第二章作业

201917090008 邹金廷

OS: Arch Linux x86_64
Kernel: 5.16.14-arch1-1
Node version: v17.7.1

运行：使用本地Node.js，或者在线运行：https://c.runoob.com/compile/22/

二分法

代码：

```
class bisection{
    constructor(fun1,left_val,right_val,N,bais1){
        this.fun=fun1;
        this.a=left_val
        this.b=right_val
        this.max_step=N
        this.bais=bais1
    }
    check(){
        var check_val;
        if(this.fun(this.a)*this.fun(this.b)<0){
            check_val=true
        }
        else{check_val=false}
        return check_val
    }

    show(){
        var temp_step=0,
            a_temp=this.a,
            b_temp=this.b,
            output_p=0,
            output_bais=[],
            output_p_temp=[];

        while(temp_step<=this.max_step){
            var p=a_temp+(b_temp-a_temp)/2
            var temp_val=this.fun(p)
            output_bais.push(p)
            output_p_temp.push(temp_val)
            if(temp_val==0||(b_temp-a_temp)<this.bais){
                output_p=p
                break
            }
            else{
                temp_step++
                if(temp_val*this.fun(a_temp)>0){a_temp=p}
                else{b_temp=p}
            }
            if(temp_step==this.max_step){output_p=p}
        }
```

```
        this.temp_step=output_p_temp
        this.bais_step=output_bais
        return output_p
    }
}

function the_fun1(x){
    return x*x-10*x+3
}

var b1=new bisection(the_fun1,1,15,20,0.0001)
var p1=b1.show()
console.log(p1)
console.log(b1.temp_step)
console.log(b1.bais_step)
```

实验函数：

$$x^2 - 10x + 3$$

初始值：1，15，最大步数：20，TOL：0.001

结果：分别最终结果，eps，每次迭代结果。

```
homework : zsh — Konsole
~/Documents/homework
node 1.js
9.690425872802734
[
                   -13,                  20.25,
                0.5625,              -6.984375,
           -3.40234375,          -1.4677734375,
       -0.464599609375,       0.04595947265625,
   -0.2100677490234375,   -0.08224105834960938,
 -0.018187522888183594,   0.013874292373657227,
-0.0021595358848571777,   0.005856648087501526,
 0.0018483735620975494, -0.00015562679618597 03,
 0.0008463619742542505, 0.00034536473685875535,
0.00009486825729254633
]
[
                 8,              11.5,
              9.75,             8.875,
            9.3125,           9.53125,
          9.640625,         9.6953125,
        9.66796875,       9.681640625,
      9.6884765625,     9.69189453125,
    9.690185546875,   9.6910400390625,
  9.69061279296875, 9.690399169921875,
9.690505981445312, 9.690452575683594,
9.690425872802734
]
~/Documents/homework
```

不动点

代码：

```
class fixed_point{
    constructor(the_fun,N,bias1){
        this.fun=the_fun
        this.max_step=N
        this.bias=bias1
    }
    show(p){
        var output_p=0,
```

```
            temp_p=p,
            temp_step=0,
            temp_p_list=[];
        while(temp_step<=this.max_step){
            var temp_p_step=this.fun(temp_p)
            temp_p_list.push(temp_p_step)
            if(Math.abs(temp_p_step-temp_p)<this.bias){
                output_p=temp_p_step
                break
            }
            else{
                temp_step++
                temp_p=temp_p_step
            }
            if(temp_step==this.max_step){output_p=temp_p_step}
        }
        this.p_list=temp_p_list
        return output_p
    }
}

function the_fun2(x){
    return Math.sqrt(10-Math.pow(x,3))/2
}

var f1=new fixed_point(the_fun2,30,0.001)
console.log(f1.show(1.5))
console.log(f1.p_list)
```

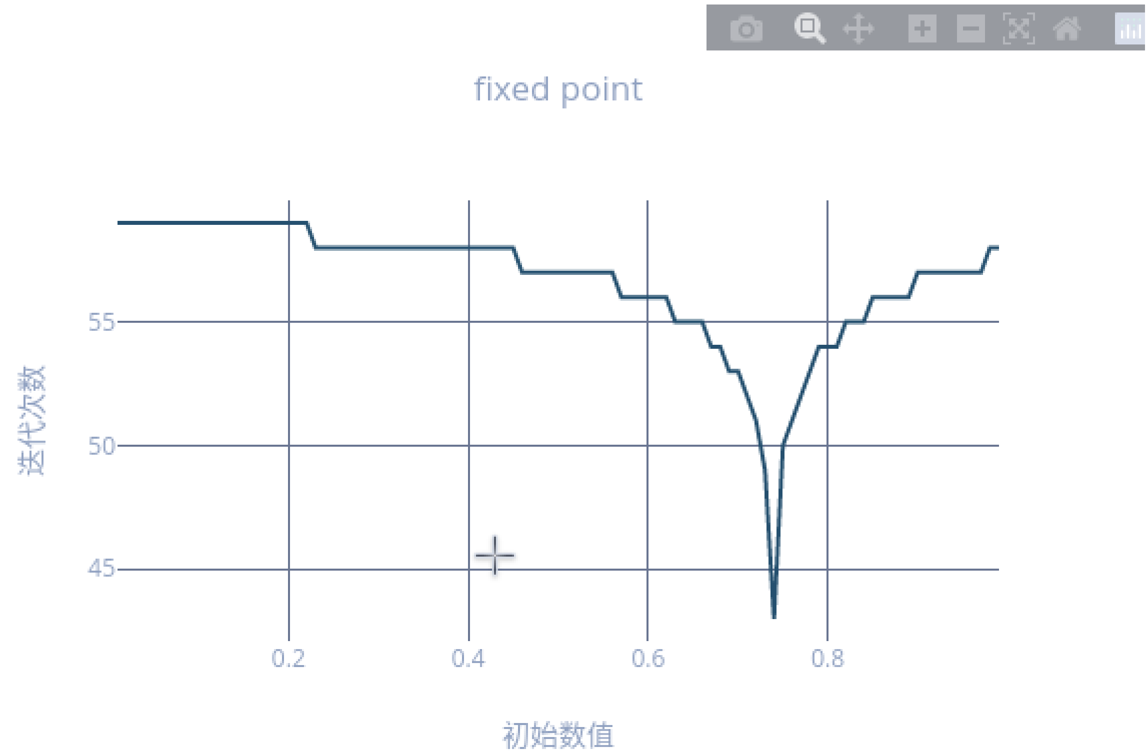
实验函数：

$$\frac{\sqrt{10-x^3}}{2}$$

初始数值：1.5，最大迭代次数：30，TOL：0.001

结果：由于精确度不同，和书上结果有一定差距

对于不同初始数值迭代次数图像：



牛顿法

代码：

```
class Newton_mtd{
    constructor(the_fun,thefun_grad,N,bias1){
        this.fun=the_fun
        this.max_step=N
        this.bias=bias1
        this.fun_grad=thefun_grad
    }

    show(p){
        var temp_step=0,
            temp_p=p,
            temp_p_list=[],
            output_p=0;

        while(temp_step<this.max_step){
            var temp_p_step=temp_p-this.fun(temp_p)/this.fun_grad(temp_p)
            if(Math.abs(temp_p-temp_p_step)<this.bias){
                output_p=temp_p_step
                break
            }
            temp_step++
            temp_p=temp_p_step
            temp_p_list.push(temp_p)

            if(temp_step==this.max_step){output_p=temp_p}
        }
        this.p_list=temp_p_list
        return output_p
    }
}
```

```
function the_fun3(x){
     return Math.cos(x)-x
}

function the_fun3_grad(x){
     return Math.sin(x)*(-1)-1
}

var ne1=new Newton_mtd(the_fun3,the_fun3_grad,20,Math.pow(10,-10))
console.log(ne1.show(Math.PI/4))
console.log(ne1.p_list)
```

实验函数：

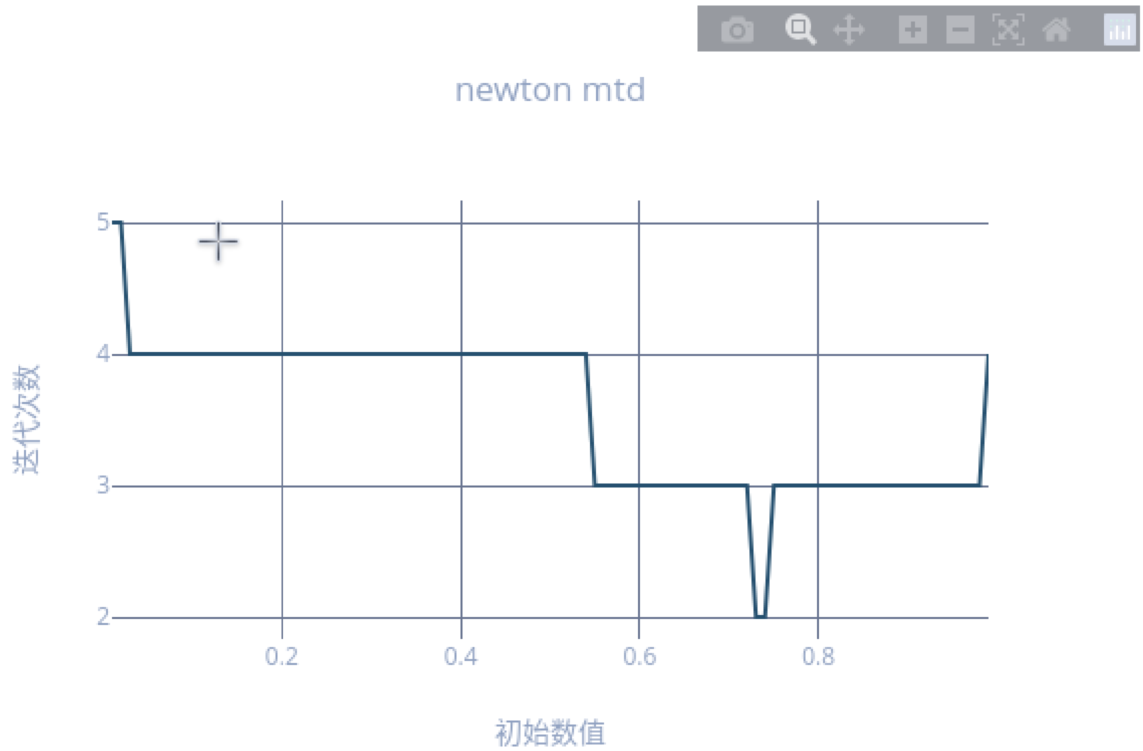
$$\cos(x) - x$$

初始数值：$\pi/4$，最大迭代次数：20，TOL：10^{-10}

下面割线法和false postion使用参数相同

结果：

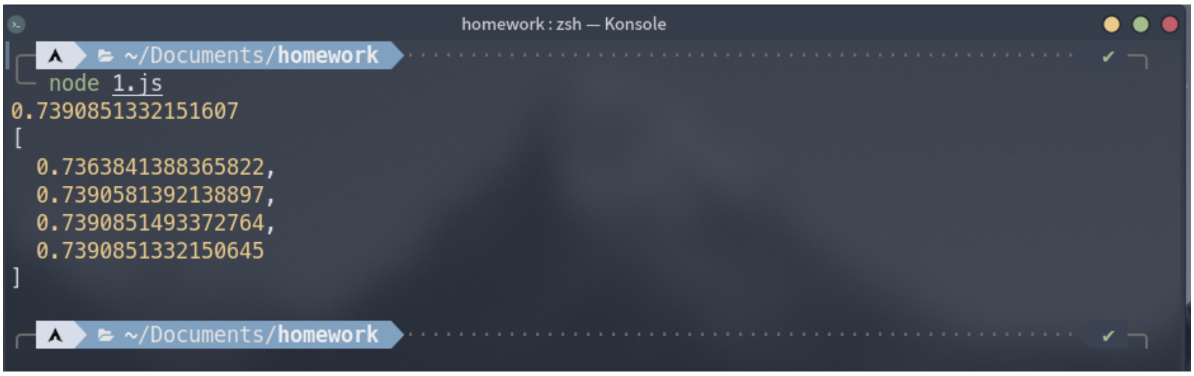
对于不同初始数值的迭代次数图像：



割线法

```
class secant_mtd extends Newton_mtd{
    show(p0,p1){
        var temp_step=0,
            temp_p0=p0,
            temp_p1=p1,
            temp_q0=this.fun(temp_p0),
            temp_q1=this.fun(temp_p1),
            temp_p_list=[],
            output_p=0;

        while(temp_step<=this.max_step){
            var temp_p_step=temp_p1-temp_q1*(temp_p1-temp_p0)/(temp_q1-temp_q0)
            if(Math.abs(temp_p_step-temp_p1)<this.bias){
                output_p=temp_p_step
                break
            }
            temp_step++

            temp_p0=temp_p1
            temp_q0=temp_q1
            temp_p1=temp_p_step
            temp_q1=this.fun(temp_p_step)
            temp_p_list.push(temp_p_step)
            if(temp_step==this.max_step){output_p=temp_p_step}
        }
        this.p_list=temp_p_list
        return output_p
    }
}

var sec1=new secant_mtd(the_fun3,the_fun3_grad,20,Math.pow(10,-10))
console.log(sec1.show(0.5,Math.PI/4))
console.log(sec1.p_list)
```

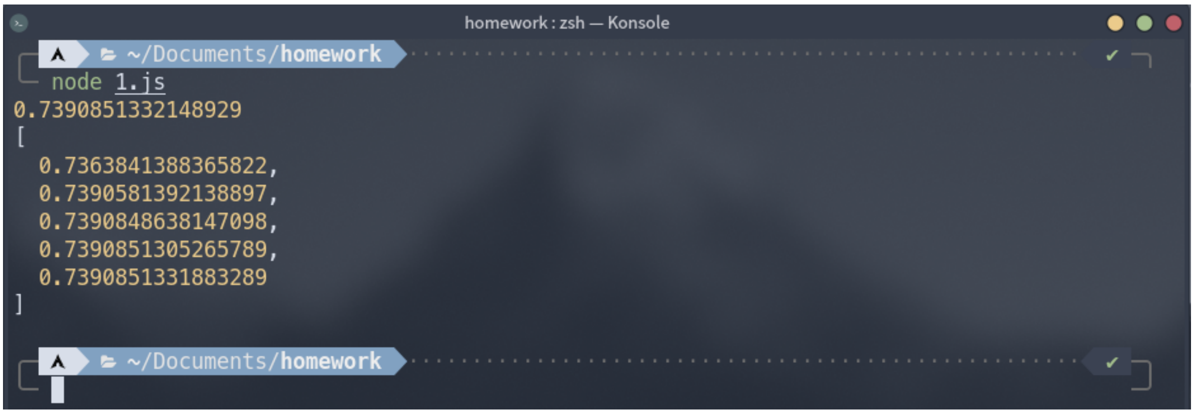
结果：



False Position

```
class false_posti extends Newton_mtd{
    show(p0,p1){
        var temp_step=0,
            temp_p0=p0,
            temp_p1=p1,
            temp_q0=this.fun(temp_p0),
```

```
            temp_q1=this.fun(temp_p1),
            temp_p_list=[],
            output_p=0;

        while(temp_step<=this.max_step){
            var temp_p_step=temp_p1-temp_q1*(temp_p1-temp_p0)/(temp_q1-temp_q0)
            if(Math.abs(temp_p_step-temp_p1)<this.bias){
                output_p=temp_p_step
                break
            }
            temp_step++
            var tem_q_step=this.fun(temp_p_step)
            if(tem_q_step*temp_q1<0){
                temp_p0=temp_p1
                temp_q0=temp_q1
            }
            temp_p1=temp_p_step
            temp_q1=tem_q_step
            temp_p_list.push(temp_p_step)
            if(temp_step==this.max_step){output_p=temp_p_step}
        }
        this.p_list=temp_p_list
        return output_p
    }
}

var fal1=new false_posti(the_fun3,the_fun3_grad,20,Math.pow(10,-10))
console.log(fal1.show(0.5,Math.PI/4))
console.log(fal1.p_list)
```



Steffensen方法

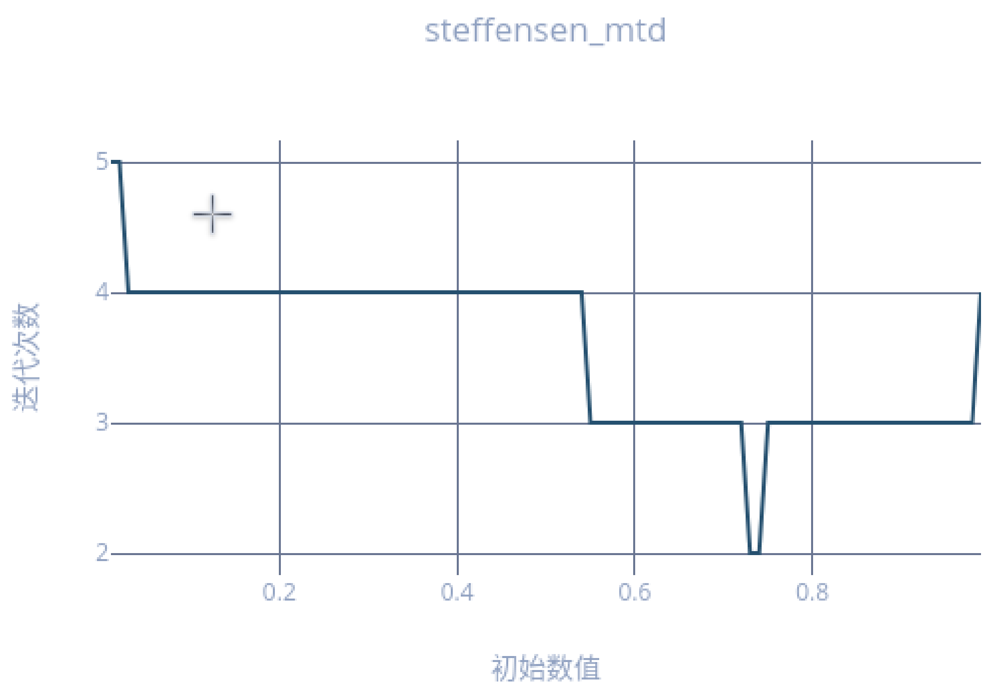
```
class steffensen_mtd{
    constructor(the_fun,N,eps){
        this.fun=the_fun
        this.max_step=N
        this.bias=eps
    }

    show(p){
        var p_temp=p,
            temp_step=0,
            output_p=0,
            output_p_list=[];
```

```
        while(temp_step<=this.max_step){
            var temp_p1=this.fun(p_temp),
                temp_p2=this.fun(temp_p1),
                temp_p_step=p_temp-Math.pow(temp_p1-p_temp,2)/(temp_p2-2*temp_p1+p_temp);
            if(Math.abs(p_temp-temp_p_step)<this.bias){
                output_p=temp_p_step
                break
            }
            temp_step++
            p_temp=temp_p_step
            output_p_list.push(p_temp)
            if(temp_step==this.max_step){
                output_p=p_temp
                break
            }
        }
        this.p_list=output_p_list
        return output_p
    }
}

function the_fun3_fix(x){
    return Math.cos(x)
}

var st1=new steffensen_mtd(the_fun3_fix,100,Math.pow(10,-10))
console.log(st1.show(0.5))
console.log(st1.p_list)
```

实验函数：$\cos(x)$，最大迭代次数：100，TOL：10^{-10}

```
 ~/Documents/homework
node 1.js
0.7390851332151607
[ 0.7313851863825818, 0.7390763403695223, 0.7390851332036612 ]
```

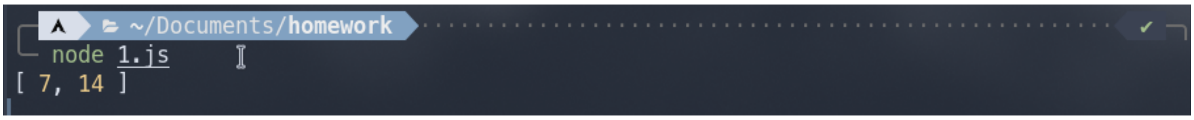
对于不同初始数值的迭代次数图像



Horner 方法

```
class horner_mtd{
    constructor(pln1=[]){   //2x^2+3x+4:[2,3,4]
        this.pln=pln1
        this.len=pln1.length
        // this.x_0=x
    }

    show(x){
        var x_0=x
        var y_output=this.pln[0]
        var z_output=this.pln[0]

        for(var i=1;i<this.len-1;i++){
            y_output=x_0*y_output+this.pln[i]
            z_output=x_0*z_output+y_output
        }

        y_output=x_0*y_output+this.pln[this.len-1]

        var output=[]
        output.push(y_output)
        output.push(z_output)
        return output
    }

}

var ho1=new horner_mtd([3,2,1,1])
console.log(ho1.show(1))
```

实验函数：$3x^3+2x^2+x+1$，数据：1



Muller方法

```
const { sqrt, abs, pow, add, multiply, subtract, divide, arg, sign, clone,
complex, subset} = require("mathjs");

class muller_mtd{
    constructor(fun1,N,t1){
        this.fun=fun1
        this.max_step=N
        this.TOL=t1
    }

    show(p0=complex(0,0),p1=complex(0,0),p2=complex(0,0)){
        var p_0=p0,
            p_1=p1,
            p_2=p2,
            h_1=subtract(p1,p0),
            h_2=subtract(p2,p1),
            es1=(this.fun(p1)-this.fun(p0))/h_1,
            es2=(this.fun(p2)-this.fun(p1))/h_2,
            d=divide(subtract(es2,es1),add(h_2,h_1)),
            p_output=complex(0,0),
            i=3;

        while(i<=this.max_step){
            var b=add(es2,multiply(h_2,d))
            var
D=sqrt(subtract(multiply(b,b),multiply(multiply(4,this.fun(p2)),d)))
            var E

            if(abs(subtract(b,D))<abs(add(b,D))){
                E=add(b,D)
            }
            else{
                E=subtract(b,D)
            }

            var h=divide(multiply(-2,this.fun(p_2)),E)
            var p=add(p_2,h)

            if(abs(h)<this.TOL){
                p_output=p
                break
            }

            p_0=clone(p_1)
            p_1=clone(p_2)
            p_2=clone(p)
            h_1=subtract(p_1,p_0)
            h_2=subtract(p_2,p_1)
            es1=divide( subtract(this.fun(p_1),this.fun(p_0)) ,h_1)
            es2=divide( subtract(this.fun(p_2),this.fun(p_1)) ,h_2)
            d=divide(subtract(es2,es1),add(h_2,h_1))
```

```
                i++

                if(i==this.max_step){
                    p_output=clone(p)
                }
                console.log(p)
            }
            return p_output
        }
}

function fun_1(x){
    return add(add(subtract(pow(x,4),multiply(3,pow(x,3))),pow(x,2)),add(x,1))
}
var m1=new muller_mtd(fun_1,100,pow(10,-10))
console.log(m1.show(complex(0.5,0),complex(-0.5,0),complex(0,0)))
```

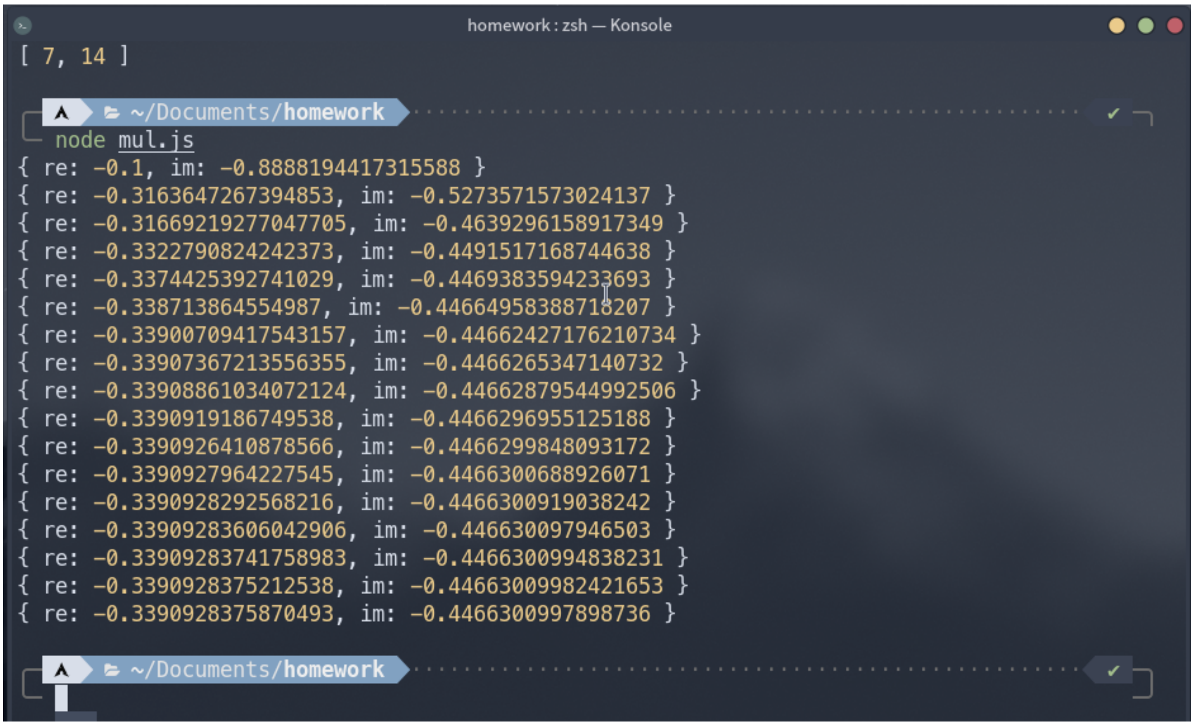
使用了mathjs库，由于javascript无法重载运算符，所以代码可读性较差，由于依赖问题，文件单独放在`mul.js`中

实验函数：

$$x^4 - 3x^3 + x^2 + x + 1$$

实验数据：0.5,-0.5,0，与书上相同

最大迭代次数：100，TOL:10^{-10}



迭代结果与书中大致相同

对比实验

实验函数为$f(x) = \cos(x) - x$，为了对比不动点方法，对于不动点使用了函数$f(x) = \cos(x)$。

最大迭代次数：100，双初始值算法：0,1，单初始值算法：0.5，TOL:10^{-10}

首先对比迭代次数

```
var b1=new bisection(the_fun3,0,1,100,Math.pow(10,-10))
console.log(b1.show())
console.log(b1.temp_step.length)

var f1=new fixed_point(the_fun3_fix,100,Math.pow(10,-10))
console.log(f1.show(0.5))
console.log(f1.p_list.length)

var n1=new Newton_mtd(the_fun3,the_fun3_grad,100,Math.pow(10,-10))
console.log(n1.show(0.5))
console.log(n1.p_list.length)

var s1=new secant_mtd(the_fun3,the_fun3_grad,100,Math.pow(10,-10))
console.log(s1.show(0,1))
console.log(s1.p_list.length)

var fal1=new false_posti(the_fun3,the_fun3_grad,20,Math.pow(10,-10))
console.log(fal1.show(0,1))
console.log(fal1.p_list.length)

var st1=new steffensen_mtd(the_fun3_fix,100,Math.pow(10,-10))
console.log(st1.show(0.5))
console.log(st1.p_list.length)
```

```
~/Documents/homework
node 1.js
0.7390851332165767
35
0.7390851332502528
57
0.7390851332151607
4
0.7390851332151607
5
0.7390851332129521
8
0.7390851332151607
3
```

而后比较运行时间，每个算法运行10^5次，单位：毫秒。

```
function time_avg(){
    var b1=new bisection(the_fun3,0,1,100,Math.pow(10,-10))
    var f1=new fixed_point(the_fun3_fix,100,Math.pow(10,-10))
    var n1=new Newton_mtd(the_fun3,the_fun3_grad,100,Math.pow(10,-10))
    var s1=new secant_mtd(the_fun3,the_fun3_grad,100,Math.pow(10,-10))
    var fal1=new false_posti(the_fun3,the_fun3_grad,20,Math.pow(10,-10))
    var st1=new steffensen_mtd(the_fun3_fix,100,Math.pow(10,-10))

    var d11=new Date()
    for(var i=0;i<Math.pow(10,5);i++){
        var temp=b1.show()
    }
    var d12=new Date()
    console.log(d12-d11)

    var d21=new Date()
    for(var i=0;i<Math.pow(10,5);i++){
        var temp=f1.show(0.5)
    }
```

```
    var d22=new Date()
    console.log(d22-d21)

    var d31=new Date()
    for(var i=0;i<Math.pow(10,5);i++){
        var temp=n1.show(0.5)
    }
    var d32=new Date()
    console.log(d32-d31)

    var d41=new Date()
    for(var i=0;i<Math.pow(10,5);i++){
        var temp=s1.show(0,1)
    }
    var d42=new Date()
    console.log(d42-d41)

    var d51=new Date()
    for(var i=0;i<Math.pow(10,5);i++){
        var temp=fall.show(0,1)
    }
    var d52=new Date()
    console.log(d52-d51)
    // return temp

    var d61=new Date()
    for(var i=0;i<Math.pow(10,5);i++){
        var temp=st1.show(0.5)
    }
    var d62=new Date()
    console.log(d62-d61)
}

time_avg()
```

可以看到，运行时间大致和迭代次数正相关。(由于运行时间和多因素有关，这里只显示运行一次结果，但同样环境多次运行时间大致如此)

附：图像代码

```
<head>
    <!-- <script src="https://cdn.plot.ly/plotly-2.10.0.min.js"></script> -->
    <script
src="https://cdn.bootcdn.net/ajax/libs/plotly.js/2.10.0/plotly.min.js">
</script>
    <script src="1.js"></script>
</head>
<style>
```

```
div.c1 {
    margin: 5px;
    border: 1px solid #ccc;
    float: left;
}

div.c1:hover {
    border: 1px solid #777;
}
</style>
<body>
<!-- <h1>avdhaudvd</h1> -->
    <div class="c1"><div id="gd"></div></div>
    <div class="c1"><div id="nd"></div></div>
    <div class="c1"><div id='std'></div></div>
    <!-- <div id="gd"></div> -->

    <script>
        var f1=new fixed_point(the_fun3_fix,100,Math.pow(10,-10))
        var step_f1=[]
        var res_f1=[]
        for (var step_temp=0.01;step_temp<1;step_temp=step_temp+0.01){
            step_f1.push(step_temp)
            var tem=f1.show(step_temp)
            res_f1.push(f1.p_list.length)
        }
        Plotly.newPlot("gd", /* JSON object */
                {
                "data": [{"x": step_f1,"y": res_f1 ,"type":'scatter'}],
                "layout": { "width": 600,
                            "height": 400,
                            title:'fixed point',
                            xaxis: {title: '初始数值'},
                            yaxis: {title: '迭代次数'},
                }});
    </script>



    <script>
        var n1=new Newton_mtd(the_fun3,the_fun3_grad,100,Math.pow(10,-10))
        var step_n1=[]
        var res_n1=[]
        for (var step_temp=0.01;step_temp<1;step_temp=step_temp+0.01){
            step_n1.push(step_temp)
            var tem=n1.show(step_temp)
                res_n1.push(n1.p_list.length)
        }
        Plotly.newPlot("nd", /* JSON object */
            {
                "data": [{"x": step_n1,"y": res_n1  ,"type":'scatter'}],
                "layout": { "width": 600,
                            "height": 400,
                            title:'newton mtd',
                            xaxis: {title: '初始数值'},
                            yaxis: {title: '迭代次数'},
                }});
    </script>
```

```
    <script>
        var st1=new steffensen_mtd(the_fun3_fix,100,Math.pow(10,-10))
        var step_st1=[]
        var res_st1=[]
        for (var step_temp=0.01;step_temp<1;step_temp=step_temp+0.01){
            step_st1.push(step_temp)
            var tem=st1.show(step_temp)
                res_st1.push(n1.p_list.length)
        }
        Plotly.newPlot("std", /* JSON object */
            {
                "data": [{"x": step_n1,"y": res_n1  ,"type":'scatter'}],
                "layout": { "width": 600,
                            "height": 400,
                            title:'steffensen_mtd',
                            xaxis: {title: '初始数值'},
                            yaxis: {title: '迭代次数'},
                }});
    </script>
</body>
```