

Matlab数值计算

Author : 郭少华

ID Numer:2021050264

Class: 应数2101

Date : 2023/7/20

Matlab数值计算

第二章 插值法

2.1 Lagrange Interpolation

2.1.1 数学基础

2.1.2 算法详解

2.2 Newton Interpolation

2.2.1 数学基础

2.2.2 算法分析

第四章 数值微积分

4.1 数值积分

4.1.1 梯形公式

4.1.2 辛普森公式

4.1.3 复合梯形公式

4.1.4 复合辛普森公式

第七章 非线性方程组的数值解法

7.1 牛顿法

7.1.1 数学基础

7.1.2 算法分析

7.2 不动点迭代

7.1.1 数学基础

7.1.2 算法分析

第二章 插值法

2.1 Lagrange Interpolation

2.1.1 数学基础

给出 $(x_i, y_i) (i = 0, 1, \dots, n)$, $n + 1$ 个节点, 可以构造出次数不超过 n 的多项式 $P_n(x)$

如两个节点 $(x_0, y_0), (x_1, y_1)$ 构造线性插值多项式

$$P_1(x) = y_0 \frac{x-x_1}{x_0-x_1} + y_1 \frac{x-x_0}{x_1-x_0}$$

为一次多项式.

2.1.2 算法详解

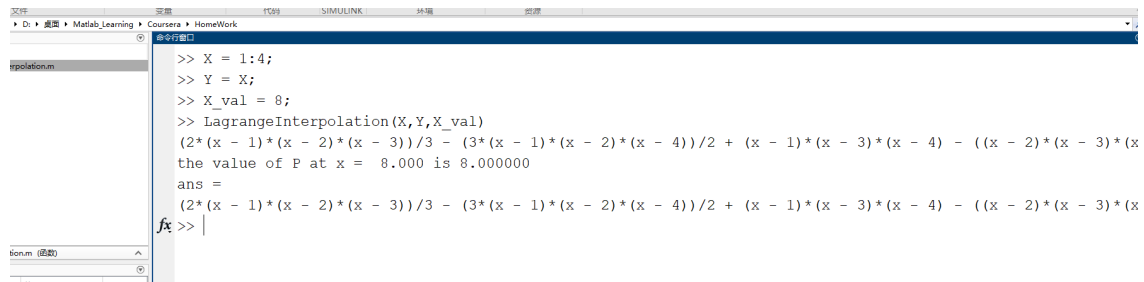
这个函数接收3个参数，第一个参数是给定节点的 x 坐标向量，第二个参数是给定节点的 y 坐标向量，第三个参数是希望根据插值多项式求出其他离散点的坐标点

函数返回两个结果，一个结果是插值多项式的表达式，注意，我们在显示表达式时，保留了拉格朗日插值多项式的原始形态，并未进行进一步的整理。另一个结果是需要的其他离散的函数值。

```
1  % LagrangeInterpolation 函数用于计算拉格朗日插值多项式及其在给定点的值。
2  % 输入参数：
3  % X - 插值点的x坐标，一个向量；特别注意，X的元素不可以相同！
4  % Y - 插值点的y坐标，一个向量
5  % x_val - 需要计算插值结果的x坐标点
6  % 输出参数：
7  % P - 拉格朗日插值多项式，一个符号表达式
8  % P_val - 插值多项式在x_val处的值
9  function [P, P_val] = LagrangeInterpolation(X, Y, x_val)
10
11  % 获取插值点的数量
12  n = length(X);
13
14  % 定义一个符号变量x
15  syms x;
16
17  % 初始化拉格朗日插值多项式为0
18  P = 0;
19
20  % 创建一个随机数向量作为系数的初始值
21  coff = rand(1,n);
22
23  % 对于每一个插值点
24  for i = 1:n
25      % 计算拉格朗日插值多项式的系数
26      coff(i) = Y(i) * 1/prod(X(i)-X([1:i-1,i+1:end]));
27
28      % 计算每个拉格朗日基函数，即x减去其他所有插值点x坐标的连乘积
29      product(i) = prod( x - X([1:i-1,i+1:end]));
30
31      % 计算每个拉格朗日基函数与其对应系数的乘积
32      term(i) = coff(i)*product(i);
33
34      % 将所有项加起来得到拉格朗日插值多项式
35      P = P + term(i);
36  end
37
38  % 显示拉格朗日插值多项式
39  disp(P);
40
41  % 计算并返回插值多项式在给定点x_val处的值
42  P_val = subs(P,x,x_val);
43
44  % 打印出插值结果
45  fprintf('the value of P at x = %.3f is %f\n',x_val,P_val);
46
47  end
```

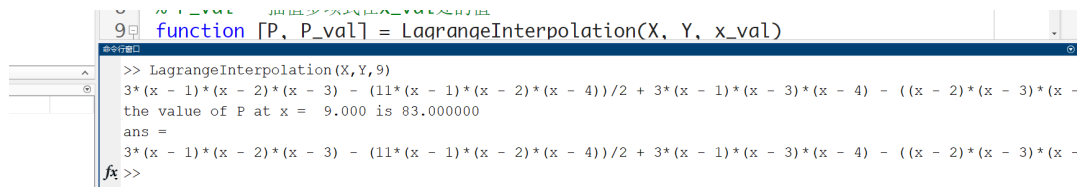
试验结果

1. $X = [1, 2, 3, 4], Y = X, X_{val} = 8$



```
>> X = 1:4;
>> Y = X;
>> X_val = 8;
>> LagrangeInterpolation(X,Y,X_val)
(2*(x - 1)*(x - 2)*(x - 3))/3 - (3*(x - 1)*(x - 2)*(x - 4))/2 + (x - 1)*(x - 3)*(x - 4) - ((x - 2)*(x - 3)*(x - 4))
the value of P at x = 8.000 is 8.000000
ans =
(2*(x - 1)*(x - 2)*(x - 3))/3 - (3*(x - 1)*(x - 2)*(x - 4))/2 + (x - 1)*(x - 3)*(x - 4) - ((x - 2)*(x - 3)*(x - 4))
fx >>
```

2. $X = [1, 2, 3, 4], Y = X^2 + 2, X_{val} = 9$



```
function [P, P_val] = LagrangeInterpolation(X, Y, x_val)
>> LagrangeInterpolation(X,Y,9)
3*(x - 1)*(x - 2)*(x - 3) - (11*(x - 1)*(x - 2)*(x - 4))/2 + 3*(x - 1)*(x - 3)*(x - 4) - ((x - 2)*(x - 3)*(x - 4))
the value of P at x = 9.000 is 83.000000
ans =
3*(x - 1)*(x - 2)*(x - 3) - (11*(x - 1)*(x - 2)*(x - 4))/2 + 3*(x - 1)*(x - 3)*(x - 4) - ((x - 2)*(x - 3)*(x - 4))
fx >>
```

可以看到，试验结果良好。

当然，我们可以对函数稍作修改，一次性计算多个离散点的函数值

```
1 % LagrangeInterpolation 函数用于计算拉格朗日插值多项式及其在给定点的值。
2 % 输入参数：
3 % x - 插值点的x坐标，一个向量
4 % Y - 插值点的y坐标，一个向量
5 % x_val - 需要计算插值结果的x坐标点，一个向量
6 % 输出参数：
7 % P - 拉格朗日插值多项式，一个符号表达式
8 % P_val - 插值多项式在x_val处的值，一个向量
9 function [P, P_val] = LagrangeInterpolation(X, Y, x_val)
10
11 % 获取插值点的数量
12 n = length(X);
13
14 % 定义一个符号变量x
15 syms x;
16
17 % 初始化拉格朗日插值多项式为0
18 P = 0;
19
20 % 创建一个随机数向量作为系数的初始值
21 coff = rand(1,n);
22
23 % 对于每一个插值点
24 for i = 1:n
25     % 计算拉格朗日插值多项式的系数
26     coff(i) = Y(i) * 1/prod(X(i)-X([1:i-1,i+1:end])));
27
28     % 计算每个拉格朗日基函数，即x减去其他所有插值点x坐标的连乘积
29     product(i) = prod( x - X([1:i-1,i+1:end])));
30
31     % 计算每个拉格朗日基函数与其对应系数的乘积
32     term(i) = coff(i)*product(i);
33
34     % 将所有项加起来得到拉格朗日插值多项式
```

```

35     P = P + term(i);
36 end
37
38 % 显示拉格朗日插值多项式
39 disp(P);
40
41 % 计算并返回插值多项式在给定点x_val处的值，x_val可以是一个向量
42 P_val = subs(P,x,x_val);
43
44 % 打印出插值结果，需要处理x_val为向量的情况
45 for i = 1:length(x_val)
46     fprintf('the value of P at x = %.3f is %f\n',x_val(i),P_val(i));
47 end
48
49 end

```

用 $X = [1, 2, 3, 4]$, $Y = X^2 + 3X$, $X_{val} = [5, 6, 7, 8]$ 试验

```

>> LagrangeInterpolation(X,Y,[5,6,7,8])
(14*(x - 1)*(x - 2)*(x - 3))/3 - 9*(x - 1)*(x - 2)*(x - 4) + 5*(x - 1)*(x - 3)*(x - 4) - (2*(x - 2)*(x - 3)*(x - 4))
the value of P at x = 5.000 is 40.000000
the value of P at x = 6.000 is 54.000000
the value of P at x = 7.000 is 70.000000
the value of P at x = 8.000 is 88.000000
ans =

```

可以看到，效果很理想。

2.2 Newton Interpolation

2.2.1 数学基础

牛顿插值多项式的公式为

$P_n(x) = a_0 + a_1(x - x_0) + a_2(x - x_0)(x - x_1) + \dots + a_n(x - x_0)(x - x_1)\dots(x - x_{n-1})$, a_i 为 $f(x)$ 关于前 $i + 1$ 个节点 $(x_0, y_0), (x_1, y_1), \dots, (x_i, y_i)$ 的差商

2.2.2 算法分析

```

1 function [P, P_val] = NewtonInterpolation(X, Y, x_val)
2     % 存储节点个数
3     n = length(X);
4     % 定义符号变量
5     syms x;
6
7     % 计算差商表
8
9     %初始化差商表为n阶矩阵
10    diff_table = zeros(n, n);
11    % 差商表第一列为节点的Y值，存储为列向量
12    diff_table(:,1) = Y';
13
14    % 从第2列开始，差商表每一列的数值由前一列的数值作差比上相应的x数值做作差
15    % 从第2列开始计算，到第n列结束
16    for j = 2:n
17        % 在第j列，分别计算第j行到第n行的差
18        for i = j:n

```

```

19         diff_table(i,j) = (diff_table(i,j-1) - diff_table(i-1,j-1)) /
    (x(i) - x((i-1) - (j-2))); % 特别注意，这里的x((i-1) + (j-2)) 是这么找到的：在第j-
    1列，向左边移动 j-2列到达第1列，每移动1列，对应的x的索引就要减少1，于是x的索引就减少了 j-
    2，对应的x为x((i-1) - (j-2)) = x(i-j+1)
20     end
21 end
22
23 % 构造牛顿插值多项式
24 % a0 等于差商表的第一个元素，用它初始化插值多项式
25 P = diff_table(1,1);
26 for i = 2:n
27     %取差商表的对角元为系数
28     term = diff_table(i,i);
29     for j = 1:i-1
30         term = term * (x - x(j));
31     end
32     P = P + term;
33 end
34
35 % 计算插值多项式在给定的x_val处的值
36 P_val = subs(P, x, x_val);
37 end

```

这个函数的构造采取了和拉格朗日插值函数同样的思路，接收三个参数：节点的X向量和Y向量，需要求值的离散点的横坐标，使用方法和拉个朗日函数相同

试验结果：

X = [1,2,3,4];Y = X; x_val = 9;

```

>> NewtonInterpolation([1,2,3,4],[1,2,3,4],9)
the NewtonInterpolation polynomial is:f(x) = x

the value of P at x = 9.000 is 9.000000

```

第四章 数值微积分

4.1 数值积分

以下讨论的近似公式，均为 $\int_a^b f(x)dx$ 的近似，约定步长 $h = \frac{1}{n}(b - a)$

4.1.1 梯形公式

数学表达式： $\int_a^b f(x) \approx \frac{1}{2}(f(a) + f(b))$

我们用 $f(x) = \sin(x)$ 试验；

```

1 % 定义梯形公式的函数
2 function I = trapezoidal_rule(f, a, b)
3     % f: 被积函数
4     % a: 积分下限
5     % b: 积分上限
6
7     h = b - a;
8     I = h / 2 * (f(a) + f(b));
9 end

```

试验:

```

1 f = @(x) sin(x);
2 I = trapezoidal_rule(f,1,10);

```

结果:

The image shows a MATLAB interface with three main components: a script editor, a workspace window, and a command window.

- Script Editor:** Displays the `trapezoidal_rule` function code. Lines 8 and 9 are highlighted, showing the calculation of `I` and the `end` statement.


```

8     I = h / 2 * (f(a) + f(b));
9 end

```
- Workspace Window:** Shows the variables `f` and `I` in the workspace.

| 名称 | 值 |
|----|------------|
| f | @(x)sin(x) |
| I | 1.3385 |
- Command Window:** Shows the execution of the script. It includes the command to edit the function, the function call, and the resulting value of `I`.


```

>> edit trapezoidal_rule
>> f = @(x) sin(x);
I = trapezoidal_rule(f,1,10);
>> I

I =

    1.3385

fx >>

```

我们再用牛顿莱布尼兹公式计算一遍:

```

1 True_value = cos(1) - cos (10)

```

计算结果为1.3794, 误差为0.0408

```

1.3385

>> True_value = cos(1) - cos(10)

True_value =

1.3794

>> e = True_value - I

e =

0.0408

```

我们在用matlab提供的积分函数计算一遍

```
1 quad(f,1,10)
```

结果也是1.3794

4.1.2 辛普森公式

```

1 function I = simpsons_rule(f, a, b)
2     % f: 被积函数
3     % a: 积分下限
4     % b: 积分上限
5
6     h = (b - a) / 2;
7     I = h / 3 * (f(a) + 4*f(a + h) + f(b));
8 end

```

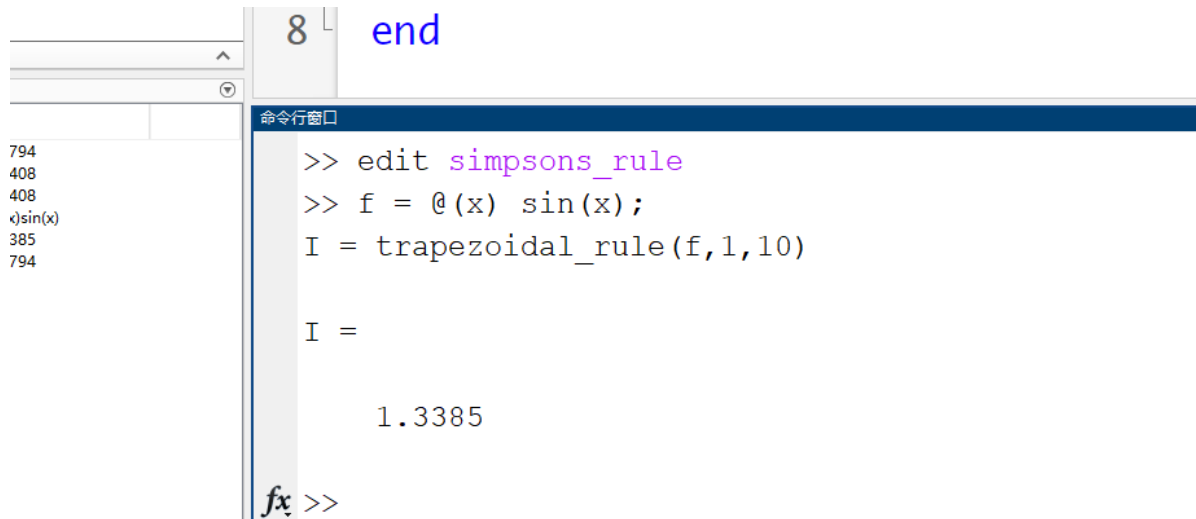
试验

```

1 f = @(x) sin(x);
2 I = trapezoidal_rule(f,1,10);

```

结果:



```
8 end

>> edit simpsons_rule
>> f = @(x) sin(x);
I = trapezoidal_rule(f,1,10)

I =

    1.3385

fx >>
```

误差:

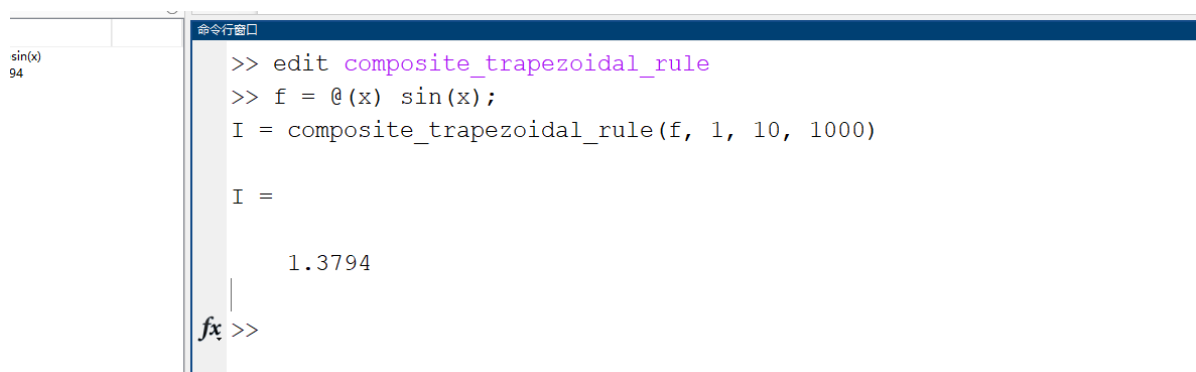
$$e = 1.3395 - 1.3794 = -0.0399$$

4.1.3 复合梯形公式

```
1 function I = composite_trapezoidal_rule(f, a, b, n)
2     % f: 被积函数
3     % a: 积分下限
4     % b: 积分上限
5     % n: 子区间个数
6
7     h = (b - a) / n;
8     x = a:h:b;
9     fx = f(x);
10    I = h / 2 * (fx(1) + 2*sum(fx(2:end-1)) + fx(end));
11 end
```

试验

```
1 f = @(x) sin(x);
2 I = composite_trapezoidal_rule(f, 1, 10, 1000);
```



```
>> edit composite_trapezoidal_rule
>> f = @(x) sin(x);
I = composite_trapezoidal_rule(f, 1, 10, 1000)

I =

    1.3794

fx >>
```

结果:

agrangleInterpolation.m
omposite_trapezoidal_rule.m

| 值 |
|-------------|
| 1.3794 |
| -9.3108e-06 |
| @(x)sin(x) |
| 1.3794 |

命令窗口

```
是不是想输入:  
>> format long  
>> I = composite_trapezoidal_rule(f, 1, 10, 1000)  
  
I =  
  
1.379364524158637  
  
>> cos(1) - cos(10)  
  
ans =  
  
1.379373834944592  
  
>> e = I - ans  
  
e =  
  
-9.310785955785050e-06
```

4.1.4 复合辛普森公式

```
1 function I = composite_simpsons_rule(f, a, b, n)  
2     % f: 被积函数  
3     % a: 积分下限  
4     % b: 积分上限  
5     % n: 子区间个数  
6  
7     h = (b - a) / (2*n);  
8     x = a:h:b;  
9     fx = f(x);  
10    I = h / 3 * (fx(1) + 2*sum(fx(3:2:end-2)) + 4*sum(fx(2:2:end)) +  
    fx(end));  
11 end
```

试验

```
1 f = @(x) sin(x);  
2 I = composite_simpsons_rule(f, 1, 10, 1000);
```

```
10 I = h / 3 * (fx(1) + 2*sum(fx(3:2:end-2)) + 4*sum
11 end
命令窗口
>> f = @(x) sin(x);
>> I = composite_simpsons_rule(f, 1, 10, 1000);
>> I

I =

    1.379373834947735

>> e = I - (cos(1) - cos(10))

e =

    3.142375248899043e-12

fx >>
```

第七章 非线性方程组的数值解法

7.1 牛顿法

7.1.1 数学基础

我们可以用迭代公式找到非线性方程 $f(x) = 0$ 的近似解

$$x_k = x_{k-1} - \frac{f(x_k)}{f'(x_k)}$$

牛顿法对迭代初值的选取要求严格，较好的迭代初值可以使得迭代格式迅速收敛到近似解，如果迭代初值选取不当，牛顿法可能发散

7.1.2 算法分析

为了使得函数具有普遍性，我们仍然采取和上面一样的思路，构造函数接收三个参数，一个是非线性多项式 $f(x)$ ，一个是迭代初值 x_0 ，一个是精度 ϵ

其中， $f(x)$ 为符号函数

```
1 function root = newtonMethod(f, x0, epsilon)
2     syms x;
3     % 计算导数
4     df = diff(f);
5
6     % 将x0转换为符号变量
7     x0 = sym(x0);
8
9     % 初始化差值
10    delta = Inf;
11
12    while abs(delta) > epsilon
13        % 计算f(x0)和df(x0)
14        f_val = double(subs(f, x, x0));
15        df_val = double(subs(df, x, x0));
16
17        % 计算新的x值
18        x1 = x0 - f_val / df_val;
```

```

19
20     % 计算差值
21     delta = abs(x1 - x0);
22
23     % 更新x0
24     x0 = x1;
25 end
26
27 root = x0;
28 end

```

调用函数，并且用 $f(x) = \sin(x)$, $x_0 = 2$ 试验

```

1 syms x;
2 f = sin(x);
3 x0 = 2;
4 epsilon = 0.001;
5 root = newtonMethod(f, x0, epsilon);
6

```

我们查看结果，并且检查在迭代结果处的函数值

```

1 double(sin(root))

```

结果：



```

1  function root = newtonMethod(f, x0, epsilon)
命令窗口
>> syms x;
f = sin(x);
x0 = 2;
epsilon = 0.001;
root = newtonMethod(f, x0, epsilon);

disp(root);
28976077833147918747/9223372036854775808

>> sin(root)

ans =

sin(28976077833147918747/9223372036854775808)

>> double(sin(root))

ans =

-9.1011e-11

```

可以看到，迭代精度非常高，说明牛顿法是一种高效的寻找近似解的方式。

7.2 不动点迭代

7.1.1 数学基础

$$x_k = f(x_{k-1}) \quad k = 1, 2, 3, \dots$$

收敛性:

若 $f(x)$ 满足

1. $f(x) \in C(a, b)$

2. $\forall x \in [a, b], a \leq f(x) \leq b$

3. $\exists L < 1, \forall x, y \in [a, b], |f(x) - f(y)| \leq L|a - b|$

则对 $\forall x_0 \in [a, b]$, 由 x_0 得到的不动点迭代序列收敛

7.1.2 算法分析

这里我们采用和上面一样的思路构建函数, 就不写那么多注释啦!

```
1 function root = newtonMethod(f, x0, epsilon)
2     syms x;
3     % 将x0转换为符号变量
4     x0 = sym(x0);
5     % 初始化差值
6     delta = Inf;
7     i = 0;%统计迭代次数
8     while abs(delta) > epsilon
9         f_val = double(subs(f, x, x0));
10        %新的x值
11        x1 = f_val;
12        %计算差值
13        delta = abs(x1 - x0);
14        %更新x值
15        x0 = x1;
16    end
17    fprintf("the number of 迭代 is %d", i);
18    root = x0;
19 end
```

这次我们利用 $f(x) = \sin(x), x \in [-1, 1]$ 来测试

```
1 root = Fixedpoint(f, -1/2, 0.001);
2 root2 = Fixedpoint(f, -1/2, 0.0001);
```

试验结果:

```
!
命令窗口
>> root2 = Fixedpoint(f,2,0.0001)
the number of 迭代 is 418
root2 =

    0.0842

>> root1 = Fixedpoint(f,2,0.001)
the number of 迭代 is 88
root1 =

    0.1800

fr \
```

(C:\Users\LENOVO\AppData\Roaming\Typora\typora-user-images\image-20230721092541117.png)

可以看到，对精度要求越高，结果越精确