600\*mol1;

        let R2=3e-6\*molDolya2\*S\*At\*3600\*mol2;

        let R3=3e-6\*molDolya3\*S\*At\*3600\*mol3;

        let R4=3e-6\*molDolya4\*S\*At\*3600\*mol4;

        let Rs=R1+R2+R3+R4;

        let Rsum=Rs/plot\_neft;

        OilWeigth=OilWeigth-Nsum-Rsum;

Рассчитаем перемещение центра пятна за время *time* по формуле перемещения (7). (рис 3.13)

Xcoord=+Xcoord+((Math.cos(gradus\_veter)\*scor\_veter\*0.03\*time)/111120)+((Math.cos(gradus\_tech)\*scor\_tech\*1.0\*time)/111120);

Добавляем массив полученных значений *OilAliasIter* для данного периода после разлива в массив всех итераций *OilAliasDate.*

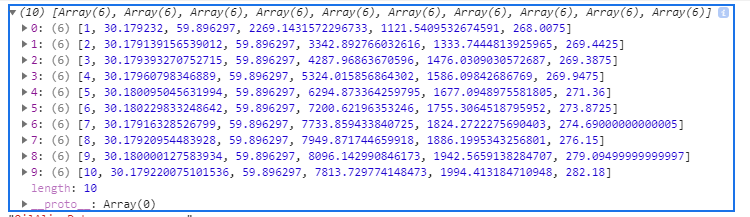
OilAliasDate.push(OilAliasIter);

Увеличим время на один час. Время для запроса метеорологических данных при помощи fetch запроса в API StormGlass будет увеличиваться до тех пор пока не пройдет все 10 итераций цикла.

IsoFormate = new Date(IsoFormate.setHours(IsoFormate.getHours()+1));

После прохождения 10 итераций цикла получен массив OilAliasDate

из 10 массивов OilAliasIter содержащих все необходимые данные для отрисовки.



**Модуль CalculateEllipse**

Модуль *CalculateEllipse* будет рассчитывать координаты эллипса на основе центральной точки и больших, малых осей и угла поворота. Результатом действия данного модуля будет массив точек с координатами который в последствии будет преобразован в полигон и добавлен на карту в виде графики.

На вход данная функция принимает X, Y координаты, большую и малую ось, угол поворота. Пропишем автоматический экспорт данной функции, т. к. это единственная функция внутри данного модуля.

export default function calculateEllipse(x,y,MajorAlias,MinorAlias,angle){

Объявим переменную meterLon, которая является соотношением метра к единице градусов.

 const meterLon = 0.0000017884826245207995;

Входные параметры передают значения большой и малой оси в виде метров, поэтому необходимо преобразовать их в координаты перемножив с константой meterLon.

  let beta=-angle\*(Math.PI/180);

    let sinbeta=Math.sin(beta);

    let cosbeta=Math.cos(beta);

    for(let i=0;i<360;i+=360/steps){

      let alpha=i\*(Math.PI/180);

      let sinalpha=Math.sin(alpha);

      let cosalpha=Math.cos(alpha);

      let X = x + (MajorAlias \* cosalpha \* cosbeta - MinorAlias \* sinalpha \* sinbeta);

      let Y = y + (MajorAlias \* cosalpha \* sinbeta + MinorAlias \* sinalpha \* cosbeta);

      let array=[X,Y];

      points.push(array);

    }

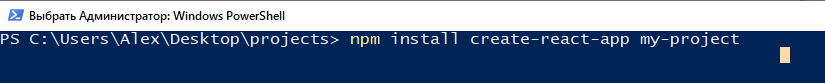
    // console.log(points);

    return (points);

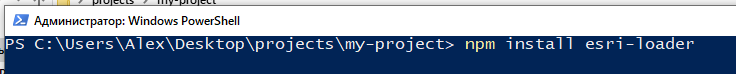
Данная функция возвращает массив точек, которые будут использованы для отрисовки эллипсов.

**Модуль MapView.js.**

Для отрисовки данных создадим одностраничное приложение SPA c помощью настроенной среды create-react-app. Более подробную информацию можно найти на официальном сайте React. Установим среду при помощи установочника npm(node package manager) командой *npm install create-react-app my-project.*



Также необходимо установить при помощи менеджера *npm* библиотеку *esri-loader* командой *npm install esri-loader.*



С*оздадим* модуль MapView.js в котором будет происходить основная логика отображения графики и карт в данном приложении.

Импортируем модули React, loadModules из библиотек установленных библиотек и написанные нами функции из модулей. Добавляем стили *map.css* для настройки отображения в браузере.

import React from 'react';

import { loadModules } from 'esri-loader';

import calculateEllipse from './calculateEllipse';

import './map.css';

import OilsAlias from './OilsAlias';

Опишем класс *WebMapView* который наследует свойства класса react.

Передадим конструктору свойства props и так же передадим их супер конструктору который является родителем данного свойства. Так же создадим ссылку использую специальную конструкцию в реакте называемые хуки.

export default class WebMapView extends React.Component {

  constructor(props) {

    super(props);

    this.mapRef = React.createRef();

  }

Укажем параметры по умолчанию для отображения карты.

static defaultProps={

    X:30.20416,

    Y:59.93237,

    zoom:10,

  };

Используем хук состояния ComponentDidMount для загрузки карты. Данный хук используется для выполнения каких-либо действий при монтировании компонента.

Используем ленивую загрузку loadModules для подгружения необходимых инструментов

из библиотеки Arcgis for API. Передадим в данный модуль параметры через запятую.

Конструктор *esri/Map* для создания карты, esri/views/MapView для рендера данной карты и настройку параметров, *'esri/Graphic'* для рисования графики на карте, *esri/geometry/geometryEngine* для проведения расчетов. *loadModules* использует асинхронную загрузку поэтому для получения результатов данных из нее используем метод then(), который является аналог конструкции async/await. В метод then передадим стрелочную функцию ()=>{}. Особенность данной функции что у нее отсутствует собственный this на функцию. Передадим в нее подгруженные конструкторы и в теле функции будем использовать их.

componentDidMount() {

    // lazy load the required ArcGIS API for JavaScript modules and CSS

    loadModules(['esri/Map','esri/views/MapView','esri/Graphic',"esri/geometry/geometryEngine"], { css: true })

    .then(([Map, MapView,Graphic,geometryEngine]) => {

При помощи конструктора *Map* создадим объект карты c параметром карты topo-vector и сохраним результат в переменную map.

 const map = new Map({

        basemap: 'topo-vector'

 });

Используем конструктор MapView для настройки параметров карты. Контейнер в который будет рендерится данная карта укажем ссылку на данный элемент. В качестве параметра карты, которая будет отображена укажем переменную map в которой хранится объект построенный на предыдущем шаге. Начальные точки координаты передадим как координаты, указанные по умолчанию в пропсах X и Y.

 this.view = new MapView({

        container: this.mapRef.current,

        map: map,

        center: [this.props.X, this.props.Y],

        zoom: this.props.zoom

      });

На лицевой панели в модуле App.js мы указываем входные параметры которые в дальнейшем будут переданы в данный модуль. Воспользуемся поиском по селектору чтобы получить значения указанные пользователем и записать их в переменные.

    const clicka = document.querySelector('#clicker');

Добавим к данному элементу событие click исполняющую в функицю по добавлению графики.

 clicka.addEventListener('click',()=>{

Произведем поиск по селектору входных параметров.

  let Xcoord = document.querySelector('#X').value;

        let Ycoord = document.querySelector('#Y').value;

        let Weight = document.querySelector('#weight').value;

        let startTime = document.querySelector('#startTime').value;

После вызовем написанную нами функцию OilsAlias и передадим в нее данные заданые пользователем на лицевой панели. Так как данная функция является асинхронной и для ее выполнения требуется неизвестное количество времени используем конструкцию .then() . В результате вызова данная функция возвращает массив точек, которые хранятся в переменной data в дальнейшем будут отрисованы. Для прохождения внутри данного массива воспользуемся циклом for который пройдется по количеству элементов внутри.

        OilsAlias(Xcoord,Ycoord,startTime,Weight).then((data)=>{

          for(let i=0;i<data.length;i++){

В переменной data хранится массив с 10 массивами на основе которых будет построены точки. Выбираем перевый массив и передаем значения внутри данного массива в функцию calculateEllipse и записываем результат в переменную ellipse/

 let ellipse\_rings = calculateEllipse(inputValue[1],inputValue[2],inputValue[3],inputValue[4],inputValue[5]);

Создадим полигон на основе полученных точек.

let OilPolygon = {

              type: "polygon",

              rings: ellipse\_rings

            };

Создадим объект, описывающий графику для данного полигона. Укажем прозрачность данного полигона в зависимости от итерации. Таким образом чем больше будет проходить времени, тем более прозрачный объект будет.

 const FillSymbolCss = {

              type: "simple-fill",

              color: [64, 38, 8, (1-i\*0.1)],

                // orange, opacity 80%

              outline: {

                color: 'white',

                width: 1

              }

            };

Используем конструктор Graphic для создания объекта графики и добавим этот объект на нашу карту.

      let OilsGraphic = new Graphic({

              geometry: OilPolygon,

              symbol: FillSymbolCss

            });

            this.view.graphics.add(OilsGraphic);

В результате действия данного модуля отрендерится карта и при запросе на копке отобразить с входными параметрами получим эллипс нефтяного пятна в результате разлива.

