2022211316 刘璐 第十周作业

第一题、第二题

文本, 信件

描述已自动生成

第三题

文本, 信件

描述已自动生成

文本, 信件

描述已自动生成

C#代码实现三种迭代算法

using System.Runtime.CompilerServices;

using System.Xml.XPath;

public class Program

{

#region 底层方法

//此处修改变量个数

static int scope = 1000;

//此处修改迭代次数上限

static int iter = 100000;

//此处修改精度限

static double tol = 0.001;

public static double GetFuncNum(int scope, double[] x)

{

double f = 0;

for (int i = 0; i < scope; i++)

{

f += Math.Pow(-x[i] + x[i + 1] + x[i + 2], 2) + Math.Pow(x[i] - x[i + 1] + x[i + 2], 2) + Math.Pow(x[i] + x[i + 1] - x[i + 2], 2);

}

return f;

}

public static double[] GetFuncDen(int scope, double[] x)

{

double[] f = new double[scope + 2];

for (int i = 0; i < scope; i++)

{

f[i] += 6 \* x[i] - 2 \* x[i + 1] - 2 \* x[i + 2];

f[i + 1] += -2 \* x[i] + 6 \* x[i + 1] - 2 \* x[i + 2];

f[i + 2] += -2 \* x[i] - 2 \* x[i + 1] + 6 \* x[i + 2];

}

return f;

}

static double[] GetX(int scope)

{

double[] x = new double[scope + 2];

if (scope % 3 == 1)

{

int count = 0;

for (int i = 0; i < scope + 2; i++)

{

switch (count)

{

case 0:

x[i] = 1; break;

case 1:

x[i] = 2; break;

default:

x[i] = 3; break;

}

++count;

if (count == 3) count = 0;

}

return x;

}

else

{

Console.WriteLine("非法变量数");

return x;

}

}

//计算收敛限

static bool GetTol(int scope, double[] x)

{

double tol\_cur = 0;

double[] tol\_den = GetFuncDen(scope, x);

for (int i = 0; i < scope + 2; i++)

{

tol\_cur += Math.Pow(tol\_den[i], 2);

}

tol\_cur = Math.Sqrt(tol\_cur);

return tol\_cur < tol;

}

#endregion

public static void Main(string[] args)

{

int[] count = new int[3];

double[] x\_ini1 = GetX(scope);

double[] x11 = new double[scope + 2];

for (int l = 0; l < iter; l++)

{

if (l == 0)

{

x11 = CalGrads(scope, x\_ini1);

}

else

{

x11 = CalGrads(scope, x11);

}

if (GetTol(scope, x11))

{

count[0] = l; break;

}

}

double[] x\_ini2 = GetX(scope);

double[] x22 = new double[scope + 2];

for (int l = 0; l < iter; l++)

{

if (l == 0)

{

x22 = CalConjGrads(scope, x\_ini2, l);

}

else

{

x22 = CalConjGrads(scope, x22, l);

}

if (GetTol(scope, x22))

{

count[1] = l; break;

}

}

double[] x\_ini3 = GetX(scope);

double[] x33 = new double[scope + 2];

for (int l = 0; l < iter; l++)

{

if (l == 0)

{

x33 = CalGradsBB(scope, x\_ini3, l);

}

else

{

x33 = CalGradsBB(scope, x33, l);

}

if (GetTol(scope, x33))

{

count[2] = l; break;

}

}

}

#region 梯度下降法

//此函数用于迭代梯度下降法中的x

static double[] CalGrads(int scope, double[] x)

{

double[] s = new double[scope + 2];

s = GetFuncDen(scope, x);

double[] xm = new double[scope + 2];

double p = 0.2;

double q = 0.5;

for (int j = 0; j < 100; j++)

{

for (int i = 0; i < scope + 2; i++)

{

xm[i] = x[i] - q \* s[i];

}

if (GetFuncNum(scope, x) - GetFuncNum(scope, xm) > 0.000000001)

{

break;

}

else

{

q = q \* p;

}

}

return xm;

}

#endregion

#region 共轭梯度法

static double g0 = 0;

static double[] d0 = new double[scope + 2];

//此函数用于迭代共轭梯度法中的x,选用的是Fletcher-Reeves公式

static double[] CalConjGrads(int scope, double[] x, int l)

{

double[] xm = new double[scope + 2];

double[] g = new double[scope + 2];

double[] d = new double[scope + 2];

double[][] G = GetG(scope);

for (int i = 0; i < scope + 2; i++)

{

for (int j = 0; j < scope + 2; j++)

{

g[i] += G[i][j] \* x[j];

if (l == 0)

{

d[i] -= G[i][j] \* x[j];

}

}

}

double p = 0; double q = 0; double[] q1 = new double[scope + 2];

if (l != 0)

{

double b = 0;

for (int i = 0; i < scope + 2; i++)

{

p += g[i] \* g[i];

}

b = p / g0; g0 = p;

for (int i = 0; i < scope + 2; i++)

{

d[i] = -g[i] + b \* d0[i];

}

for (int i = 0; i < scope + 2; i++)

{

for (int j = 0; j < scope + 2; j++)

{

q1[i] += d[j] \* G[j][i];

}

q += d[i] \* q1[i];

}

p /= q;

for (int i = 0; i < scope + 2; i++)

{

d0[i] = d[i];

}

for (int i = 0; i < scope + 2; i++)

{

xm[i] = x[i] + p \* d[i];

}

}

else

{

for (int i = 0; i < scope + 2; i++)

{

p += g[i] \* g[i];

for (int j = 0; j < scope + 2; j++)

{

q1[i] += d[j] \* G[j][i];

}

q += d[i] \* q1[i];

}

g0 = p;

p /= q;

for (int i = 0; i < scope + 2; i++)

{

d0[i] = d[i];

}

for (int i = 0; i < scope + 2; i++)

{

xm[i] = x[i] + p \* d[i];

}

}

return xm;

}

static double[][] GetG(int scope)

{

double[][] G = new double[scope + 2][];

//构造矩阵G

for (int i = 0; i < scope + 2; i++)

{

G[i] = new double[scope + 2];

for (int j = 0; j < scope + 2; j++)

{

if (i == 0)

{

G[i][0] = 6; G[i][1] = -2; G[i][2] = -2;

}

else if (i == 1)

{

G[i][0] = -2; G[i][1] = 12; G[i][2] = -4; G[i][2] = -2;

}

else if (i == scope)

{

G[i][scope - 2] = -2; G[i][scope - 1] = -4; G[i][scope] = 12; G[i][scope + 1] = -2;

}

else if (i == scope + 1)

{

G[i][scope - 1] = -2; G[i][scope] = -2; G[i][scope + 1] = 6;

}

else

{

G[i][i] = 18; G[i][i - 1] = -4; G[i][i - 2] = -2; G[i][i + 1] = -4; G[i][i + 2] = -2;

}

}

}

return G;

}

#endregion

#region 梯度BB法

static double[] df1 = new double[scope + 2];

static double[] x1 = new double[scope + 2];

//此函数用于迭代梯度BB法中的x,选取的是第一类BB算子

static double[] CalGradsBB(int scope, double[] x, int l)

{

double[] xm = new double[scope + 2];

double[] df = new double[scope + 2];

if (l == 0)

{

df = GetFuncDen(scope, x);

xm = CalGrads(scope, x);

for (int i = 0; i < scope + 2; i++)

{

df1[i] = df[i];

}

df = GetFuncDen(scope, xm);

double p = 0; double q = 0;

for (int i = 0; i < scope + 2; i++)

{

p += (xm[i] - x[i]) \* (df[i] - df1[i]);

q += Math.Pow(df[i] - df1[i], 2);

}

p /= q;

for (int i = 0; i < scope + 2; i++)

{

x1[i] = xm[i];

xm[i] = xm[i] - p \* df[i];

}

for (int i = 0; i < scope + 2; i++)

{

df1[i] = df[i];

}

}

else

{

df = GetFuncDen(scope, x);

double p = 0; double q = 0;

for (int i = 0; i < scope + 2; i++)

{

p += (x[i] - x1[i]) \* (df[i] - df1[i]);

q += Math.Pow(df[i] - df1[i], 2);

}

p /= q;

for (int i = 0; i < scope + 2; i++)

{

x1[i] = x[i];

xm[i] = x[i] - p \* df[i];

}

for (int i = 0; i < scope + 2; i++)

{

df1[i] = df[i];

}

}

return xm;

}

#endregion

}

运行后，从k为0开始计数

当变量个数为3时，梯度法是7次，共轭梯度法是2次，BB法是5次；

当变量个数为12时，梯度法是42次，共轭梯度法是14次，BB法是13次；

当变量个数为102时，梯度法是43次，共轭梯度法是14次，BB法是13次。

实际上，梯度法的精度无论在什么规模下都是最劣的，而共轭梯度法要显著优于BB法，BB法的误差会出现波动，并非一直下降，会出现反弹，证明了BB法不单调，但出现下降情况时会快于共轭梯度法，这导致其在小规模情况中，不如共轭梯度法，但在大规模情况，能够追上共轭梯度法。

各方法的迭代过程如下图所示