回溯直线搜索求解无约束优化问题：参数α、β对收敛速度的影响

实验平台

以下工作均在MATLAB R2017b中实现，采用计算精度为30位。由于采用的精度较高，计算用时较长，所以程序的主要运行时间都用于数值计算上。因此，运行时间可以作为衡量算法性能的指标。

回溯直线搜索的原理

回溯直线搜索是在下降过程中确定下降步长t的搜索方法。从初始步长$t = 1$开始判断回溯停止条件：

$$f(x+td) < f(x) + \alpha \bigtriangledown^T f(x) * td$$

如果不满足，步长变为原来的β倍。重复判断回溯停止条件，至满足要求为止。
本题目的迭代求解采用梯度下降+回溯直线搜索，函数实现如下所示：

```
% One Step of Backtracking Line Search is defined in step.m
function xUpdated = step(x, a, b)
% Negetive Gradient Direction
d = -gf(x);
d2 = gf(x)' * gf(x); % -d_k^2, or gf(x)^T*d
% Decide the Step
t = vpa(1, 30);
while(f(x + t * d) > f(x) - a * t * d2)
        t = t * b;
end
% Output
xUpdated = x + t * d;
```

梯度+回溯求解模型建立

原始问题

$$\min_x f(x) = e^{x1+3x2-0.1} + e^{x1-3x2-0.1} + e^{-x1-0.1}, in\ which\ x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix}$$

目标函数分析

由已知可得，目标函数$f(x)$是凸函数，因其为几个指数函数的非负线性和，而根据仿射性质，这几个指数函数均为凸函数。
$f(x)$在$[-3, 0.5] * [-1, 1]$上的函数图像由如下的代码绘制，在附录1中展示：

```
% The Graph of Objective Function
[xx, yy] = meshgrid(-3: 0.1: 1, -1: 0.1: 1)
mesh(xx, yy, f(xx, yy))
title('The Graph of Objective Function')
x1=xlabel('x1')
x2=ylabel('x2')
x3=zlabel('f(x)')
```

如图所示，$x_2 = 0$是函数值的一个低谷。从计算角度来看，因为

$$\triangledown f(x) = \begin{bmatrix} e^{x1+3x2-0.1} + e^{x1-3x2-0.1} - e^{-x1-0.1} \\ 3e^{x1+3x2-0.1} - 3e^{x1-3x2-0.1} \end{bmatrix}$$

梯度的第二分量为0的点集中在$x_2 = 0$的直线上，说明函数总在$x_2 = 0$时取得最小值。通过分析计算，问题的最优解为

$$\begin{bmatrix} x_1^* = \frac{-ln2}{2} \\ x_2^* = 0 \end{bmatrix}$$

总体而言，函数的性态非常优良，可以直接应用回溯直线搜索求解，而没有什么需要额外注意的地方。
上文目标函数、梯度函数的实现程序如下所示：

```
% The Objective Function is defined in f.m
%  y = f(x), in which 'x' should be 2 * 1 vector.
%  y = f(x1, x2), in which 'x1', 'x2' should be numbers, as the components of 'x'
function y = f(x, x2)
if(nargin == 1)
        y = vpa(exp(x(1) + 3 * x(2) - 0.1) + exp(x(1)
            - 3 * x(2) - 0.1) + exp(-x(1) - 0.1), 30);
end
if(nargin == 2)
        y = vpa(exp(x + 3 * x2 - 0.1)
            + exp(x - 3 * x2 - 0.1) + exp(-x - 0.1), 30);
end
% The Gradient Function is defined in gf.m
%  y = gf(x), in which 'x' should be 2 * 1 vector.
%  y = gf(x1, x2), in which 'x1', 'x2' should be numbers, as the components of 'x'
function y = gf(x, x2)
if(nargin == 1)
        y = [vpa(exp(x(1) + 3 * x(2) - 0.1) + exp(x(1) - 3 * x(2) - 0.1)
             - exp(-x(1) - 0.1), 30);
             vpa(3 * exp(x(1) + 3 * x(2) - 0.1)
             - 3 * exp(x(1) - 3 * x(2) - 0.1), 30)];
end
if(nargin == 2)
```

```
        y = [vpa(exp(x + 3 * x2 - 0.1) + exp(x - 3 * x2 - 0.1)
             - exp(-x - 0.1), 30);
             vpa(3 * exp(x + 3 * x2 - 0.1)
             - 3 * exp(x - 3 * x2 - 0.1), 30)];
end
```

问题求解

求解准备

求解原问题所使用的回溯函数如下所示：

```
% The Function of Back Search is defined in backSearch.m
%  'xProcess' corresponds to the points during the process, an 2 * (count + 1) vector.
%  'xSolved' is the final result, an 2 * 1 vector.
%  'count' is the loop time.
function [xProcess, xSolved, count] = backSearch(x, a, b, e)
xSolved = vpa(x, 30);
xProcess = [xSolved];
count = 0;
while(sum(abs(gf(xSolved))) > e)
        xSolved = step(xSolved, a, b)
        count = count + 1;
        xProcess = [xProcess, xSolved];
end
```

由上所述，初值的选取对求解没有太大影响。由理论推导，回溯下降方法大致以线性收敛。因此，我们参考梯度误差，选取3组具有等比梯度误差的初值进行求解：

$\begin{bmatrix} x_1 = -0.47 \\ x_2 = 0.10 \end{bmatrix}$，梯度误差略大于1e0；

$\begin{bmatrix} x_1 = -0.40 \\ x_2 = 0.03 \end{bmatrix}$，梯度误差略大于1e-1；

$\begin{bmatrix} x_1 = -0.36 \\ x_2 = 0.01 \end{bmatrix}$，梯度误差略大于1e-2.

同时用程序生成待评估的参数列表，具体实现如下所示：

```
% Prepare Data
initList = [[-0.47; 0.10], [-0.40; 0.03], [-0.36; 0.01]];
eList = [1e-2, 1e-3, 1e-4];
aList = [0.05, 0.15, 0.25, 0.35, 0.45];
bList = [0.1, 0.3, 0.5, 0.7, 0.9];
colorList = ['r', 'g', 'b', 'y', 'k'];
```

收敛速度分析

对三组初值点在$\alpha = 0.25, \beta = 0.5$的情况下进行求解，求解程序如下所示：

```
% Available
countMatrix = zeros(3);
```

```
a = 0.25; b = 0.5;
for i = 1: 3
        for j = 1: 3
                [xProcess, xSolved, count]
                = backSearch(initList(:, i), a, b, eList(4 - j));
                plot(xProcess(1, :), xProcess(2, :), colorList(j))
                hold on
                countMatrix(i, j) = count
        end
end
```

以下为求解结果：

初值序号	迭代步数（1e-4）	迭代步数（1e-3）	迭代步数（1e-2）
1e0	18	14	9
1e-1	16	12	7
1e-2	15	10	5

下降路径图包含于附录2.如图显示，不同的初始值在他们对应量度的最初几步未进入主下降路径$x_2 = 0$，在x_2方向上下参差较大，所以下降量比较随机，但可以保证线性下降速度。随着迭代步数的增加，在进入主下降路径之后目标点的下降更具方向性，下降速度更快。

求解参数对结果的影响

以下使用初值点$\begin{bmatrix} x_1 = -0.47 \\ x_2 = 0.10 \end{bmatrix}$进行求解，梯度误差约为1e0.默认参数为$\alpha = 0.25, \beta = 0.5.$

α对结果的影响

求解程序如下：

```
% Alpha
countMatrix = zeros(5, 1);
timeMatrix = zeros(5, 1);
x = [-0.47; 0.10]; b = 0.5; e = 1e-2;
for j = 1: 5
        timej = cputime;
        [xProcess, xSolved, count] = backSearch(x, aList(j), b, e);
        plot(xProcess(1, :), xProcess(2, :), colorList(j))
        hold on
        countMatrix(j, 1) = count
        timeMatrix(j, 1) = cputime - timej
end
```

α	迭代步数	执行用时
0.05	9	25.3
0.15	9	22.6
0.25	9	22.1
0.35	9	23.1
0.45	9	28.7

α对迭代步数、迭代用时均无太大的影响，只是造成了迭代路线的些微区别，如附录3所示。实际上，α的理论作用是规定步长不能太小，同时也保证下降性：$\bigtriangledown^T f(x) * td$是负的，而$f(x+td) \rightarrow f(x) + \bigtriangledown^T f(x) * td, t \rightarrow 0$，相当于$\alpha = 1.0 < \alpha < 0.5$的设定让这个过程在中间结束，防止过度迭代。

β对结果的影响

求解程序如下：

```
% Beta
countMatrix = zeros(5, 1);
timeMatrix = zeros(5, 1);
x = [-0.47; 0.10]; a = 0.25; e = 1e-2;
for j = 1: 5
        timej = cputime;
        [xProcess, xSolved, count] = backSearch(x, a, bList(j), e);
        plot(xProcess(1, :), xProcess(2, :), colorList(j))
        hold on
        countMatrix(j, 1) = count
        timeMatrix(j, 1) = cputime - timej
end
```

求解结果如下：

β	迭代步数	执行用时
0.1	12	23.4
0.3	6	15.2
0.5	9	20.9
0.7	9	25.7
0.9	11	61.3

迭代路径如附录4所示。综合迭代步数和执行用时可以发现，β较小时求得的步长不够精确且数值太小，因此迭代步数较多，如图中的红线；β稍大时能获得精确的步长，算法迅速收敛，如图中绿线；β较大时，因为与α不配合，每一步迭代会产生对α的过拟合现象，产生的迭代路径点受α的误差影响较大，反而不利于

迭代，如图中黑线。对于内部迭代次数，β越大内部迭代次数越多，耗时也越长；这一点得到了实验结果的印证。