《科学与工程计算》实验报告

|  |  |  |
| --- | --- | --- |
| 学  号：  1004161221 |  | 姓  名：    白云鹏 |

**实验一：**

**一、实验内容**

使用共轭梯度法求解  X =

**二、程序设计的基本思想、原理和算法描述：**

**1、算法**

**共轭梯度法，朴素矩阵乘法**

**2、函数组成**

1. **void** input();  //从键盘输入所需的矩阵
2. **void** getWork();  //程序构造题目给定矩阵
3. **void** init();  //初始化程序
4. **bool** run(**double** \*k);  //判断是否达到了精度要求
5. **double** fun1(**double** rk[maxn], **double** rk2[maxn]);  //行向量与列向量相乘，返回其结果
6. **void** fun2(**double** rk[maxn], **double** rk2[maxn][maxn], **double** qqq[maxn]);  //行向量与矩阵相乘，结果保存在qqq中
7. **void** fun3(**double** a, **double** rk[maxn], **double** rk\_1[maxn]);  //行向量与实数a相乘，结果保存在rk\_1中
8. **void** fun4(**double** rk[maxn][maxn], **double** dk[maxn], **double** ans[maxn]);  //矩阵与列向量相乘，结果保存在ans中
9. **void** solv();  //共轭梯度法的控制流程

**3、输入/输出设计**

**在这个问题中，由于共轭梯度法不仅可以解决固定的问题，所以设计了两种输入输出：**

**1.解决题目给定的问题时，犹豫问题的阶数不定，所以需要输入未知量的个数，在输入结束后，程序会自动的构造题目中的矩阵。**

**2.解决其他的问题，首先需要输入未知量的个数，接下来需要输入右端列向量，再输入对称正定的同阶矩阵A作为系数矩阵，最后输入x向量矩阵作为迭代的初始向量。**

**在输出上，除了最后的结果外，还额外打印了每一次迭代过程中解向量的值。**

**4、符号名说明**

**全局变量有以下几个：**

1. **double** B[maxn];  //储存右端列向量
2. **double** xk[maxn];  //储存迭代时的解向量
3. **double** q[maxn][maxn];  //储存左端的系数向量
4. **double** ans[maxn];  //储存最终结果向量
5. **double** eps = 0.005;  //精度控制

而在算法流程控制中，有：

1. **double** rk[maxn];  //储存迭代后的残差向量
2. **double** dk[maxn];  //储存迭代所需的梯度向量
3. **double** xk\_1[maxn];  //储存迭代后新的解向量
4. **double** rk\_1[maxn];  //储存迭代新的残差向量

**5、算法描述**

  假设所要解的线性方程组为:

最后的解向量为:

那么算法的流程如下：

*,*

*4.Recalculate the precision, here using the 2-Norm.*

*.*

**三、源程序及注释：**

1. #include <cmath>
2. #include <cstdio>
3. #include <cstring>
4. #include <iostream>
5. **using** **namespace** std;
6. **const** **int** maxn = 50;
7. **double** B[maxn];
8. **double** xk[maxn];
9. **double** q[maxn][maxn];
10. **double** ans[maxn];
11. **double** eps = 0.005;
12. **int** n;
13. **void** getWork() {
14. cout << "Input the nums of the unknown x:" << endl;
15. cin >> n;  //输入未知向量的个数
16. B[0] = 14;
17. B[n - 1] = 14;
18. **for** (**int** i = 1; i < n - 1; ++i) {
19. B[i] = 22;
20. }  //构造右端向量
21. q[0][0] = 6;
22. q[0][1] = 8;
23. q[n - 1][n - 1] = 6;
24. q[n - 1][n - 2] = 8;
25. **for** (**int** i = 1; i < n - 1; ++i) {
26. q[i][i] = 6;
27. q[i][i - 1] = 8;
28. q[i][i + 1] = 8;
29. }  //构造系数矩阵
30. }
32. **void** input() {
33. cout << "Input the nums of the unknown x:" << endl;
34. cin >> n;  //输入未知向量的个数
35. cout << "Input the b:" << endl;
36. **for** (**int** i = 0; i < n; ++i) {
37. cin >> B[i];
38. }  //输入右端向量
39. cout << "Input the A:" << endl;
40. cout << "The A should be symmetric" << endl;
41. **for** (**int** i = 0; i < n; ++i) {
42. **for** (**int** j = 0; j < n; ++j) {
43. cin >> q[i][j];
44. }
45. }  //输入系数矩阵
46. cout << "Input the initial x:" << endl;
47. **for** (**int** i = 0; i < n; ++i) {
48. cin >> xk[i];
49. }  //输入初始的迭代向量
50. }
52. **void** init() {
53. memset(B, 0, **sizeof** B);
54. memset(xk, 0, **sizeof** xk);
55. memset(q, 0, **sizeof** q);
56. **int** mode;
57. cout << "1.The martix is in the homework" << endl;
58. cout << "2.Input the martix" << endl;
59. cout << "Input the choice:";
60. cin >> mode;
61. **if** (mode == 1)
62. getWork();  //给定题目
63. **else**
64. input();  //手动输入
65. }
67. **bool** run(**double** \*k) {
68. **double** sum1 = 0;
69. **for** (**int** i = 0; i < n; ++i) {
70. sum1 += (k[i] \* k[i]);  //计算内积和
71. }
72. **return** sum1 >= eps \* eps;  //是否已经满足精度需求
73. }
75. **double** fun1(**double** rk[maxn], **double** rk2[maxn]) {  //朴素计算行向量与列向量的乘积
76. //由于1\*n的向量与n\*1的向量相乘，结果为1\*1的矩阵，即只有一个值，所以直接返回了这个值
77. **double** sum1 = 0;
78. **for** (**int** i = 0; i < n; ++i) {
79. sum1 += (rk[i] \* rk2[i]);
80. }
81. **return** sum1;
82. }
83. **void** fun2(**double** rk[maxn], **double** rk2[maxn][maxn], **double** qqq[maxn]) {
84. //计算行向量与矩阵的乘积，结果保存在qqq中
85. **for** (**int** i = 0; i < n; ++i) {
86. qqq[i] = 0;
87. **for** (**int** j = 0; j < n; ++j) {
88. qqq[i] += (rk2[j][i] \* rk[j]);
89. }
90. }
91. }
92. **void** fun3(**double** a, **double** rk[maxn], **double** rk\_1[maxn]) { //计算向量与实数的乘积，结果保存在rk\_1中
93. **for** (**int** i = 0; i < n; ++i) {
94. rk\_1[i] = xk[i];  //由于算法中需要再加xk，所以直接将操作写在这里，减少了一次循环
95. rk\_1[i] += rk[i] \* a;
96. }
97. }
98. **void** fun4(**double** rk[maxn][maxn], **double** dk[maxn], **double** ans[maxn]) {
99. //计算矩阵与列向量的乘积，结果保存在ans中.
100. **for** (**int** i = 0; i < n; ++i) {
101. ans[i] = 0;
102. **for** (**int** j = 0; j < n; ++j) {
103. ans[i] += rk[i][j] \* dk[j];
104. }
105. }
106. }
107. **void** solv() { //共轭梯度法的主要控制流程
108. **double** rk[maxn];
109. **double** dk[maxn];
110. **for** (**int** i = 0; i < n; ++i) {
111. rk[i] = B[i];
112. **double** sum1 = 0;
113. **for** (**int** j = 0; j < n; ++j) {
114. sum1 += q[i][j] \* xk[j];
115. }
116. rk[i] -= sum1;
117. dk[i] = rk[i];
118. }  //初始化rk和dk
119. **while** (1) {
120. **double** up = fun1(rk, rk);
121. **double** tem2[maxn];
122. memset(tem2, 0, **sizeof** tem2);
123. fun2(dk, q, tem2);
124. **double** down = fun1(tem2, dk);
125. **double** ak = up / down;   //计算ak
126. **double** xk\_1[maxn];
127. fun3(ak, dk, xk\_1);//计算xk+1
128. cout << "xk\_1:" << endl;
129. **for** (**int** i = 0; i < n; ++i) {
130. cout << xk\_1[i] << " ";
131. }  //输出xk+1
132. cout << endl;
133. memset(tem2, 0, **sizeof** tem2);
134. fun4(q, dk, tem2);
135. **double** rk\_1[maxn];
136. **for** (**int** i = 0; i < n; ++i) {
137. rk\_1[i] = rk[i] - ak \* tem2[i];
138. }  //计算rk+1
139. **if** (run(rk\_1) == 0) {
140. **for** (**int** i = 0; i < n; ++i) {
141. ans[i] = xk\_1[i];
142. }
143. **return**;
144. }  //判断是否满足精度需求，如果满足，直接返回
145. up = fun1(rk\_1, rk\_1);
146. down = fun1(rk, rk);
147. **double** bk = up / down;
148. **for** (**int** i = 0; i < n; ++i) {
149. dk[i] = rk\_1[i] + bk \* dk[i];
150. xk[i] = xk\_1[i];
151. rk[i] = rk\_1[i];
152. }  //更新dk,rk,xk
153. cout << endl;
154. }
155. }
157. **int** main() {
158. init();
159. solv();
160. cout << endl << endl;
161. cout << "The ans is:" << endl;
162. **for** (**int** i = 0; i < n; ++i) {
163. cout << ans[i] << " ";
164. }
165. cout << endl;
166. }

**四、运行输出结果：**

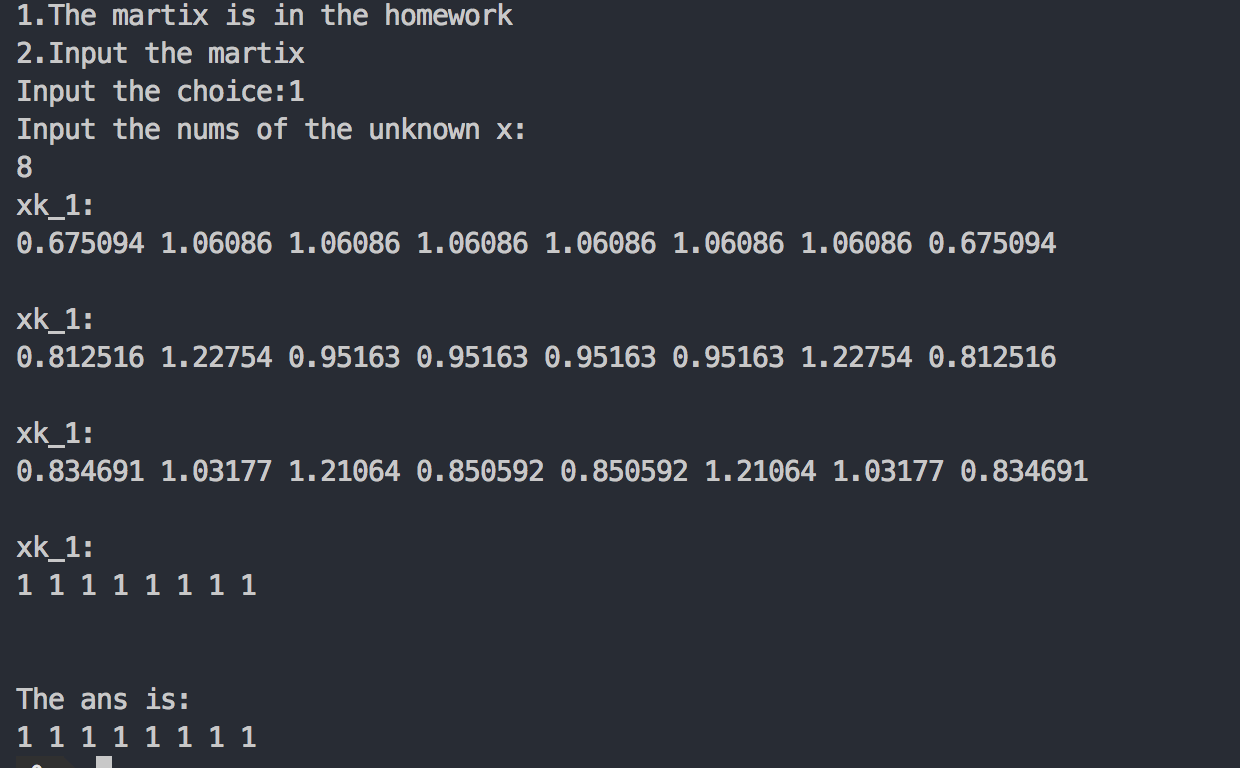


图 1 8阶给定矩阵结果

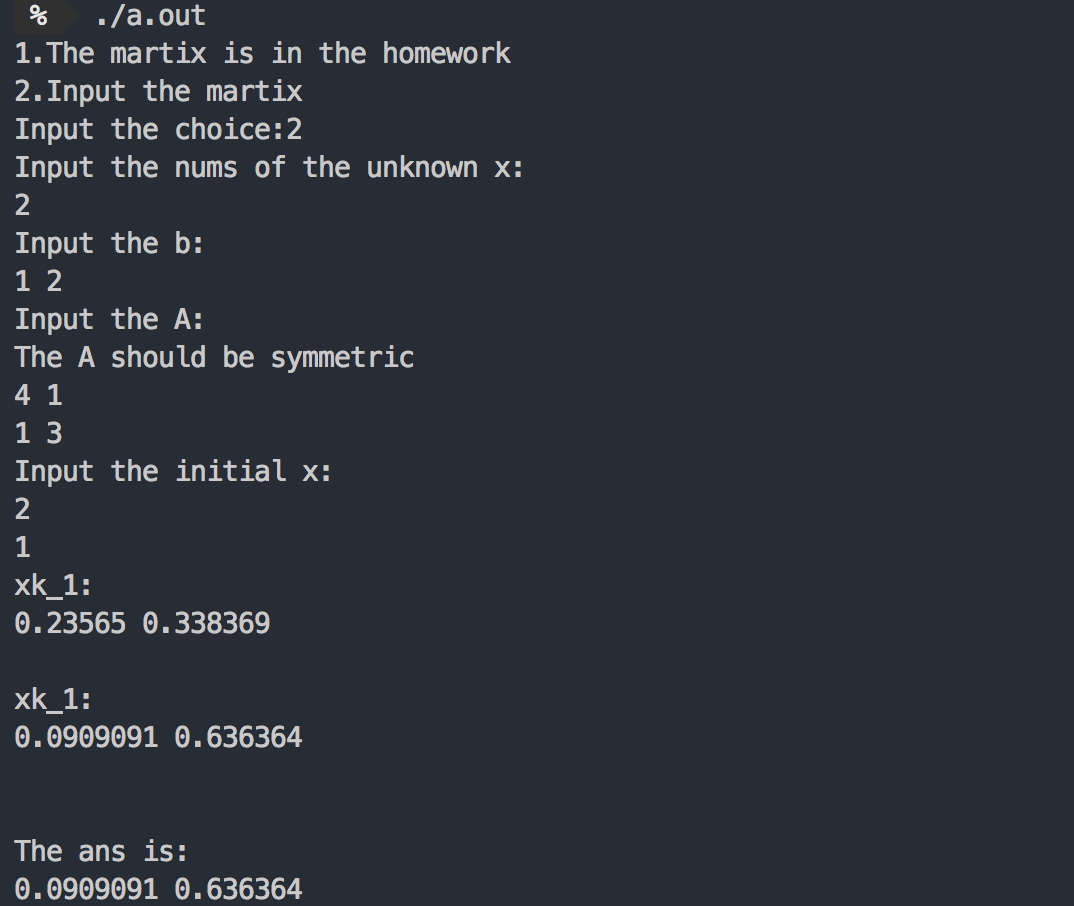


图 2 手动输入矩阵结果 （样例来源于百度百科）

**五、调试和运行程序过程中产生的问题及采取的措施：**

      1、

      2、

      3、

**六、对算法的程序的讨论、分析，改进设想，其它经验教训：**

      1、

      2、

      3、

**实验二：**

**一、实验内容**

对函数考虑区间[-1,1]上的一个等距划分，分点为，i = 0,1,2,…,n的插值节点，不断增加插值节点的个数，作三次样条插值。随节点个数的增加比较样条插值与拉格朗日多项式插值在0.98处的计算结果。

**二、程序设计的基本思想、原理和算法描述：**

**1、算法**

**三次样条插值，拉格朗日多项式插值，LU分解法**

**2、函数组成**

1. **void** LU(**int** n);  //使用LU分解法解矩阵
2. **double** f(**const** **double** &k);  //函数f(x)
3. **void** getxy();  //处理插值样本点
4. **void** geth();  //处理hx
5. **void** getdlymu();  //获取d,
6. **double** f2(**const** **double** &ttt);  //二次导函数f’’(x)
7. **void** getmartix();  //构造矩阵
8. **double** sx(**int** i);  //最终的插值函数
9. **double** solv();  //获取s(x)
10. **double** init();  //初始化算法

**3、输入/输出设计**

**由于f(x)已经固定在代码中，所以对于三次样条插值，在样本点上设计了两种输入方式：**

**1.首先是给定的题目，此时只需要输入样本点的个数，程序会自动计算所需要的x以及f(x)。**

**2.输入样本点的数目以及样本点。**

**最后再输入插值点。**

**输出上只输出了最后插值点在插值函数中对应的函数值。**

**4、符号名说明**

1. **const** **int** maxn = 50;  //最大样本点数
2. **double** d[maxn];  //右端列向量
3. **double** x[maxn];  //样本点x
4. **double** h[maxn];
5. **double** y[maxn];  //样本点y
6. **double** lymbda[maxn];  //系数矩阵中的
7. **double** martix[maxn][maxn];
8. **double** m[maxn];  //右端矩阵中的M
9. **double** u[maxn];  //系数矩阵中的
10. **double** testx;  //所要计算的x
11. **int** n;  //样本点值

**5、算法描述**

.

.计算H,.

.

：

**三、源程序及注释：**

1. #include <algorithm>
2. #include <cmath>
3. #include <cstring>
4. #include <iostream>
5. **using** **namespace** std;
6. **const** **int** maxn = 50;
7. **double** d[maxn];
8. **double** x[maxn];
9. **double** h[maxn];
10. **double** y[maxn];
11. **double** lymbda[maxn];
12. **double** martix[maxn][maxn];
13. **double** m[maxn];
14. **double** u[maxn];
15. **double** testx;  //要计算的x
16. **int** n;  //样本点个数
18. **double** f(**const** **double** &k) {  //函数f(x)
19. **return** (**double**) 1.0 / ((**double**) 1.0 + (**double**) 25 \* k \* k);
20. }
21. **void** getxy() {  //初始化x和y
22. **for** (**int** i = 0; i <= n; ++i) {
23. x[i] = (**double**) (-1) + (**double**) 2.0 \* i / n;
24. y[i] = f(x[i]);
25. }
26. }
27. **void** geth() {  //初始化h
28. **for** (**int** i = 1; i <= n; ++i) {
29. h[i] = x[i] - x[i - 1];
30. }
31. }
32. **void** getdlymu() {  //初始化右端向量，向量
33. **for** (**int** i = 1; i <= n; ++i) {
34. d[i] = (**double**) 6.0 \* ((y[i + 1] - y[i]) / h[i + 1] - (y[i] - y[i - 1]) / h[i]) / (h[i] + h[i + 1]);  //计算d向量
35. lymbda[i] = h[i + 1] / (h[i] + h[i + 1]);  //计算
36. u[i] = 1 - lymbda[i];  //计算向量
37. }
38. lymbda[0] = 0;
39. u[n] = 0;
40. }
41. **double** f2(**const** **double** &ttt) {  //f’’(x)
42. **double** tem = f(ttt);
43. **return** (**double**) (-50) \* (tem \* tem - (**double**) (100 \* ttt) \* tem \* tem \* tem);
44. }
45. **void** getmartix() {  //构造左端系数数组
46. m[0] = f2(-1);
47. m[n] = f2(1);
48. lymbda[0] = u[n] = 0;
49. d[0] = 2 \* m[0];
50. d[n] = 2 \* m[n];
51. martix[0][0] = 2;
52. martix[0][1] = lymbda[0];
53. martix[n][n] = 2;
54. martix[n][n - 1] = u[n];
55. **for** (**int** i = 1; i <= n - 1; ++i) {
56. martix[i][i] = 2;
57. martix[i][i - 1] = u[i];
58. martix[i][i + 1] = lymbda[i];
59. }
60. }
61. **double** sx(**int** i) {  //计算对应的插值函数值
62. **double** sum1 = 0;
63. **double** t1 = x[i] - testx;
64. **double** t2 = testx - x[i - 1];
65. sum1 += (m[i - 1] \* t1 \* t1 \* t1) / ((**double**) 6.0 \* h[i]) + m[i] \* (t2 \* t2 \* t2) / ((**double**) 6.0 \* h[i]);
66. sum1 += (y[i - 1] - m[i - 1] \* h[i] \* h[i] / (**double**) 6.0) \* (t1 / h[i]);
67. sum1 += (y[i] - m[i] \* h[i] \* h[i] / (**double**) 6.0) \* (t2 / h[i]);
68. **return** sum1;
69. }
70. **double** solv() {
71. //获取x对应的插值函数
72. **for** (**int** i = 1; i <= n; ++i) {
73. **if** (testx < x[i]) **return** sx(i);
74. }
75. }
76. **double** init() {
77. memset(x, 0, **sizeof** x);
78. memset(y, 0, **sizeof** y);
79. memset(h, 0, **sizeof** h);
80. memset(lymbda, 0, **sizeof** lymbda);
81. memset(u, 0, **sizeof** u);
82. memset(d, 0, **sizeof** d);
83. cout << "Input the nums of the sample point(start from 0):";
84. cin >> n;
85. **int** mode;
86. cout << "1.The martix in the homework:" << endl;
87. cout << "2.Input the sample points." << endl;
88. cout << "Input your choice:";
89. cin >> mode;
90. **if** (mode == 1) {
91. getxy();
92. } **else** **if** (mode == 2) {
93. cout << "Input the x vector:" << endl;
94. **for** (**int** i = 0; i < n; ++i) {
95. cin >> x[i];
96. }
97. cout << "Input the y vector:" << endl;
98. **for** (**int** i = 0; i < n; ++i) {
99. cin >> y[i];
100. }
101. }
102. geth();
103. getdlymu();
104. getmartix();
105. LU(n);
106. **while** (1) {
107. cout << "Input the x(var from -1 to 1):" << endl;
108. cin >> testx;
109. cout << solv() << endl;
110. }
111. }
113. **void** LU(**int** nn) {  //LU分解法解线性方程组，解向量储存在m数组中
114. ++nn;
115. **double** l[maxn][maxn];
116. **double** uu[maxn][maxn];
117. memset(l, 0, **sizeof** l);
118. memset(uu, 0, **sizeof** uu);
119. **for** (**int** i = 0; i < nn; ++i) {
120. uu[0][i] = martix[0][i];
121. l[i][i] = 1;
122. }
123. **for** (**int** i = 1; i < nn; ++i) {
124. l[i][0] = martix[i][0] / uu[0][0];
125. martix[i][0] /= martix[0][0];
126. }
127. **for** (**int** r = 1; r < nn; ++r) {
128. **for** (**int** i = r; i < nn; ++i) {
129. **double** tem = 0;
130. **double** tem2 = 0;
131. **for** (**int** k = 0; k < r; ++k) {
132. tem2 += (martix[r][k] \* martix[k][i]);
133. tem += (l[r][k] \* uu[k][i]);
134. }
135. uu[r][i] = martix[r][i] - tem;
136. martix[r][i] -= tem2;
137. }
138. **if** (r != nn) {
139. **for** (**int** i = r + 1; i < nn; ++i) {
140. **double** tem = 0;
141. **double** tem2 = 0;
142. **for** (**int** k = 0; k < r; ++k) {
143. tem2 += martix[i][k] \* uu[k][r];
144. tem += l[i][k] \* uu[k][r];
145. }
146. l[i][r] = (martix[i][r] - tem) / uu[r][r];
147. martix[i][r] -= tem2;
148. martix[i][r] /= martix[r][r];
149. }
150. }
151. }
152. **double** yy[maxn];
153. memset(yy, 0, **sizeof** yy);
154. yy[0] = d[0];
155. **for** (**int** i = 1; i < nn; ++i) {
156. **for** (**int** j = 0; j < i; ++j) {
157. d[i] -= l[i][j] \* yy[j];
158. }
159. yy[i] = d[i];
160. }
161. m[nn - 1] = yy[nn - 1] / uu[nn - 1][nn - 1];
162. **for** (**int** i = nn - 1; i >= 0; --i) {
163. **for** (**int** k = i + 1; k < nn; ++k) {
164. yy[i] -= uu[i][k] \* m[k];
165. }
166. m[i] = yy[i] / uu[i][i];
167. }
168. }
169. **int** main() {
170. init();
171. }

**四、运行输出结果：**

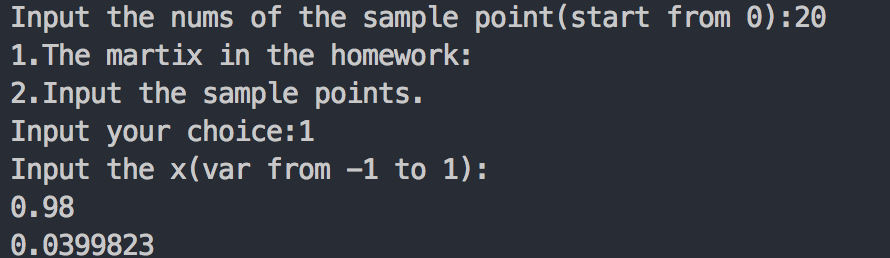


图 3 21个样本点时0.98的值

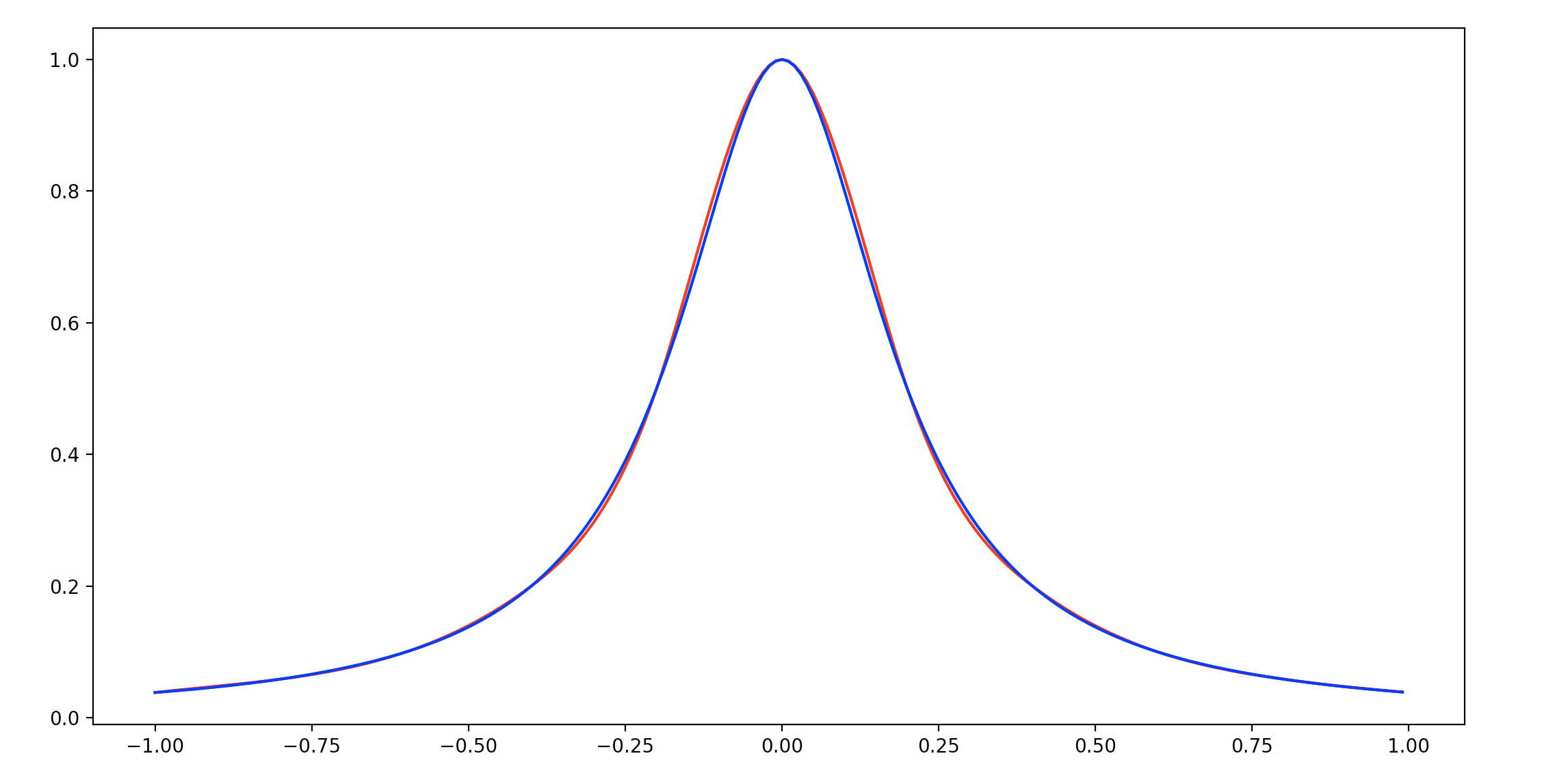


图 4 11个样本点时，插值函数与原函数的图像(蓝线为原函数)

**五、调试和运行程序过程中产生的问题及采取的措施：**

      1、

      2、

      3、

**六、对算法的程序的讨论、分析，改进设想，其它经验教训：**

      1、

      2、

      3、

实验三：

**一、实验内容**

**相同条件下对龙贝格积分以及变步长辛普森积分进行对比。**

**二、程序设计的基本思想、原理和算法描述：**

**1、算法**

**2、函数组成**

**3、输入/输出设计**

本程序通过键盘进行输入、屏幕进行输出。

（根据实际程序情况，还可以选择随机产生输入数据、将输出数据输出到文件等其它方式）

**4、符号名说明**

**5、算法描述**

**三、源程序及注释：**

1. #include <algorithm>
2. #include <cmath>
3. #include <cstdio>
4. #include <iostream>
5. **using** **namespace** std;
6. **const** **double** low = 0;
7. **const** **double** high = 1;
8. **const** **int** maxn = 50;
9. **const** **double** eps = 1e-8;
10. **double** T[maxn];
11. **double** S[maxn];
12. **double** R[maxn];
13. **double** C[maxn];
14. **double** fx(**const** **double** &t) {
15. **return** (**double**) 1.0 / (t + (**double**) 1.0);
16. }
17. **bool** run(**const** **double** &last, **const** **double** &rk) {
18. **if** (abs(last - rk) < eps)
19. **return** 0;
20. **else**
21. **return** 1;
22. }
23. **double** romberg(**int** k) {
24. k += 3;
25. **double** h = (high - low);
26. T[0] = h / 2 \* (fx(low) + fx(high));
27. **for** (**int** j = 1; j < k; ++j) {
28. **double** sum1 = 0;
29. **double** now = low;
30. **while** (now < high) {
31. sum1 += fx(now);
32. now += h;
33. }
34. T[j] = (T[j - 1] / 2 + h \* sum1 / 2);
35. h /= 2;
36. }
37. **for** (**int** i = 0; i < k - 1; ++i) {
38. S[i] = T[i + 1] + (T[i + 1] - T[i]) / (**double**) 3.0;
39. }
40. **for** (**int** i = 0; i < k - 2; ++i) {
41. C[i] = S[i + 1] + (S[i + 1] - S[i]) / (**double**) 15.0;
42. }
43. **int** i;
44. **for** (i = 0; i < k - 3; ++i) {
45. R[i] = C[i + 1] + (C[i + 1] - C[i]) / (**double**) 63.0;
46. }
47. **return** R[k - 4];
48. }
49. **void** init() {
50. T[1] = (high - low) \* (fx(high) + fx(low)) / 2;
51. }
52. **double** solv() {
53. init();
54. **int** k = 1;
55. **double** last = 0;
56. **double** rk = 0;
57. **do** {
58. last = rk;
59. rk = romberg(k++);
60. cout << rk << endl;
61. } **while** (run(last, rk));
62. **return** rk;
63. }
64. **int** main() {
65. **double** ans = solv();
66. printf("%.8lf", ans);
67. }

**四、运行输出结果：**

      （贴出程序运行完成时的屏幕截图或者输出文件的内容）

**五、调试和运行程序过程中产生的问题及采取的措施：**

      1、

      2、

      3、

**六、对算法的程序的讨论、分析，改进设想，其它经验教训：**

      1、

      2、

      3、

实验四：

实验三：

**一、实验内容**

**相同条件下对龙贝格积分以及变步长辛普森积分进行对比。**

**二、程序设计的基本思想、原理和算法描述：**

（包括程序的算法、函数组成、输入/输出设计、符号名说明等）

**1、算法**

**2、函数组成**

**3、输入/输出设计**

本程序通过键盘进行输入、屏幕进行输出。

（根据实际程序情况，还可以选择随机产生输入数据、将输出数据输出到文件等其它方式）

**4、符号名说明**

**5、算法描述**

**三、源程序及注释：**

**四、运行输出结果：**

      （贴出程序运行完成时的屏幕截图或者输出文件的内容）

**五、调试和运行程序过程中产生的问题及采取的措施：**

      1、

      2、

      3、

**六、对算法的程序的讨论、分析，改进设想，其它经验教训：**

      1、

      2、

      3、