

数值分析上机作业一

杨弦昊-2023P8270108012

中国机械总院

【题目一】求 $f(x) = \sin(\pi x)$ 在区间 $[0, 1]$ 上的二次最佳平方逼近多项式 $P^*(x)$ ，并绘出 $f(x)$ 和 $P^*(x)$ 的图像进行比较。

解：取空间 $\Phi = \text{Span}\{1, x, x^2\}$ ，权 $\omega(x) = 1$ ，按式1分别计算并存进数组 $A[3][3]$ 、 $B[3]$ 。

$$\begin{cases} (\varphi_i, \varphi_j) = \int_a^b x^{i+j} dx, \\ (f, \varphi_j) = \int_a^b f(x)x^j dx \end{cases} \quad (1)$$

$$\begin{aligned} (\varphi_0, \varphi_0) &= A[0][0], & (\varphi_1, \varphi_0) &= A[0][1], & (\varphi_2, \varphi_0) &= A[0][2], & (f, \varphi_0) &= B[0] \\ (\varphi_1, \varphi_1) &= A[1][1], & (\varphi_2, \varphi_1) &= A[1][2], & (f, \varphi_1) &= B[1] \\ (\varphi_2, \varphi_2) &= A[2][2], & (f, \varphi_2) &= B[2] \end{aligned} \quad (2)$$

则正规方程为

$$\begin{cases} A[0][0]C_0 + A[0][1]C_1 + A[0][2]C_2 = B[0] \\ A[1][0]C_0 + A[1][1]C_1 + A[1][2]C_2 = B[1] \\ A[2][0]C_0 + A[2][1]C_1 + A[2][2]C_2 = B[2] \end{cases} \quad (3)$$

解上述线性方程即得二次最佳平方逼近多项式 $P^*(x) = C_0 + C_1x + C_2x^2$ 。

通过编写 C 语言程序即可得到 $f(x) = \sin(\pi x)$ 在区间 $[0, 1]$ 上的二次最佳平方逼近多项式 $P^*(x) = -4.12251x^2 + 4.12251x - 0.050465$ ； $f(x)$ 和 $P^*(x)$ 的图像如图1所示；代码如下：

```
1 #include <stdio.h>
2 #include <math.h>
3
4 // 定义被积函数 x^(i+j)
5 double f1(int i, int j, double x) {
6     return pow(x, i + j);
7 }
8
9 // 定义被积函数 sin(πx) * x^i
10 double f2(int i, double x) {
11     return sin(M_PI * x) * pow(x, i);
12 }
13
14 double f3(double x){
15     return sin(M_PI * x);
16 }
17
18 // 梯形法则计算定积分
19 double integrate(double a, double b, int n, int i, double (*func)(int, double)) {
20     double h = (b - a) / n;
21     double sum = 0.0;
22
23     for (int k = 0; k <= n; k++) {
24         double x = a + k * h;
25         double value = func(i, x);
```

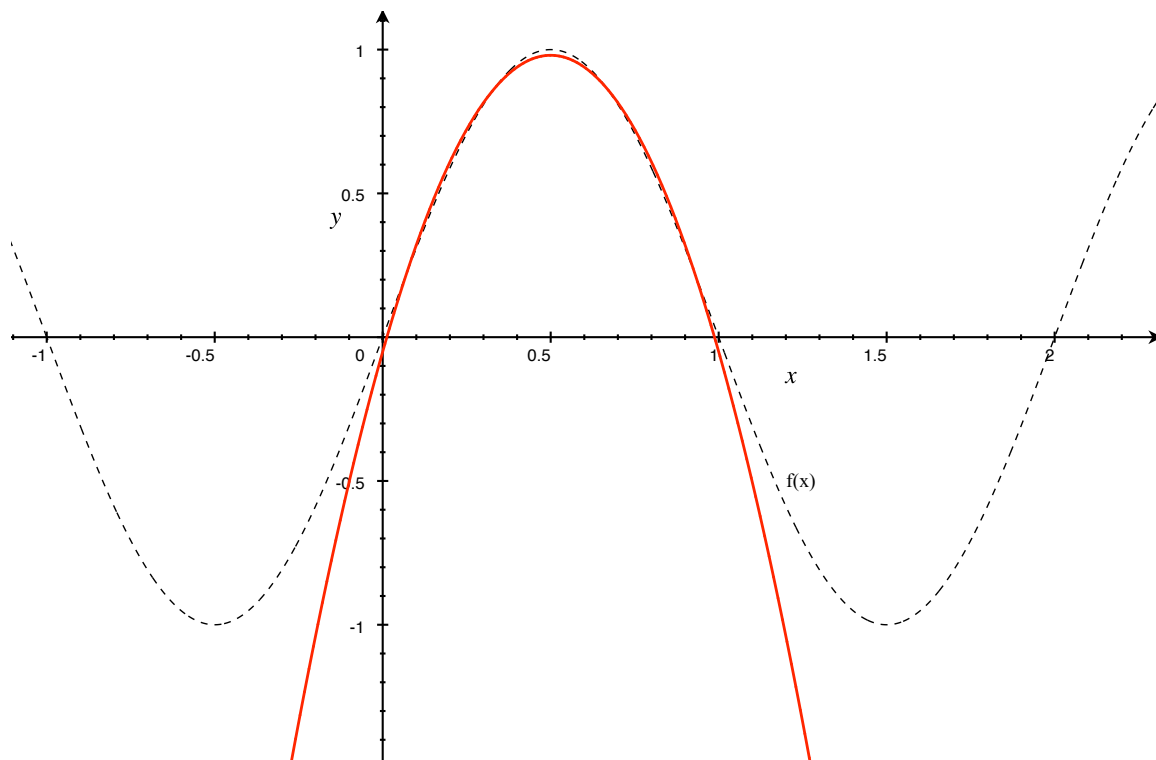


图 1: $f(x)$ 和 $P^*(x)$ 的图像

```

26
27     if (k == 0 || k == n) {
28         sum += value / 2.0;
29     } else {
30         sum += value;
31     }
32 }
33
34 return sum * h;
35 }
36
37 void solveLinearSystem(double A[3][3], double B[3], double x[3]) {
38     // 高斯消元法
39     int i, j, k;
40     double factor;
41
42     for (i = 0; i < 3; i++) {
43         for (j = i + 1; j < 3; j++) {
44             factor = A[j][i] / A[i][i];
45             for (k = i; k < 3; k++) {
46                 A[j][k] -= factor * A[i][k];
47             }
48             B[j] -= factor * B[i];
49         }
50     }
51
52     // 回代求解
53     for (i = 2; i >= 0; i--) {
54         x[i] = B[i] / A[i][i];
55         for (j = i - 1; j >= 0; j--) {
56             B[j] -= A[j][i] * x[i];
57         }
58     }
59 }
60
61 int main() {

```

```

62     int n = 3; // 多项式次数
63     int i;
64
65     double temp = 0;
66     temp = f3(0.5);
67     printf("%f\n",temp);
68
69     // 步骤1: 计算矩阵A
70     double A[3][3];
71     for (i = 0; i < n; i++) {
72         for (int j = 0; j < n; j++) {
73             A[i][j] = integrate(0.0, 1.0, 1000, i + j, f1);
74             printf("%f\n",A[i][j]);
75         }
76     }
77
78     // 步骤2: 计算数组B
79     double B[3];
80     for (i = 0; i < n; i++) {
81         B[i] = integrate(0.0, 1.0, 1000, i, f2);
82         printf("B数组=%f\n",B[i]);
83     }
84
85     // 步骤3: 解线性方程组  $AC = B$ 
86     double C[3];
87     solveLinearSystem(A, B, C);
88
89     // 步骤4: 打印多项式  $P(x)$ 
90     printf("P(x) = %.6f + %.6fx + %.6fx^2\n", C[0], C[1], C[2]);
91
92     return 0;
93 }

```

【题目二】对 Runge 函数 $R(x)$ (式 (2.5.2)), 利用下列条件作插值逼近, 并与 $R(x)$ 的图像进行比较。

- (1) 用等距节点 $x_i = -5 + i (i = 0, 1, 2, \dots, 10)$, 绘出它的 10 次 Newton 插值多项式的图像;
- (2) 用节点 $x_i = 5 \cos \left(\frac{2i+1}{42} \pi \right) (i = 0, 1, 2, \dots, 20)$, 绘出它的 20 次 Lagrange 插值多项式的图像;
- (3) 用等距节点 $x_i = -5 + i (i = 0, 1, 2, \dots, 10)$, 绘出它的分段线性插值函数的图像;
- (4) 用等距节点 $x_i = -5 + i (i = 0, 1, 2, \dots, 10)$, 绘出它的分段三次 Hermite 插值函数图像;
- (5) 用等距节点 $x_i = -5 + i (i = 0, 1, 2, \dots, 10)$, 绘出它的三次自然样条插值函数的图像。

解:

(1) 如图2所示, 可以看到 Newton 插值多项式图像在中部与 $R(x)$ 拟合程度较好, 而在两端由于计算的舍入误差会迅速扩大, 因此图像产生较大波动。

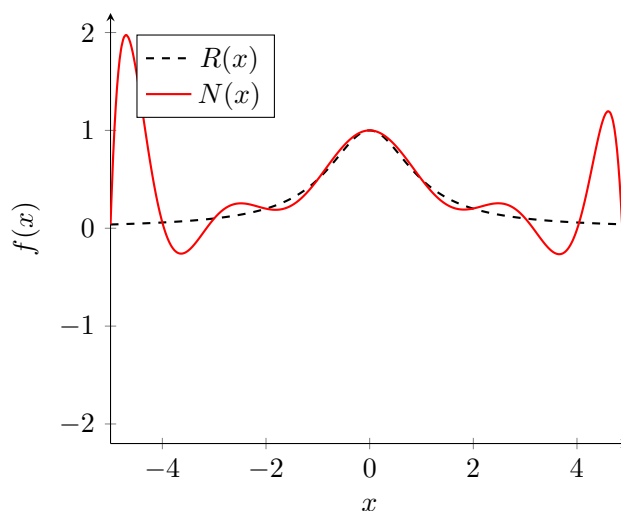


图 2: $R(x)$ 图像与 $N(x)$ 图像

代码如下:

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define N 11 // 插值节点的数量
5
6 double f(double x) {
7     return 1.0 / (1 + x * x);
8 }
9
10 int main() {
11     double x[N];
12     double A[N]; // 存储差商的数组
13     double y[N];
14     double TMP[N];
15     int i, j;
16
17     // 初始化节点
18     for (i = 0; i < N; i++) {
19         x[i] = -5 + i;
20         printf("x[%d]=%f\n", i, x[i]);
21     }
22
23     // 计算插值节点值 f(x_i)
24     for (int i = 0; i < N; i++) {
25         y[i] = f(x[i]);
26         printf("Y[%d]=%f\n", i, y[i]);
27     }
28
29     // 计算差商
```

```

30  A[0] = y[0];
31  for (int m = 1; m < N; m++) { // 外层m阶差商
32      double result_sum = 0;
33      for (int i = 0; i <= m; i++) { // 计算求和
34          double result_prod = 1;
35          double result_frac = 0;
36          for (int j = 0; j <= m; j++) { // 计算分母的连乘积
37              if (j != i) {
38                  result_prod *= x[i] - x[j];
39              }
40          }
41          result_frac = y[i] / result_prod;
42          result_sum += result_frac;
43      }
44      A[m] = result_sum;
45      printf("A[%d] = %f\n", m, A[m]);
46  }
47
48
49  // 输出多项式形式的结果
50  printf("N(x)=");
51  for(int k = 0;k < N; k++){
52      printf("(%f)",A[k]);
53      for(int n = 0;n < k; n++){
54          if(k > 0) printf("*");
55          printf("(x-%f)",x[n]);
56      }
57      if(k<N-1) printf("+");
58  }
59  printf("\n");
60
61  return 0;
62 }

```

(2) 如图3所示，与 Newton 插值类似，Lagrange 插值也会在两端产生波动。

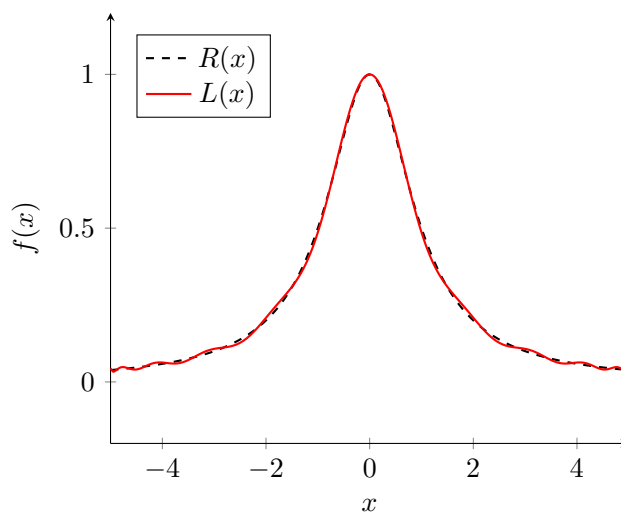




图 3: $R(x)$ 图像与 $L(x)$ 图像

代码如下:

```
1 #include <stdio.h>
2 #include <math.h>
3
4 int main() {
5     int n = 21;
6     double xi[21];
7     double yi[21];
8     double temp_result[21];
9
10    for (int i = 0; i < n; i++) {
11        xi[i] = 5 * cos(((2 * i + 1) / 42.0) * M_PI);
12        yi[i] = 1 / (1 + xi[i] * xi[i]);
13    }
14
15    // 为了简化计算先把\prod_{j \neq i} (x_i - x_j) 计算出来
16    for (int i = 0; i < n; i++) {
17        double result = 1;
18        for (int j = 0; j < n; j++) {
19            if (i != j) {
20                result *= xi[i] - xi[j];
21            }
22        }
23        temp_result[i] = result;
24        printf("%f, %f\n", temp_result[i], xi[i]);
25    }
26
27
28    // 计算插值多项式并在指定区间上进行插值
29    printf("L(x)=");
30    for (int i = 0; i < n; i++) {
31        printf("(%f)*(", yi[i]);
32        for (int j = 0; j < n; j++) {
33            if (j != i) {
34                if (j == 0 || i == 0) {
35                    printf("(x-%f)", xi[j]);
36                }
37                else {
38                    printf("*(x-%f)", xi[j]);
39                }
40            }
41        }
42    }
```



```
42     printf("/");
43     printf("(%f)",temp_result[i]);
44     if(i==20){
45         printf("");
46     }
47     else {
48         printf(")+");
49     }
50 }
51 printf("\n");
52 return 0;
53 }
```

(3) 如图4所示，线性插值基本能够拟合目标函数但是在插值节点处导数不连续因此并不是平滑过渡。

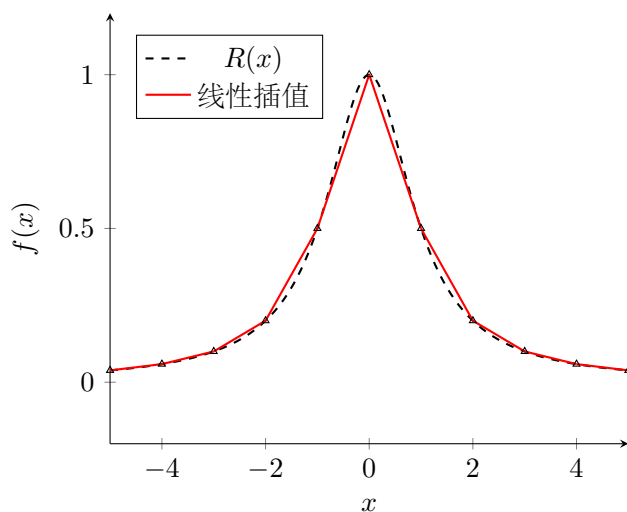


图 4: $R(x)$ 图像与分段线性函数图像

代码如下：

```
1 #include <stdio.h>
2
3 int main() {
4     // 等距节点
5     double x[] = {-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5};
6     int numPoints = 11;
7
8     // 输出分段插值多项式
9     for (int i = 0; i < numPoints - 1; i++) {
10         double x1 = x[i];
11         double x2 = x[i + 1];
12         double y1 = 1.0 / (1.0 + x1 * x1);
13         double y2 = 1.0 / (1.0 + x2 * x2);
14         double slope = (y2 - y1) / (x2 - x1);
15         double intercept = y1 - slope * x1;
16
17         printf("在区间 [%2f, %2f] 上的插值多项式: \n", x1, x2);
18         printf("f(x) = %.6fx + %.6f\n\n", slope, intercept);
19     }
20
21     return 0;
22 }
```


(4) 如图5所示，由图像可以看出 Hermite 插值生成的图像在插值节点处非常平滑。

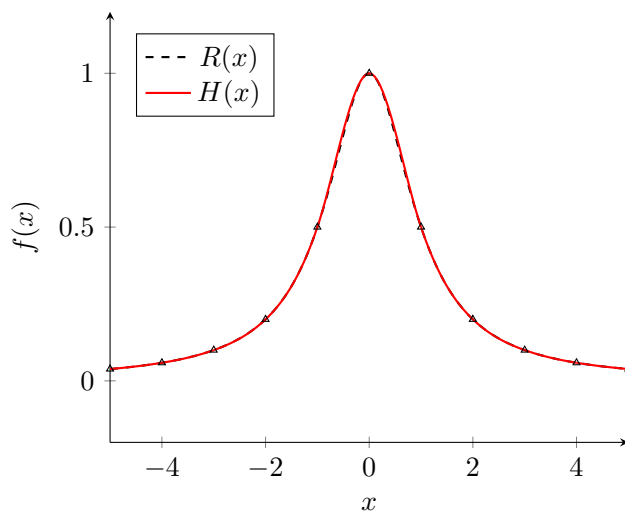


图 5: $R(x)$ 图像与 $H(x)$ 图像

代码如下：

```
1 #include <stdio.h>
2 #include <math.h>
3
4
5 #define N 11 // 插值节点的数量
6 // 目标函数
7 double targetFunction(double x) {
8     return 1.0 / (1.0 + x * x);
9 }
10
11 // 导数函数
12 double derivativeFunction(double x) {
13     return -2.0 * x / pow(1.0 + x * x, 2);
14 }
15
16 double x[] = {-5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5}; // 插值节点数组
17 int main() {
18     double y[11]; // 插值节点对应的函数值
19     double y_prime[11]; // 插值节点对应的导数值
20
21     // 初始化插值节点
22     for (int i = 0; i < N; i++) {
23         y[i] = targetFunction(x[i]);
24         y_prime[i] = derivativeFunction(x[i]);
25     }
26
27     // 输出插值节点数据
28     printf("插值节点和函数值: \n");
29     for (int i = 0; i < N; i++) {
30         printf("x[%d] = %f, y[%d] = %f, y'[%d] = %f\n", i, x[i], i, y[i], i, y_prime[i]);
31     }
32
33     // 分段三次埃尔米特插值
34
35     printf("\n分段三次埃尔米特插值表达式: \n");
36     for (int i = 0; i < N-1; i++) {
37         printf("H(x)=");
38         printf("(1+2*(x-%f)/(%f-%f))", x[i], x[i+1], x[i]);
39         printf(" * (((x-%f)/(%f-%f))^2)", x[i+1], x[i], x[i+1]);
40         printf(" * (%f)", y[i]);
41         printf("+");
```

```
42     printf("(1+2*(x-%f)/(%f-%f))",x[i+1],x[i],x[i+1]);
43     printf("*((x-%f)/(%f-%f))^2",x[i],x[i+1],x[i]);
44     printf("*(%f)",y[i+1]);
45     printf("+");
46     printf("(x-%f)",x[i]);
47     printf("*((x-%f)/(%f-%f))^2",x[i+1],x[i],x[i+1]);
48     printf("*(%f)",y_prime[i]);
49     printf("+");
50     printf("(x-%f)",x[i+1]);
51     printf("*((x-%f)/(%f-%f))^2",x[i],x[i+1],x[i]);
52     printf("*(%f)",y_prime[i+1]);
53     printf(" x在[%f,%f]\n",x[i],x[i+1]);
54 }
55 return 0;
56 }
```

(5) 如图6所示:

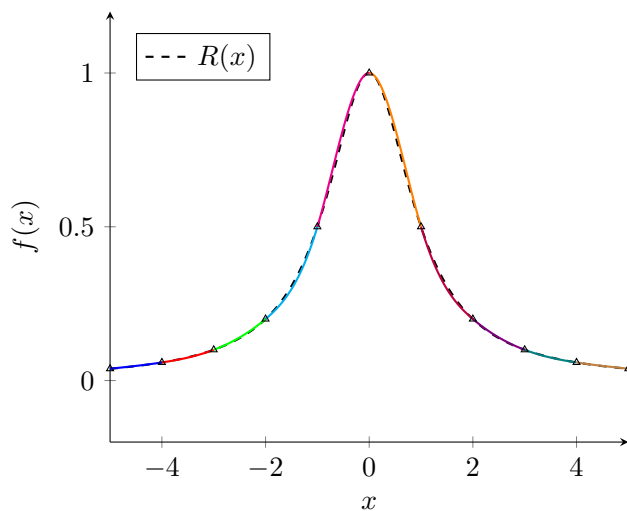


图 6: $R(x)$ 与 $S(x)$ 的图像

代码如下:

```
1 #include <stdio.h>
2 #include <math.h>
3
4 #define N 11 // 节点数量
5
6 double f(double x) {
7     return 1.0 / (1.0 + x * x);
8 }
9
10 int main() {
11     double x[N]; // 等距节点
12     double y[N]; // 存储函数值
13
14     for(int i = 0; i < N; i++) {
15         x[i] = -5 + i;
16     }
17
18     for (int i = 0; i < N; i++) {
19         y[i] = f(x[i]);
20         printf("x[%d] = %.4f  y[%d] = %.4f\n", i, x[i], i, y[i]);
21     }
22
23     // 计算三次样条插值
24     double h[N - 1];
25     for (int i = 0; i < N - 1; i++) {
26         h[i] = x[i + 1] - x[i];
27     }
28
29     double alpha[N - 1];
30     for (int i = 1; i < N - 1; i++) {
31         alpha[i] = (3.0 / h[i]) * (y[i + 1] - y[i]) - (3.0 / h[i - 1]) * (y[i] - y[i - 1]);
32     }
33
34     double l[N], mu[N], z[N];
35     l[0] = 1.0;
36     mu[0] = 0.0;
37     z[0] = 0.0;
38
39     for (int i = 1; i < N - 1; i++) {
40         l[i] = 2.0 * (x[i + 1] - x[i - 1]) - h[i - 1] * mu[i - 1];
41         mu[i] = h[i] / l[i];
```

```

42         z[i] = (alpha[i] - h[i - 1] * z[i - 1]) / l[i];
43     }
44
45     double c[N], b[N], d[N];
46     l[N - 1] = 1.0;
47     z[N - 1] = 0.0;
48     c[N - 1] = 0.0;
49
50     for (int j = N - 2; j >= 0; j--) {
51         c[j] = z[j] - mu[j] * c[j + 1];
52         b[j] = (y[j + 1] - y[j]) / h[j] - h[j] * (c[j + 1] + 2.0 * c[j]) / 3.0;
53         d[j] = (c[j + 1] - c[j]) / (3.0 * h[j]);
54     }
55
56     // 输出样条曲线表达式
57     printf("样条曲线表达式: \n");
58     for (int i = 0; i < N - 1; i++) {
59         printf("区间 %d: %.2f <= x <= %.2f\n", i, x[i], x[i + 1]);
60         printf("S(x) = %.4f + (%.4f)*(x - %.2f) + (%.4f)*(x - %.2f)^2 + (%.4f)*(x - %.2f)^3\n", y[i], b[i],
61             x[i], c[i], x[i], d[i], x[i]);
62     }
63     return 0;
64 }

```