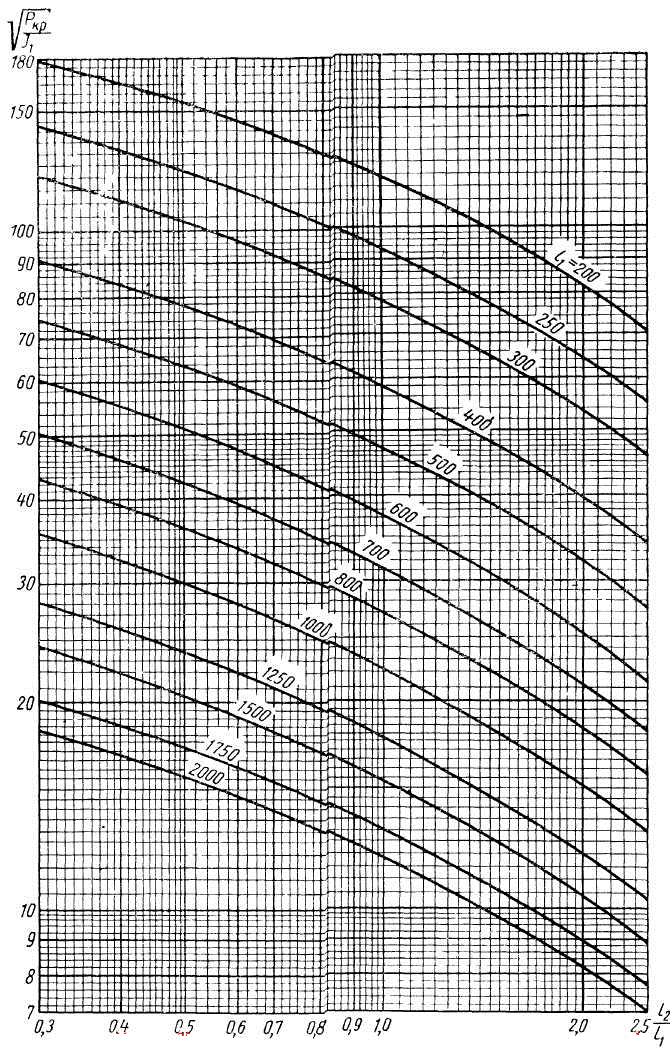
**Задание:**

Создать решение для аппроксимации данных с графиков.

Для образца дана зависимость от для напряжений штырей гидроцилиндра в максимально вытянутой позиции.



**Анализ:**

Воспользуемся web платформой. Будем использовать JavaScript.

В выборе языка исполнения мы руководствуемся следующими принципами:

- JavaScript исполняется в среде браузера, что значит, что данное программное обеспечение будет без проблем запущено на любой платформе(Windows, Linux, BSD)

- Графические средства браузеров не сравнимы по затратам с другими платформами

**Реализация:**

Для реализации воспользуемся библиотекой D3.js Марка Блокстока (http://d3js.org/). Данная библиотека основана на технологии SVG (Scalable Vector Graphics) – масштабируемой векторной графики.

Отрисуем систему координат. Возьмем половину ширины экрана и половину высоты окна браузера для координат центра. Очень важно учитывать что начало координат окна браузера берет начало с левого верхнего угла и инвентировано по оси y.

var widthSVGdescart = document.body.clientWidth;

var heightSVGdescart = screen.height \* 0.75;

var centrHSVGdescart = screen.height \* 0.75 / 2;

var centrWSVGdescart = document.body.clientWidth / 2;

Теперь можно задать поля для рисования

var svg = d3.select("body")

.append("svg")

.attr("width", widthSVGdescart)

.attr("height", heightSVGdescart);

Теперь можно отрисовать оси

var descart = svg.append("g")

.attr("id", "descart")

.classed("descart", true);

descart.append("line")

.attr("id", "axisY")

.attr("x1", widthSVGdescart / 2)

.attr("y1", heightSVGdescart \* -10)

.attr("x2", widthSVGdescart / 2)

.attr("y2", heightSVGdescart \* 10)

.style({

"stroke": "red",

"stroke-width": 1

});

descart.append("line")

.attr("x1", widthSVGdescart / 2)

.attr("y1", heightSVGdescart \* -10)

.attr("x2", widthSVGdescart / 2)

.attr("y2", heightSVGdescart \* 10)

.style({

"stroke": "red",

"opacity": 0,

"stroke-width": 30

});

descart.append("line")

.attr("id", "axisX")

.attr("x1", widthSVGdescart \* -10)

.attr("y1", heightSVGdescart / 2)

.attr("x2", widthSVGdescart \* 10)

.attr("y2", heightSVGdescart / 2)

.style({

"stroke": "red",

"stroke-width": 1

});

descart.append("line")

.attr("x1", widthSVGdescart \* -10)

.attr("y1", heightSVGdescart / 2)

.attr("x2", widthSVGdescart \* 10)

.attr("y2", heightSVGdescart / 2)

.style({

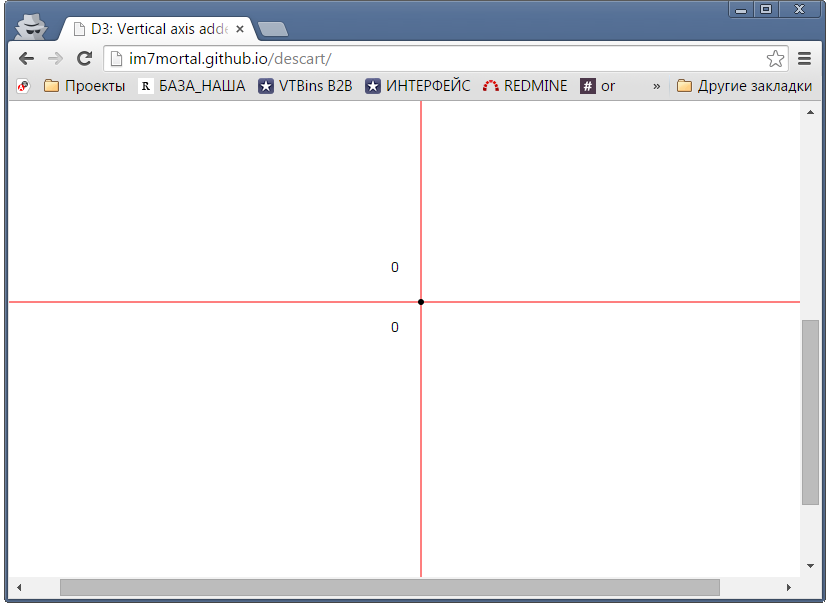
"stroke": "red",

"opacity": 0,

"stroke-width": 30

});

Получаем



Теперь нужно добавить возможность добавлять метки на оси с заданными значениями.

Так как графики сильно искажены нужно иметь возможность двигать отметки и настраивать их положение.

Для интерактивности воспользуемся библиотекой interact.js

Для начала настроим интерактивность для системы координат,

чтобы была возможность передвигать систему координат вдоль графика для более точной настройки.

interact('.descart')

.draggable({

inertia: true,

restrict: {

elementRect: { top: 0, left: 0, bottom: 1, right: 1 }

},

onmove: function (event) {

var target = event.target,

x = (parseFloat(target.getAttribute('data-x')) || 0) + event.dx,

y = (parseFloat(target.getAttribute('data-y')) || 0) + event.dy;

// translate the element

target.style.webkitTransform =

target.style.transform =

'translate(' + x + 'px, ' + y + 'px)';

origin.x.x += event.dx;

origin.y.y -= event.dy;

// update the posiion attributes

target.setAttribute('data-x', x);

target.setAttribute('data-y', y);

},

// call this function on every dragend event

onend: function (event) {

var textEl = event.target.querySelector('p');

textEl && (textEl.textContent =

'moved a distance of '

+ (Math.sqrt(event.dx \* event.dx +

event.dy \* event.dy)|0) + 'px');

}

});

Теперь создадим класс для отрисовки отметок.

function Mark() {

}

Mark.prototype.handler = function () {

var number = parseFloat(this.input.value);

var flag;

this.array.some(function (object) {

if (object.value === number) {

flag = true;

return true

}

});

if (this.g) {

this.value = null;

this.quantity = null;

this.g.remove();

removeMark(this.name, this.array);

}

if (isFinite(number) && this.origin && !flag) {

this.value = number;

this.render();

}

this.calc();

};

Mark.prototype.init = function () {

this.name = 'mark' + this.axes + countMark++;

this.inputGroup = d3.select('#input' + this.axes)

.append('div')

.classed('input-group', true);

this.inputGroup

.append('span')

.classed('input-group-btn', true)

.append('button')

.on('click', this.remove.bind(this))

.attr('type', 'button')

.classed('btn btn-default', true)

.append('span')

.attr('aria-hidden', 'true')

.classed('glyphicon glyphicon-remove', true);

this.input = this.inputGroup

.append('input')

.classed('form-control', true)

.attr('type', 'text')

.attr('name', this.name)

.attr('placeholder', 'Search for...')[0][0];

this.input.onchange = this.handler.bind(this);

if(this.value && this.quantity) {

this.input.value = this.value;

this.render();

}

};

Mark.prototype.remove = function () {

if (this.g) this.g.remove();

if (this.inputGroup) this.inputGroup.remove();

removeMark(this.name, this.array)

};

Mark.prototype.prepareArray = function () {

this.array.sort(function (object1, object2) {

if (object1.value > object2.value) {

return 1;

} else {

return -1;

}

})

};

Mark.prototype.accelerator = function (quantity) {

this.array.forEach(function (object, i, array) {

if (object.name === this.name) {

if (quantity < 0) {

if (i !== array.length - 1) {

if (array[i + 1].dQuantity + quantity <= 5) {

quantity = 0;

}

}

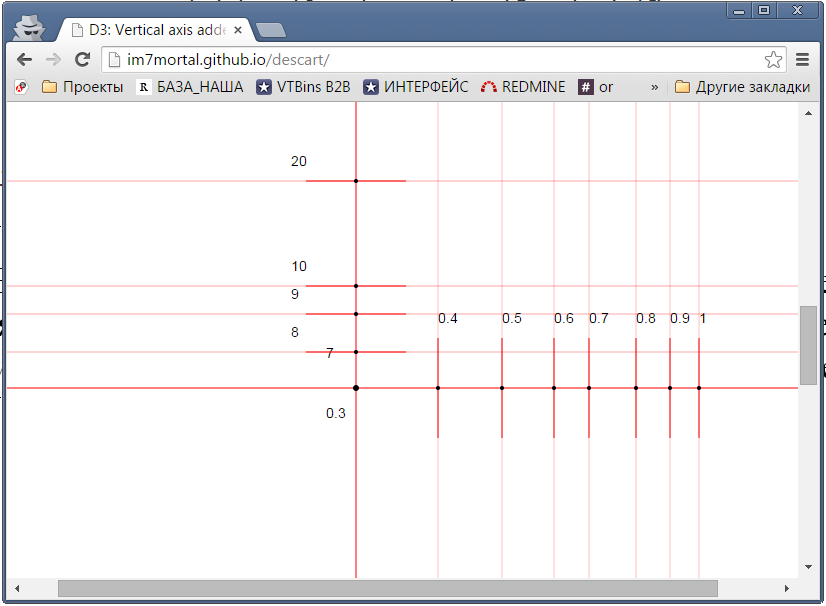
}

}, this);

return quantity;

};

Зададим данные из кода для теста. Получаем следующий результат



Нужно создать удобный интерфейс для задания данных.

Сначала создадим поля ввода для задания начала координат. Так как существует возможность что начало координат может начинаться не с нуля.

Формы задаются в html

<div id="tipX">Задайте начало отсчета по X</div>

<div class="input-group">

<span class="input-group-addon">Начало отсчета по оси X</span>

<input type="text" class="form-control" id="originX" placeholder="Username"

aria-describedby="basic-addon1">

</div>

<div id="tipY">Задайте начало отсчета по Y</div>

<div class="input-group">

<span class="input-group-addon">Начало отсчета по оси Y</span>

<input type="text" class="form-control" id="originY" placeholder="Username"

aria-describedby="basic-addon1">

</div>

Формы для полей отметок отрисовываются динамически из js скриптов

Добавляем к классу отметок новый метод

Mark.prototype.init = function () {

this.name = 'mark' + this.axes + countMark++;

this.inputGroup = d3.select('#input' + this.axes)

.append('div')

.classed('input-group', true);

this.inputGroup

.append('span')

.classed('input-group-btn', true)

.append('button')

.on('click', this.remove.bind(this))

.attr('type', 'button')

.classed('btn btn-default', true)

.append('span')

.attr('aria-hidden', 'true')

.classed('glyphicon glyphicon-remove', true);

this.input = this.inputGroup

.append('input')

.classed('form-control', true)

.attr('type', 'text')

.attr('name', this.name)

.attr('placeholder', 'Search for...')[0][0];

this.input.onchange = this.handler.bind(this);

if(this.value && this.quantity) {

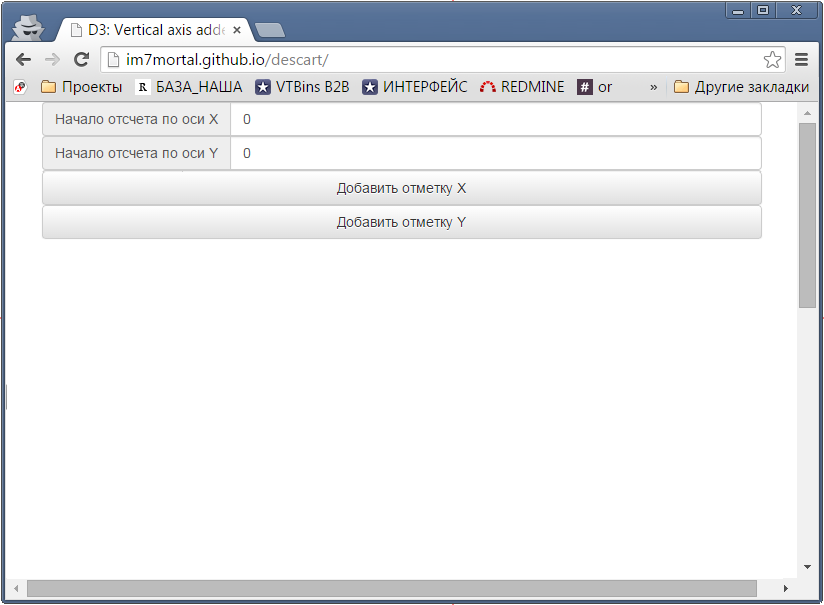
this.input.value = this.value;

this.render();

}

};

Получаем следующий интерфейс



Следующим шагом добавим возможность загружать выбранный график в область построения графика

Добавим интерфейс

<div class="input-group">

<span class="input-group-addon">Загрузите график</span>

<input name="files" type="file" id="fileInput" class="form-control" placeholder="Username"

aria-describedby="basic-addon1">

</div>

Добавим обработчик

var fileInput = document.getElementById("fileInput");

fileInput.addEventListener("change", function () {

if (this.files.length) {

graph = svg.insert("image", 'g#descart')

.classed("gap", true)

.attr("xlink:href", this.files[0].name)

.attr("x", 0)

.attr("y", 0);

if (FileReader) {

var reader = new FileReader();

reader.onloadend = function () {

var img = document.createElement('img');

img.onload = function () {

graph.attr("width", this.width)

.attr("height", this.height);

};

img.src = reader.result;

};

reader.readAsDataURL(this.files[0]);

} else {

graph.attr("height", 956)

.attr("width", 606);

}

}

});

Добавим интерактивность

interact('.gap')

.draggable({

inertia: true,

restrict: {

elementRect: { top: 0, left: 0, bottom: 1, right: 1 }

},

onmove: function (event) {

var target = event.target,

x = (parseFloat(target.getAttribute('data-x')) || 0) + event.dx,

y = (parseFloat(target.getAttribute('data-y')) || 0) + event.dy;

// translate the element

target.style.webkitTransform =

target.style.transform =

'translate(' + x + 'px, ' + y + 'px)';

// update the posiion attributes

target.setAttribute('data-x', x);

target.setAttribute('data-y', y);

},

// call this function on every dragend event

onend: function (event) {

var textEl = event.target.querySelector('p');

textEl && (textEl.textContent =

'moved a distance of '

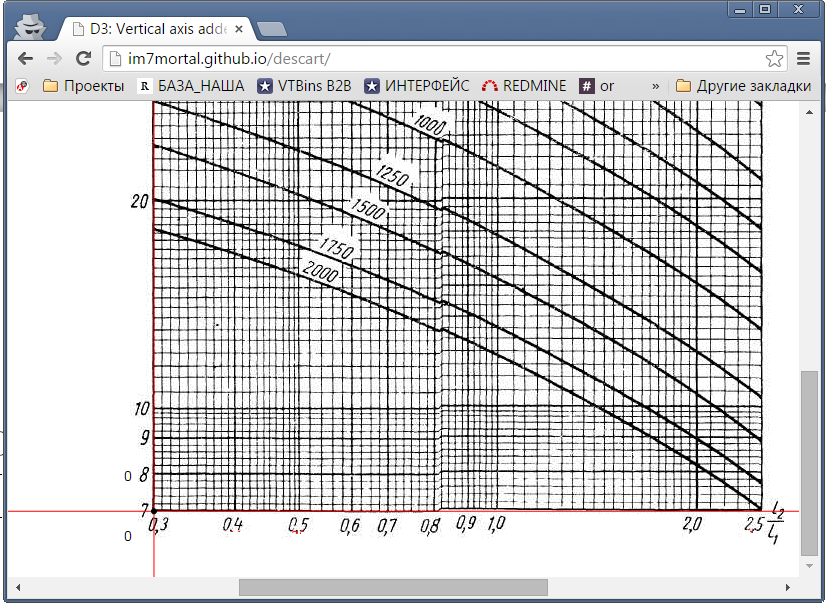
+ (Math.sqrt(event.dx \* event.dx +

event.dy \* event.dy)|0) + 'px');

}

})

Оценим полученный результат

полученный

Добавим возможность добавлять точки на график

function Point (x , y) {

this.name = "point" + countPoints++;

points[this.name] = this;

var self = this;

this.x = x || 20;

this.y = y || 20;

this.g = descart.append("g")

.attr("id", this.name)

.classed("point", true);

var cx = widthSVGdescart / 2 + this.x,

cy = heightSVGdescart / 2 - this.y;

this.point = this.g.append("circle")

.attr("id", "zero")

.attr("cx", cx)

.attr("cy", cy)

.attr("r", 1.5)

.style({

"stroke": "red"

});

this.g.append("circle")

.attr("id", "zero")

.attr("cx", cx)

.attr("cy", cy)

.attr("r", 3)

.style({

"stroke": "red",

"opacity": 0

});

}

Point.prototype.eng = function (event) {

this.x += event.dx;

this.y -= event.dy;

this.rationer()

};

Point.prototype.rationer = function () {

var X = 0;

var quantityX = 0;

var arrx = range.x.filter(function (o,i,a) {

if(!i) return true;

if(this.x > o.originalQuantity) {

return true;

} else if (this.x > a[i - 1].originalQuantity) {

return true;

}

return false;

}, this);

arrx.forEach(function (object, i, array) {

var offset = object.offset?object.offset:0;

if(i === (array.length - 1)) {

X += (this.x - quantityX) \* object.ratio + offset;

} else {

quantityX += object.quantity;

X += object.quantity \* object.ratio + offset;

}

}, this);

Сразу же снимаем координаты точек и выводим их в интерфейс

}

var Y = 0;

var quantityY = 0;

var arry = range.y.filter(function (o,i,a) {

if(!i) return true;

if(this.y > o.originalQuantity) {

return true;

} else if (this.y > a[i - 1].originalQuantity) {

return true;

}

return false;

}, this);

console.log(arry);

arry.forEach(function (object, i, array) {

var offset = object.offset?object.offset:0;

if(i === (array.length - 1)) {

Y += (this.y - quantityY) \* object.ratio + offset;

} else {

quantityY += object.quantity;

Y += object.quantity \* object.ratio + offset;

}

}, this);

this.valueX = X;

this.valueY = Y;

console.log({

"x": X,

"y": Y

});

return {

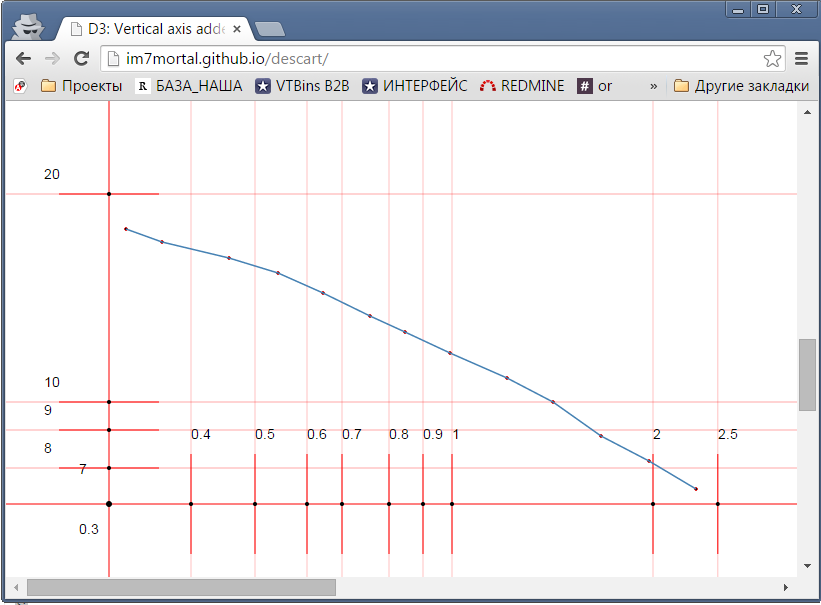
"x": X,

"y": Y

}

};

Посмотрим на полученный результат



Оценим полученные данные.

**Вывод:**

В результате проведенной работы, я снял данные с графика . зависимости от для напряжений штырей гидроцилиндра в максимально вытянутой позиции пригодные для дальнейших изысканий.