getComplex и подобные перенести в новый классс universal calculator

В методе getValue соответствующие методы вызывать через ключевое слово this.

Calculator.js

getVector(sh){

if (str intstanceof Array) return new Vector(str)

if (str && typeof str === ‘string’){

const arr=str.replace(‘(‘,’’).replace(‘),’’).split(‘ ’).map(el => this.getValue(el));

return new Vector(arr);

}

return null;

}

Для метода getMatrix сделать то же самое

complex(re,im){

return new Complex(re,im);

}

vector(values){return new Vector(values);}

matrix(values){return new Matrix(values);}

Поскольку теперь у сущности разрешена любая степень вложенности, мы не можем воспользоваться только одним калькулятором и нам необходимо написать метод общего калькулятора, который может работать со всем.

Метод get по элементу определяет какой калькулятор должен быть задействован.

get(elem){

if (elem instanseof Matrix){

return new MatrixCalculator(this.get(elem.values[0][0]));

}

if (elem instanseof Vector){

return new VectorCalculator(this.get(elem.values[0]));}

if (elem instanseof Complex){

return new ComplexCalculator;

}

return new RealCalculator;

}

add(a,b){

return this.get(a).add(a,b);

}

Также с другими методами

zero(type, elem){

type = type ? type: elem ? elem.constructor.name : null;

switch(type){

case ‘Complex’ : return this.get(this.complex()).zero();

case ‘Vector’ : return this.get(this.vector()).zero(elem.values.length);

case ‘Matrix’ : return this.get(this.matrix()).zero(elem.values.length);

default : return this.get().zero();

}

}

Единица пишется аналогичным образом

class MatrixCalculator{

constructor(calc = new RealCalculator){

this.calc = calc;

}

add(a,b){

return new Matrix(a.values.map((arr,i) =>

arr.map((elem,j)=>this.calc.add(elem,b.values[i][j])))

);}

}

Полином

class Member{

constructor(value=0, power=0){

this.value = value;

this.power = power;

}

toString(){…}

//нужно самим написать

class Polynomial{

constructor(poly = []){

this.poly = poly;

this.poly.sort((a,b)=>b.pow-a.pow);d

}

getValue(x){

const calc = new Calculator;

return this.poly.reduce((S,elem) =>

calc.add(S,calc.mult(elem.value, calc.pow(x, elem.power)),

calc.zero(null,x)

);

}

}

class PolynomialCalculator{

polynomial(members){

return new Polynomial(members)

}

add(a,b){

const calc = new Calculator;

const members = [ ];

a.poly.forEach(elemA => {

const member = b.poly.find(elemB => elemB.power === elemA.power);

if (member){  
 members.push(new Member(calc.add(elemA.value, member.value), elemA.power));

}else{

members.push(new Member(elemA.value, elemA.power));

}

});

b.poly.forEach(elemB => {

if(!members.find(elem=>

elem.power === elemB.power)){

members.push(new Member(elemB.value, elemB.power));

}

});

return new Polynomial(members);

//вычитание также только elemB.value надо умножить на минус 1

mult(a,b){

const calc = new Calculator;

let polynomial = new Polynominal;

a.poly.forEach(elemA => {

const members = [ ];

b.poly.forEach(elemB = >{

members.push(new Member(calc.mult(elemA.value, elemB.value),elemA.power+elemB.power));

});

polynomial = this.add(polynomial, new Polynomial(members));

});

//деление не надо

//prod, zero

}