数值分析第 2 次作业

211240021 田铭扬

§1 问题

考虑函数 $R(x)=\frac{1}{1+x^2}$，用下列条件做插值逼近，并与 R(x) 的图像比较：

(1) 用等距节点 $x_i=-5+i(i=0,1,\dots,10)$ 进行 10 次 Newton 插值；

(2) 用节点 $x_i=5\cos(\frac{2i+1}{42}\pi)(i=0,1,\dots,20)$ 进行 20 次 Lagrange 插值；

(3) 用等距节点 $x_i=-5+i(i=0,1,\dots,10)$ 进行分段线性插值；

(4) 用等距节点 $x_i=-5+i(i=0,1,\dots,10)$ 进行三次自然样条插值；

(5) 用等距节点 $x_i=-5+i(i=0,1,\dots,10)$ 进行分段三次 Hermite 插值。

§2 代码框架

由于需要画图，本次作业的代码使用 JavaScript 实现。解决五个问题的代码，有许多部分是相同的，包括最开始创建一个由（基点、基点处函数值）构成的数组，以及最后得画图部分等。如下：

```
<html>
<head> <title>Problem_X</title> </head>
<body>
  <canvas id="canvas" width="2100" height="1300"></canvas>
  <script>
    function func(_x){ return 1/(1+_x*_x); } //计算函数值
    /*(5)中需计算导数值的函数function func_d(_x){
    var _tmp=_x*_x+1; return -2*_x/_tmp/_tmp; }*/
    //同理，(4)中还需计算二阶导数值的子函数func_dd(_x)

    var canvas=document.getElementById("canvas");
    var ctx=canvas.getContext("2d");
    var base_point=[],base_diff=[],M=10; //(2)中M=20
    //会根据实际计算的需要会再创建若干数组

    for(var i=0;i<=M;i++){
      base_point.push({
        x:-5+i, //(2)中此处为Math.cos((2*i+1)/42*Math.PI)*5;
        y:func(-5+i) //同上
      }); //(5)中还需纪录导数值的成员dy
    }
    for(var i=0;i<M;i++)
      base_diff.push(base_point[i+1].x-base_point[i].x);

    //根据实际计算的需要，计算中间量

    function calcul(_x,_id){
      //_id参数仅(3)、(4)和(5)中需要，表示_x前一个插值基点的编号
      //利用中间量（和_id），计算自变量_x处的插值函数值
    }
// 以上是计算部分，以下是画图部分
```

```
    ctx.fillStyle="#000000"; ctx.font="30px courier";
    ctx.beginPath();
    ctx.moveTo(50,1050); ctx.lineTo(2050,1050);
    ctx.moveTo(1050,50); ctx.lineTo(1050,1250);
    ctx.stroke();
    for(var i=-5;i<=5;i++)
        ctx.fillText(i,30+(i+5)*200,1080);
    for(var i=-0.5;i<=2.5;i+=0.5){
      if(i==0) continue;
      ctx.fillText(i.toFixed(1),1055,1255-(i+0.5)*400);
    }// 绘制坐标系，(1) 需要 y 的取值范围更大，为 -0.5 置 2.5

    ctx.lineWidth=5; ctx.strokeStyle="#ff0000";
    ctx.beginPath();
    ctx.moveTo(50,1250-(func(-5)+0.5)*400);
    for(var i=-5;i<=5;i+=0.005){
      ctx.lineTo((i+5)*200+50,1250-(func(i)+0.5)*400);
      //(2) 到 (5)， (1) 中 y 取值范围更大
      // 故 lineTo 第 2 个参数改为 1250-(func(i)+0.5)*400，下同
    }
    ctx.stroke();// 绘制 R(x) 图像

    ctx.strokeStyle="#0000ff"; ctx.beginPath();
    ctx.moveTo(50,1250-(calcul(-5,0)+0.5)*400);
    var id=0;
    for(var i=-5;i<=5;i+=0.005){
      if(i>=base_point[id+1].x) id++;
      ctx.lineTo((i+5)*200+50,1250-(calcul(i,id)+0.5)*400);
    }
    ctx.stroke();// 绘制插值函数图像
  </script>
</body>
</html>
```

§3 算法分析与实现

·3.1 问题(1)

Newton 插值多项式的实现比较容易。只需要按照定义，利用二维递推的方法（即教材 P140 的算法）计算差商并储存在数组中，然后在调用函数 calcul()时用秦九韶算法计算即可。两部分的代码如下：

```
var d_quotient=[];
for(var i=0;i<=M;i++) d_quotient.push([]);
for(var i=0;i<=M;i++){
    d_quotient[i][0]=base_point[i].y;
    for(var j=1;j<=i;j++){
        d_quotient[i][j]=d_quotient[i][j-1]-d_quotient[i-1][j-1];
        d_quotient[i][j]/=(base_point[i].x-base_point[i-j].x);
    }
}
function calcul(_x){
    var result=d_quotient[M][M];
    for(var i=M-1;i>-1;i--)
        result=d_quotient[i][i]+result*(_x-base_point[i].x);
    return result;
}
```

·3.2 问题(2)

Lagrange 插值多项式的计算也是容易的，可以之间按照定义进行。不过注意到，x_i-x_j的值在计算中会多次用到，因此可以预先计算并存储在一个二维数组中，以节省时间。具体代码如下:

```
Var base_diff=[];
for(var i=0;i<=M;i++){
    base_diff.push([]);
    for(var j=0;j<=M;j++)
        base_diff[i].push(base_point[i].x-base_point[j].x);
}
function calcul(_x){
    var result=0,tmp_result;
    for(var i=0;i<=M;i++){
        tmp_result=base_point[i].y;
        for(var j=0;j<=M;j++){
            if(j==i) continue;
            tmp_result*=(_x-base_point[j].x);
            tmp_result/=base_diff[i][j];
        }
        result+=tmp_result;
    }
    return result;
}
```

·3.3 问题(3)

分段线性插值是五个问题中最容易计算的，且有与(2)同理的“空间换时间”方法，代码如下：

```
var _k=[];
for(var i=0;i<M;i++)
    _k.push((base_point[i].y-base_point[i+1].y)/(base_point[i].x-
base_point[i-1].x));
function calcul(_x,_id){
    var _k=;
    return _k[_id]*(_x-base_point[_id].x)+base_point[_id].y;
}
```

·3.4 问题(4)

问题(4)的代码较为复杂。首先，我们需要一个使用“追赶法”解三对角方程组的子函数，如下：

```
//Ax=b 系数矩阵 A 储存在 func_matrix[][]、常数向量 b 存在 func_constant[] 中
function solve_tridiagonal(){
    var _c=[],_d=[],_m=[]; // 结果储存在全局数组 result_m[] 中
    _c.push(func_matrix[0][1]/func_matrix[0][0]);
     for(var i=1;i<M-2;i++)
        _c.push(func_matrix[i][i+1]/(func_matrix[i][i]-
func_matrix[i][i-1]*_c[i-1]));

    _d.push(func_constant[0]/func_matrix[0][0]);
    for(var i=1;i<M-1;i++)
        _d.push((func_constant[i]-func_matrix[i][i-1]*_d[i-1])/(fun
c_matrix[i][i]-func_matrix[i][i-1]*_c[i-1]));

    _m.push(_d[M-2]);
    for(var i=M-3;i>-1;i--) _m.push(_d[i]-_c[i]*_m[M-3-i]);
    for(var i=0;i<M-1;i++) result_m.push(_m[M-2-i]);
    return ;
}
```

有了这个子函数后，代码的实现就比较容易了。只需要按公式(4.6.6)计算出func_matrix[][]与func_constant[]的各元素值，调用子函数计算。然后根据计算结果——即result_m[]各元素的值——计算（每一段）的插值多项式的各项系数，存储在一个全局的二维数组中。最后在calcul()函数中调用该数组的值，利用秦九韶算法进行计算即可。代码如下：

(注，本题由于插值基点等距，算法可以进行较大幅度的简化。考虑到实际运用中会插值基点不等距的一般情况，我在一小部分语句后的注释里，写了在此一般情况下的代码，对于其它语句是同理的。)

```
var d_quotient=[],func_matrix=[],func_constant=[];
var result_m=[],result_coefficient=[[],[],[],[]];
for(var i=0;i<M;i++)
    d_quotient.push(base_point[i+1].y-base_point[i].y);
    //d_quotient.push((base_point[i+1].y-base_point[i].y)/(base_poi
nt[i+1].x-base_point[i].x));

func_constant.push(6*(d_quotient[1]-d_quotient[0])-func_dd(-5));
//func_constant.push(6*(d_quotient[1]-d_quotient[0])-func_dd(-
5)*(base_point[1].x-base_point[0].x));
for(var i=1;i<M;i++)
    func_constant.push(6*(d_quotient[i+1]-d_quotient[i]));
func_constant.push(6*(d_quotient[M]-d_quotient[M-1])-func_dd(5));

for(var i=0;i<M;i++){
    func_matrix.push([]);
    for(var j=0;j<M;j++) func_matrix[i].push(0);
}
/*func_matrix[0][0]=2*(base_point[2].x-base_point[0].x);
func_matrix[0][1]=base_point[2].x-base_point[1].x;
for(var i=1;i<M-1;i++){
    func_matrix[i][i-1]=base_point[i+1].x-base_point[i].x;
    func_matrix[i][i]=2*(base_point[i+2].x-base_point[i].x);
    func_matrix[i][i+1]=base_point[i+2].x-base_point[i+1].x;
    }
func_matrix[M-1][8]=base_point[M].x-base_point[M-1].x;
func_matrix[M-1][M-1]=2(base_point[M].x-base_point[M-2].x);*/

func_matrix[0][0]=4; func_matrix[0][1]=1;
func_matrix[M-1][M-2]=1; func_matrix[M-1][M-1]=4;
for(var i=1;i<M-1;i++){
    func_matrix[i][i-1]=1;
    func_matrix[i][i]=4;
    func_matrix[i][i+1]=1;
}

result_m.push(func_dd(-5));
solve_tridiagonal();//见上文
result_m.push(func_dd(5));
for(var i=0;i<M;i++){
   result_coefficient[0].push((-result_m[i]+result_m[i+1])/6/*/(ba
se_point[i+1].x-base_point[i].x)*/);
   result_coefficient[1].push(result_m[i]/2);
   result_coefficient[2].push(d_quotient[i]-(result_m[i+1]/6+resul
t_m[i]/3)/*再*(base_point[i+1].x-base_point[i].x)*/);
    result_coefficient[3].push(base_point[i].y);
}
```

```
function calcul(_x,_id){
    var _tmp1=_x-base_point[_id].x,_tmp2=1,_result=0;
    for(var i=3;i>-1;i--){
        _result=_result+result_coefficient[i][_id]*_tmp2;
        _tmp2=_tmp2*_tmp1;
    }
    return _result;
}
```

·3.5 问题(5)

问题(5)也只需依据公式(4.5.7)进行计算即可。注意到，在 n=1 时，该公式可以做如下的变形以进一步简化计算。

$$H_3(x)=\sum_{i=1,2} y_i(1-2(x-x_i)l_i'(x))l_i^2(x)+\sum_{i=1,2} y_i'(x-x_i)l_i^2(x)$$

$$=y_1(1-2\frac{x-x_1}{x_1-x_2})(\frac{x-x_2}{x_1-x_2})^2+y_2(1-2\frac{x-x_2}{x_2-x_1})(\frac{x-x_1}{x_2-x_1})^2+y_1'(x-x_1)\frac{(x-x_2)^2}{(x_1-x_2)^2}+y_2'(x-x_2)\frac{(x-x_1)^2}{(x_2-x_1)^2}$$

$$=B^2(y_1(1-2A)+y_1'AC)+A^2(y_2(1+2B)+y_2'BC)\quad(\text{其中 } A=\frac{x-x_1}{x_1-x_2}\ B=\frac{x-x_2}{x_1-x_2}\ C=x_1-x_2)$$

```
var base_diff=[];
for(var i=0;i<M;i++)
    base_diff.push(base_point[i+1].x-base_point[i].x);

function calcul(_x,_id){
    var _result=0, _C=-base_diff[_id];// 多次用到的量，可以空间换时间
    var _A=(_x-base_point[_id].x)/_C, _B=(_x-base_point[_id+1].x)/_C;
    _result+=_B*_B*(base_point[_id].y*(1-2*_A)+base_point[_id].dy*_A*_C);
    _result+=_A*_A*(base_point[_id+1].y*(1+2*_B)+base_point[_id+1].dy*_B*_C);
    return _result;
}
```

§4 运行结果

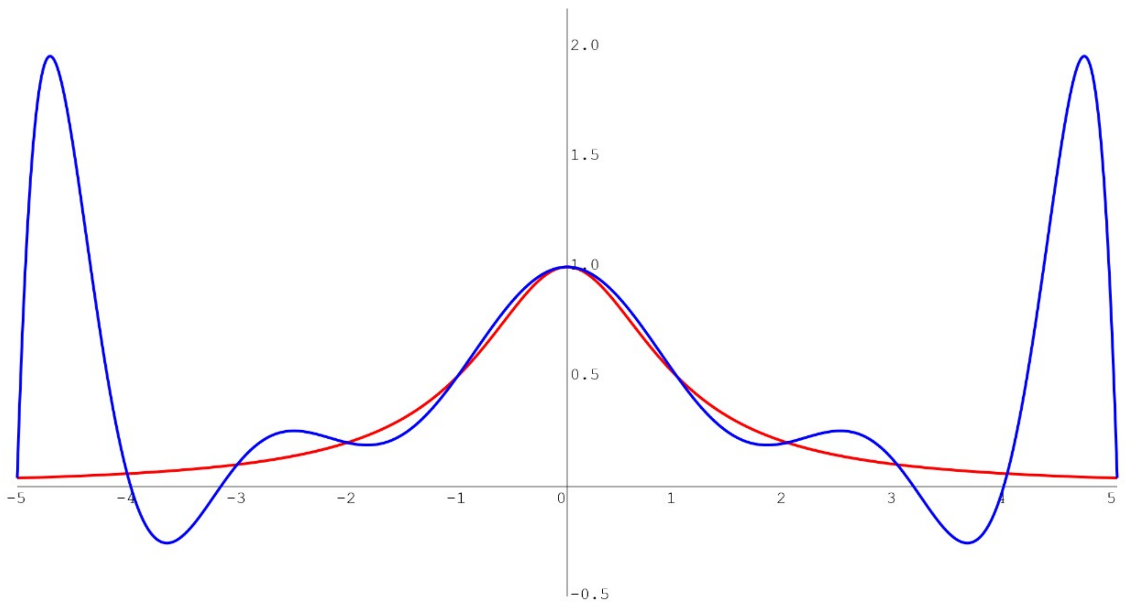

图 1：问题(1)的运行结果，注意到有明显的 Runge 现象

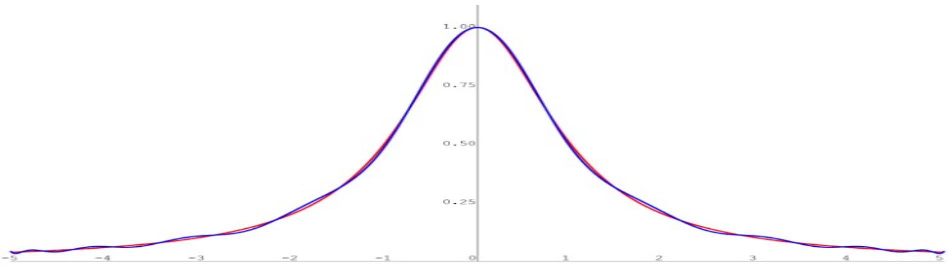

图 2：问题(2)的运行结果，由于插值基点的选取采用了 Chebshev 多项式的零点，余项（误差）上界较小，插值的表现更好

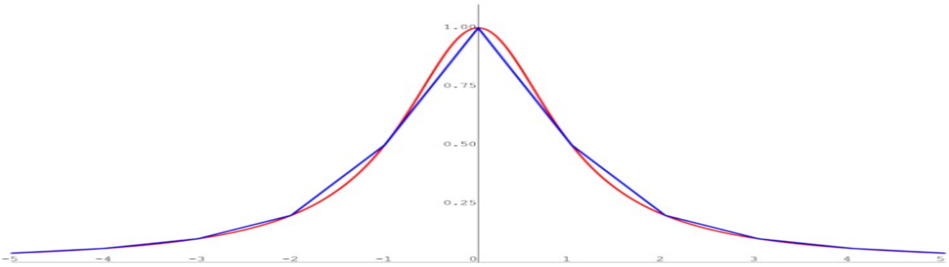

图 3：问题(3)的运行结果，分段线性插值实现最简单，结果不太理想

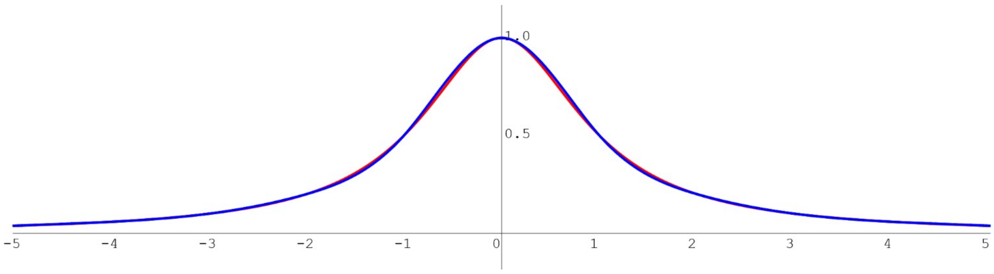

图 4：问题(4)的运行结果

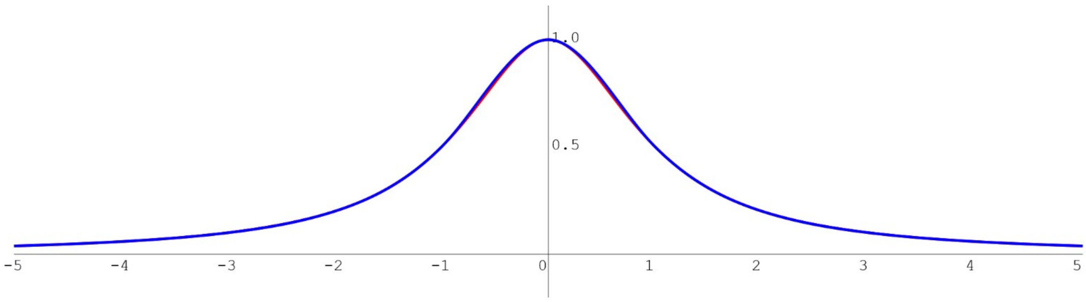

图 5：问题(5)的运行结果，可以看到，由于考虑到了导函数的值，样条插值与分段 Hermite 插值结果明显好于更“朴素”的 Lagrange 与 Newton 插值