作业四：插值算法

英才 1701 赵鹏威 U201710152

2019 年 10 月 16 日

目录

1 引言

实验中得到的数据都是离散的，如果想知道测量位置以外的点的值，或者需要求积分导数，就需要有连续的函数形式. 一种做法是拟合，但是拟合得到的函数不一定经过给定的数据点. 如果要求函数必须经过给定的数据点，就需要使用插值算法.

2 问题描述

问题 1. 使用拉格朗日插值、牛顿插值和分段立方样条插值方法求下列数据点的插值函数

$$y_i = \frac{1}{1+x_i^2}, \quad x_i = -5 + \frac{5-(-5)}{15} i, \quad i = 0, 1, \ldots, 15.$$

这个问题的数据点一共有 16 个，我们知道的是这 16 个点的 x 坐标和函数值 y，不知道任何关于导数的信息. 这里需要说明的是，尽管问题的描述让我们感觉原函数是

$$y = f(x) = \frac{1}{1+x^2}, \quad x \in [-5, 5],$$

但是实际上我们并不知道原函数是什么，因为任何经过这 16 个点的函数都可能是原函数. 但是，作为对各种插值方法的分析，下面我们会将插值函数与 $f(x) = 1/(1+x^2)$ 画在同一张图里对比.

3 使用的误差分析公式

由于本次作业使用的误差估计式与课件上的有一些区别，因此单独作为一节来说明.

3.1 拉格朗日插值和牛顿插值

对于拉格朗日插值和牛顿插值，它们的真实误差（与真值的差）可以写成下面的形式

$$R_{n+1}(x) = \frac{f^{(n+1)}(\xi)}{(n+1)!} (x - x_0) \cdots (x - x_n).$$

但是这个公式需要知道原函数，实际不可操作. 因此有下面的做法. 假设有 $n+1$ 个数据点，用前 n 个和后 n 个分别计算插值函数，近似认为 $f^{(n+1)}(\xi)$ 相等，可以得到下面的误差估计式

$$R_{n+1}(x) = \frac{x - x_0}{x_0 - x_{n+1}} (L_n(x) - L'_n(x)).$$

上面是课件中给出的方法. 但是，这个误差估计式有一个缺点：当插值的数据点为偶数个并且关于 y 轴对称时，算出的误差恒为 0. 下面来证明这一点，只要证明两次插值的函数完全一样即可.

问题 2. 给定 $2n$ 个插值数据点

$$(-x_n, y_n),\ (-x_{n-1}, y_{n-1}),\ \ldots,\ (-x_2, y_2),\ (-x_1, y_1),\ (x_1, y_1),\ (x_2, y_2),\ \ldots, (x_{n-1}, y_{n-1}),\ (x_n, y_n)$$

证明使用前 $2n-1$ 个数据点

$$(-x_n, y_n),\ (-x_{n-1}, y_{n-1}),\ \ldots,\ (-x_2, y_2),\ (-x_1, y_1),\ (x_1, y_1),\ (x_2, y_2),\ \ldots, (x_{n-1}, y_{n-1})$$

得到的拉格朗日插值（或牛顿插值）函数 $L_{-1}(x)$，与使用后 $2n-1$ 个数据点

$$(-x_{n-1}, y_{n-1}),\ \ldots,\ (-x_2, y_2),\ (-x_1, y_1),\ (x_1, y_1),\ (x_2, y_2),\ \ldots, (x_{n-1}, y_{n-1}),\ (x_n, y_n)$$

得到的拉格朗日插值（或牛顿插值）函数 $L_{+1}(x)$ 相等.

证明. 由于拉格朗日插值和牛顿插值等价，下面使用牛顿插值来证明. 假设使用中间 $2n-2$ 个数据点

$$(-x_{n-1}, y_{n-1}),\ \ldots,\ (-x_2, y_2),\ (-x_1, y_1),\ (x_1, y_1),\ (x_2, y_2),\ \ldots, (x_{n-1}, y_{n-1})$$

得到的插值函数为 $L_0(x)$. 由于牛顿插值的特性，增加一个数据点再插值只需要在原来的插值函数上增加一项，可以知道

$$L_{-1}(x) = L_0(x) + q_{-1}(x), \quad L_{+1}(x) = L_0(x) + q_{+1}(x)$$

其中这些加上的项的表达式是可以算出的

$$q_{-1}(x) = f[-x_n, -x_{n-1}, \ldots, x_{n-1}]\Pi_{i=1}^{n-1}(x^2 - x_i^2)$$
$$q_{+1}(x) = f[-x_{n-1}, \ldots,\ x_{n-1},\ x_n]\Pi_{i=1}^{n-1}(x^2 - x_i^2)$$

那么只要证明 $f[-x_n, -x_{n-1}, \ldots, x_{n-1}] = f[-x_{n-1}, \ldots,\ x_{n-1},\ x_n]$. 可以发现，只要将等号左边的差商括号里的每一项取相反数，就得到了等号右边的差商. 利用偶函数差商的性质：

$$f[-x_1, \ldots, -x_n] = (-1)^{n+1} f[x_1, \ldots, x_n],$$

立即可以证明 $f[-x_n, -x_{n-1}, \ldots, x_{n-1}] = f[-x_{n-1}, \ldots,\ x_{n-1},\ x_n]$. 因此 $L_{-1}(x) = L_{+1}(x)$. 证毕.

不幸的是，这次作业给出的数据点刚好是关于 y 轴对称的，因此我们需要换一种误差估计公式. 比如有 n 个数据点，用前 $n-2$ 个点插值得到 $L(x)$，用后 $n-2$ 个点插值得到 $L'(x)$，可以推导下面的误差估计式

$$R = f(x) - \frac{L(x) + L'(x)}{2} = \frac{(x - x_n)(x - x_{n-1}) + (x - x_1)(x - x_2)}{(x - x_n)(x - x_{n-1}) - (x - x_1)(x - x_2)} \frac{L(x) - L'(x)}{2}$$

下面在估计拉格朗日插值和牛顿插值的误差时均使用这个误差估计式.

3.2 分段立方样条插值

由于分段立方样条插值的每一段都是一个两点三次 Hermit 插值. Hermit 插值的误差估计式同样含有原函数的导数项，但是如果按照上面的方法尝试用两个插值来消去这个导数项，就会因为插值的点只有一个而产生较大的误差. 另外由于没有找到较好的误差估计方法，本次作业的分段立方样条插值将使用真实误差，其中原函数认为是

$$f(x) = \frac{1}{1 + x^2}.$$

4 程序实现

所有的插值算法都用 function 实现，接受数据点参数 sample 和一个 x 值，函数用指定的插值算法计算出插值函数在 x 点的值. 另外，函数返回的类型是另外定义的 result 类型，可以同时储存值和误差. 函数还接受一个参数，require_error，如果这个参数为真，那么会计算误差，否则不会. require_error 默认为真. 完整代码见附录.

4.1 拉格朗日插值

拉格朗日插值，流程图见图1，代码见 Listing 1.

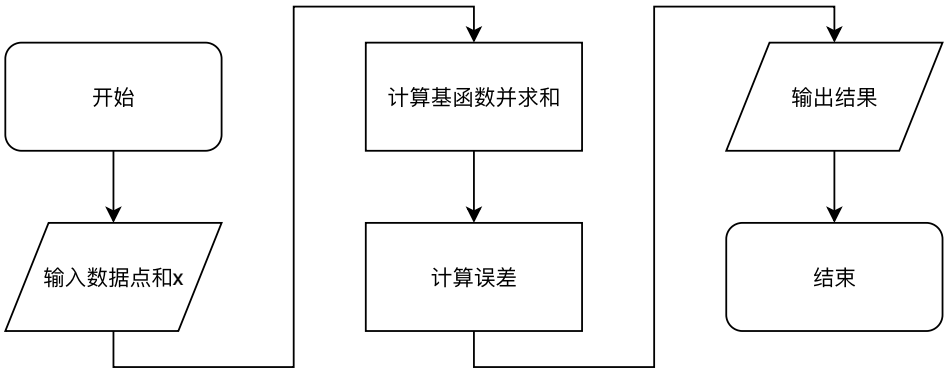

图 1: 拉格朗日插值流程图

Listing 1: **拉格朗日插值**

```
recursive type(result) function lagrange(sample, x, require_error) result(y)
    implicit none
    real(8), dimension(*, *), intent(in) :: sample(:, :)
    real(8), intent(in) :: x
    logical, optional :: require_error

    real(8) :: item
    type(result) :: L, R
    integer :: i, j, nos

    if (.not. present(require_error)) then
        require_error = .true.
    end if

    nos = size(sample, dim=1)

    y%value = 0.0d0
    do i = 1, nos
```

```
19          item = 1.0d0
20          do j = 1, i-1
21              item = item * (x - sample(j, 1)) / (sample(i, 1) - sample(j, 1))
22          end do
23          do j = i+1, nos
24              item = item * (x - sample(j, 1)) / (sample(i, 1) - sample(j, 1))
25          end do
26          y%value = y%value + item * sample(i, 2)
27      end do
28      if (require_error) then
29          L = lagrange(sample(1:nos-2, :), x, require_error=.false.)
30          R = lagrange(sample(3:nos , :), x, require_error=.false.)
31          y%error = (x-sample(nos, 1))*(x-sample(nos-1, 1))/&
32          ((x-sample(nos, 1))*(x-sample(nos-1, 1))-(x-sample(1, 1))*(x-sample(2, 1)))*L%value&
33           - (x-sample(1, 1))*(x-sample(2, 1))/&
34          ((x-sample(nos, 1))*(x-sample(nos-1, 1))-(x-sample(1, 1))*(x-sample(2, 1)))*R%value&
35          -(L%value + R%value)/2.0d0
36      end if
37
38      return
39  end function
```

4.2 牛顿插值

牛顿插值需要用到差商，我选择的是写一个计算差商的函数，每次需要的时候就调用这个函数. 虽然这样做会降低效率，但是程序看起来会更简单，并且只需要在拉格朗日插值代码的基础上稍作一些修改即可. 差商用递归实现，代码见 Listing 2. 牛顿插值的流程图见2，代码见 Listing 3.

Listing 2: **差商**

```
1   recursive real(8) function diff_quot(points) result(dq)
2       implicit none
3       real(8), dimension(*, *), intent(in) :: points(:, :)
4
5       integer :: ndim
6
7       ndim = size(points, dim=1)
8
9       if (ndim == 1) then
10          dq = points(1, 2)
11      else if (ndim >= 2) then
12          dq = (diff_quot(points(2:ndim, :)) - diff_quot(points(1:ndim-1, :))) / (points(ndim,
                1) - points(1, 1))
```

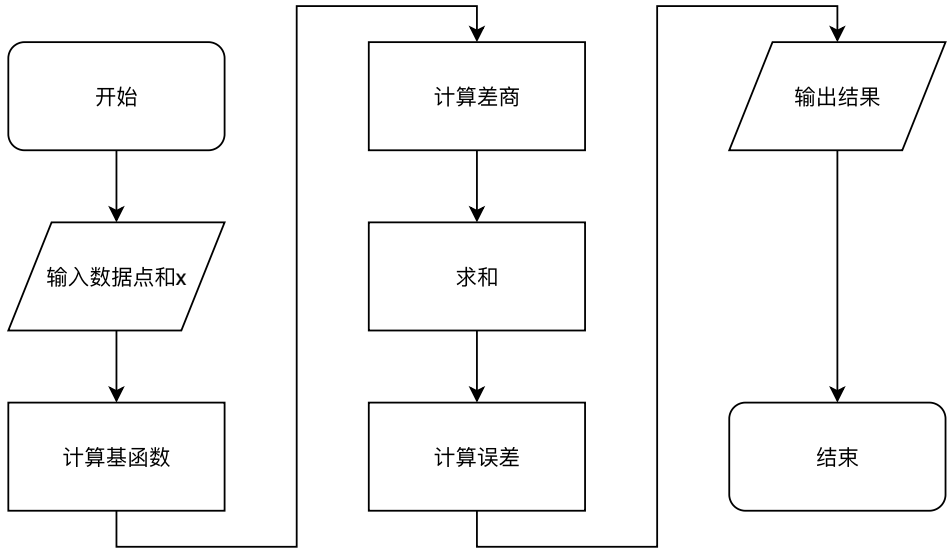

图 2: 牛顿插值流程图

```
    else
        stop "ERROR: the size of list for differential quotient must greater then 2"
    end if

    return
end function
```

Listing 3: **牛顿插值**

```
recursive type(result) function newton(sample, x, require_error) result(y)
    implicit none
    real(8), dimension(*, *), intent(in) :: sample(:, :)
    real(8), intent(in) :: x
    logical, optional :: require_error

    real(8) :: item
    type(result) :: L, R
    integer :: i, j, nos

    if (.not. present(require_error)) then
        require_error = .true.
    end if

    nos = size(sample, dim=1)

    y%value = 0.0d0
```

```
    do i = 1, nos
        item = 1.0d0
        do j = 1, i-1
            item = item * (x - sample(j, 1))
        end do
        y%value = y%value + item * diff_quot(sample(1:i, :))
    end do
    if (require_error) then
        L = newton(sample(1:nos-2, :), x, require_error=.false.)
        R = newton(sample(3:nos , :), x, require_error=.false.)
        y%error = (x-sample(nos, 1))*(x-sample(nos-1, 1))/&
        ((x-sample(nos, 1))*(x-sample(nos-1, 1))-(x-sample(1, 1))*(x-sample(2, 1)))*L%value&
         - (x-sample(1, 1))*(x-sample(2, 1))/&
        ((x-sample(nos, 1))*(x-sample(nos-1, 1))-(x-sample(1, 1))*(x-sample(2, 1)))*R%value&
        -(L%value + R%value)/2.0d0
    end if

    return
end function
```

4.3 分段立方样条插值

这里使用的边界条件是两端的一阶导为 0，因此需要解三转角方程，可以使用追赶法. 追赶法的程序见 Listing 4. 分段立方样条插值流程图见图3，代码见 Listing 5.

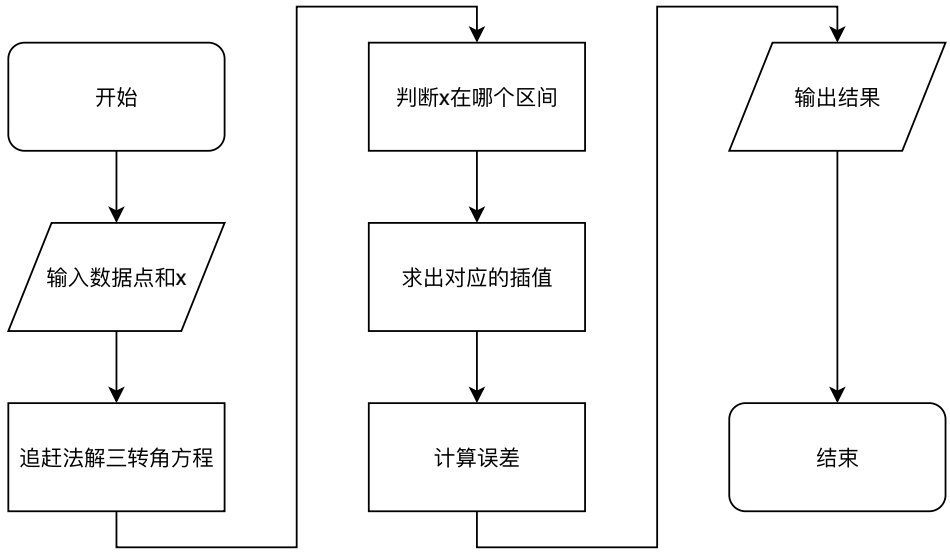

图 3: 分段立方样条插值流程图

Listing 4: **追赶法**

```
subroutine chasing_method(ndim, s, a, u, d, x)
    implicit none
    integer, intent(in) :: ndim
    real(8), dimension(ndim), intent(in) :: a, d
    real(8), dimension(ndim-1), intent(in) :: s, u
    real(8), dimension(ndim), intent(out) :: x

    real(8), dimension(ndim) :: alpha, y
    real(8), dimension(ndim-1) :: g, b
    integer :: i

    do i = 1, ndim-1
        b(i) = u(i)
    end do
    alpha(1) = a(1)
    do i = 2, ndim
        g(i-1) = s(i-1)/alpha(i-1)
        alpha(i) = a(i) - g(i-1)*b(i-1)
    end do

    y(1) = d(1)
    do i = 2, ndim
        y(i) = d(i) - g(i-1)*y(i-1)
    end do
    x(ndim) = y(ndim)/alpha(ndim)
    do i = ndim-1, 1, -1
        x(i) = (y(i)-b(i)*x(i+1))/alpha(i)
    end do

    return
end subroutine
```

Listing 5: **分段立方样条插值**

```
type(result) function cubic_spline(sample, x) result(y)
    implicit none
    real(8), dimension(*, *), intent(in) :: sample(:, :)
    real(8), intent(in) :: x

    integer :: nos
    real(8), dimension(size(sample, dim=1)) :: m
    real(8), dimension(size(sample, dim=1)-1) :: h
    real(8), dimension(size(sample, dim=1)-2) :: a, b, q, dia
    real(8), dimension(size(sample, dim=1)-3) :: u
```

```
    real(8) :: L, R
    integer :: i

    nos = size(sample, dim=1)

    do i = 1, nos-1
        h(i) = sample(i+1, 1) - sample(i, 1)
    end do
    do i = 1, nos-2
        a(i) = h(i) / (h(i) + h(i+1))
        b(i) = 3.0d0 * ((1.0d0-a(i))*(sample(i+1, 2)-sample(i, 2))/h(i) + a(i)*(sample(i+2, 2)
            -sample(i+1, 2))/h(i+1))
    end do

    m(1) = 0.0d0
    m(nos) = 0.0d0
    ! solve m by chasing method
    dia(:) = 2.0d0
    call chasing_method(nos-2, 1.0d0-a(2:nos-2), dia, a(1:nos-3), b(:), m(2:nos-1))

    do i = 1, nos-1
        if (x >= sample(i, 1) .and. x <= sample(i+1, 1)) then
            exit
        end if
    end do

    y%value = sample(i, 2) * (h(i)+2*(x-sample(i, 1))) * (x-sample(i+1, 1))**2 / h(i)**3
    y%value = y%value + sample(i+1, 2) * (h(i)-2*(x-sample(i+1, 1))) * (x-sample(i, 1))**2 /
        h(i)**3
    y%value = y%value + m(i) * (x-sample(i, 1)) * (x-sample(i+1, 1))**2 / h(i)**2
    y%value = y%value + m(i+1) * (x-sample(i+1, 1)) * (x-sample(i, 1))**2 / h(i)**2

    y%error = 1.0d0/(1.0d0+x**2) - y%value

    return
end function
```

5 插值结果

下面的结果是由问题给出的 16 个数据点插值，并在 $[-5,5]$ 的区间内选取 10000 个点绘制的图像，结果见图4，上面的三个图是插值曲线，下面的三个图是画出误差棒的插值曲线，由于点取的很密，所以误差棒看起来是连续的，这样可以反应整体的误差分布情况. 拉格朗日插值的结果

如图4左图所示. 可见出现了龙格现象，在两端的插值出现很大的误差，而在区间中心误差较小. 牛顿插值结果见图4中图所示，可见插值结果与拉格朗日插值完全一样. 分段立方样条插值结果如图4右图所示，可见两端的误差较小，中间的误差较大. 整体来讲，这三种方法中，分段立方样条插值的结果抖动更小，并且与原函数更接近. 但是就实际问题来讲，并没有理由说明抖动更小的结果就更接近真实值，因为真实值很多情况下是不可知的.

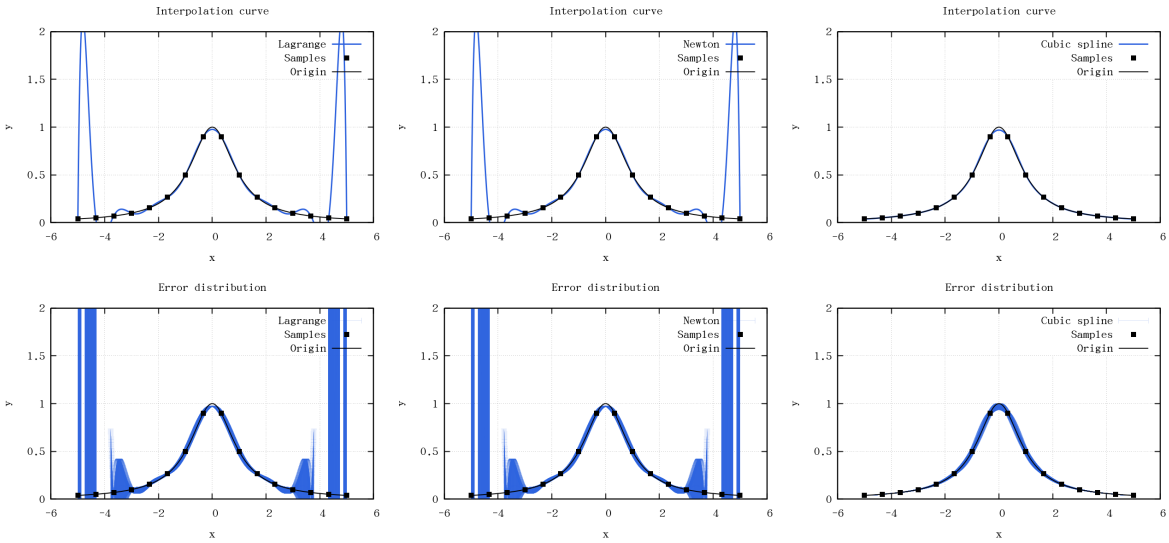

图 4: 三种方法的插值结果

附录

代码可在https://github.com/ZipWin/computational_physics/tree/master/assignments/assignment4找到.

Listing 6: **interpolation.f90**

```
module utils
    implicit none
    type result
        real(8) :: value, error
    end type

    contains
    recursive real(8) function diff_quot(points) result(dq)
        implicit none
        real(8), dimension(*, *), intent(in) :: points(:, :)

        integer :: ndim

        ndim = size(points, dim=1)

        if (ndim == 1) then
            dq = points(1, 2)
        else if (ndim >= 2) then
            dq = (diff_quot(points(2:ndim, :)) - diff_quot(points(1:ndim-1, :))) / (points(
                ndim, 1) - points(1, 1))
        else
            stop "ERROR: the size of list for differential quotient must greater then 2"
        end if

        return
    end function

    subroutine chasing_method(ndim, s, a, u, d, x)
        implicit none
        integer, intent(in) :: ndim
        real(8), dimension(ndim), intent(in) :: a, d
        real(8), dimension(ndim-1), intent(in) :: s, u
        real(8), dimension(ndim), intent(out) :: x

        real(8), dimension(ndim) :: alpha, y
        real(8), dimension(ndim-1) :: g, b
        integer :: i

```

```
        do i = 1, ndim-1
            b(i) = u(i)
        end do
        alpha(1) = a(1)
        do i = 2, ndim
            g(i-1) = s(i-1)/alpha(i-1)
            alpha(i) = a(i) - g(i-1)*b(i-1)
        end do

        y(1) = d(1)
        do i = 2, ndim
            y(i) = d(i) - g(i-1)*y(i-1)
        end do
        x(ndim) = y(ndim)/alpha(ndim)
        do i = ndim-1, 1, -1
            x(i) = (y(i)-b(i)*x(i+1))/alpha(i)
        end do

        return
    end subroutine

end module


module interpolation
    use utils
    contains
    recursive type(result) function lagrange(sample, x, require_error) result(y)
        implicit none
        real(8), dimension(*, *), intent(in) :: sample(:, :)
        real(8), intent(in) :: x
        logical, optional :: require_error

        real(8) :: item
        type(result) :: L, R
        integer :: i, j, nos

        if (.not. present(require_error)) then
            require_error = .true.
        end if

        nos = size(sample, dim=1)

        y%value = 0.0d0
```

```
82          do i = 1, nos
83              item = 1.0d0
84              do j = 1, i-1
85                  item = item * (x - sample(j, 1)) / (sample(i, 1) - sample(j, 1))
86              end do
87              do j = i+1, nos
88                  item = item * (x - sample(j, 1)) / (sample(i, 1) - sample(j, 1))
89              end do
90              y%value = y%value + item * sample(i, 2)
91          end do
92          if (require_error) then
93              L = lagrange(sample(1:nos-2, :), x, require_error=.false.)
94              R = lagrange(sample(3:nos , :), x, require_error=.false.)
95              y%error = (x-sample(nos, 1))*(x-sample(nos-1, 1))/&
96              ((x-sample(nos, 1))*(x-sample(nos-1, 1))-(x-sample(1, 1))*(x-sample(2, 1)))*L%
                    value&
97               - (x-sample(1, 1))*(x-sample(2, 1))/&
98              ((x-sample(nos, 1))*(x-sample(nos-1, 1))-(x-sample(1, 1))*(x-sample(2, 1)))*R%
                    value&
99              -(L%value + R%value)/2.0d0
100         end if
101
102         return
103     end function
104
105     recursive type(result) function newton(sample, x, require_error) result(y)
106         implicit none
107         real(8), dimension(*, *), intent(in) :: sample(:, :)
108         real(8), intent(in) :: x
109         logical, optional :: require_error
110
111         real(8) :: item
112         type(result) :: L, R
113         integer :: i, j, nos
114
115         if (.not. present(require_error)) then
116             require_error = .true.
117         end if
118
119         nos = size(sample, dim=1)
120
121         y%value = 0.0d0
122         do i = 1, nos
123             item = 1.0d0
```

```
124             do j = 1, i-1
125                 item = item * (x - sample(j, 1))
126             end do
127             y%value = y%value + item * diff_quot(sample(1:i, :))
128         end do
129         if (require_error) then
130             L = newton(sample(1:nos-2, :), x, require_error=.false.)
131             R = newton(sample(3:nos , :), x, require_error=.false.)
132             y%error = (x-sample(nos, 1))*(x-sample(nos-1, 1))/&
133             ((x-sample(nos, 1))*(x-sample(nos-1, 1))-(x-sample(1, 1))*(x-sample(2, 1)))*L%
                    value&
134              - (x-sample(1, 1))*(x-sample(2, 1))/&
135             ((x-sample(nos, 1))*(x-sample(nos-1, 1))-(x-sample(1, 1))*(x-sample(2, 1)))*R%
                    value&
136             -(L%value + R%value)/2.0d0
137         end if
138
139         return
140     end function
141
142     type(result) function cubic_spline(sample, x) result(y)
143         implicit none
144         real(8), dimension(*, *), intent(in) :: sample(:, :)
145         real(8), intent(in) :: x
146
147         integer :: nos
148         real(8), dimension(size(sample, dim=1)) :: m
149         real(8), dimension(size(sample, dim=1)-1) :: h
150         real(8), dimension(size(sample, dim=1)-2) :: a, b, q, dia
151         real(8), dimension(size(sample, dim=1)-3) :: u
152         real(8) :: L, R
153         integer :: i
154
155         nos = size(sample, dim=1)
156
157         do i = 1, nos-1
158             h(i) = sample(i+1, 1) - sample(i, 1)
159         end do
160         do i = 1, nos-2
161             a(i) = h(i) / (h(i) + h(i+1))
162             b(i) = 3.0d0 * ((1.0d0-a(i))*(sample(i+1, 2)-sample(i, 2))/h(i) + a(i)*(sample(i
                    +2, 2)-sample(i+1, 2))/h(i+1))
163         end do
164
```

```
165         m(1) = 0.0d0
166         m(nos) = 0.0d0
167         ! solve m by chasing method
168         dia(:) = 2.0d0
169         call chasing_method(nos-2, 1.0d0-a(2:nos-2), dia, a(1:nos-3), b(:), m(2:nos-1))
170
171         do i = 1, nos-1
172             if (x >= sample(i, 1) .and. x <= sample(i+1, 1)) then
173                 exit
174             end if
175         end do
176
177         y%value = sample(i, 2) * (h(i)+2*(x-sample(i, 1))) * (x-sample(i+1, 1))**2 / h(i)**3
178         y%value = y%value + sample(i+1, 2) * (h(i)-2*(x-sample(i+1, 1))) * (x-sample(i, 1))**2
                / h(i)**3
179         y%value = y%value + m(i) * (x-sample(i, 1)) * (x-sample(i+1, 1))**2 / h(i)**2
180         y%value = y%value + m(i+1) * (x-sample(i+1, 1)) * (x-sample(i, 1))**2 / h(i)**2
181
182         y%error = 1.0d0/(1.0d0+x**2) - y%value
183
184         return
185     end function
186
187 end module
188
189
190 program main
191     use utils
192     use interpolation
193     implicit none
194     integer, parameter :: N = 15, NS = 9999
195     real(8), dimension(N+1, 2) :: sample
196     real(8) :: x
197     type(result) :: y
198     integer :: i
199
200     open(file='samples.dat', unit=10, action='write')
201     do i = 0, N
202         sample(i+1, 1) = -5.0d0 + 10.0d0/N * i
203         sample(i+1, 2) = 1.0d0 / (1.0d0 + sample(i+1, 1)**2)
204         write (10, "(2f16.8)") sample(i+1, 1), sample(i+1, 2)
205     end do
206     close(unit=10)
207
```

```
208     open(file='lagrange.dat', unit=10, action='write')
209     do i = 0, NS
210         x = -5.0d0 + 10.0d0/NS * i
211         y = lagrange(sample, x, require_error=.true.)
212         write (10, "(4f16.8)") x, 1.0d0/(1.0d0+x**2), y%value, y%error
213     end do
214     close(unit=10)
215
216     open(file='newton.dat', unit=10, action='write')
217     do i = 0, NS
218         x = -5.0d0 + 10.0d0/NS * i
219         y = newton(sample, x, require_error=.true.)
220         write (10, "(4f16.8)") x, 1.0d0/(1.0d0+x**2), y%value, y%error
221     end do
222     close(unit=10)
223
224     open(file='cubic_spline.dat', unit=10, action='write')
225     do i = 0, NS
226         x = -5.0d0 + 10.0d0/NS * i
227         y = cubic_spline(sample, x)
228         write (10, "(4f16.8)") x, 1.0d0/(1.0d0+x**2), y%value, y%error
229     end do
230     close(unit=10)
231
232 end program
```