MATLAB: DELS

郭龙昕，江诗毅，胡进

2019 年 4 月 21 日

任务一

解决思路

1 计算 m 期生产满足到第 $n-1$ 期的成本。

2 计算从第 1 期开始到第 N 最低成本。

3 输出从第 1 期开始到第 N 的最优生产路径。

4 检查一个路线是否是最优路线

具体实现过程

- 计算 m 期生产满足到第 $n-1$ 期的成本。程序：mToNCost.m

```
function cost = mToNCost(d,k,c,h,m,n)
%
%输入:
%   d(vector) : 各阶段的需求
%   k(vector) : 各阶段的固定成本
%   c(vector) : 各阶段的单位边际成本
%   h(vector) : 各阶段的持有库存的边际成本
%   m(number) : 开始的阶段
%   n(number) : 结束的后一阶段
%
%输出:
%   cost(number) : 第m期生产满足第m到n−1期的所有需求带来的成本
for i = m:n−1
  cost = cost + c(m) .* d(i) + sum(h(m:i−1)) .* d(i);
end
end
```

- 计算从第 1 期开始到第 N 最低成本。

  程序:dySolution.m

```
%算法时间复杂度为O(n^2),n为维度
%输入:
%   d(vector) : 各阶段的需求
%   k(vector) : 各阶段的固定成本
%   c(vector) : 各阶段的单位边际成本
%   h(vector) : 各阶段的持有库存的边际成本
%
%输出:
%   result(number) : 最小成本
%   road(vector) : 达到最小成本的方案(0代表不生产, 1代表生产)
%
%example(d,k,c,h都为n维向量):
%   [optResult,road] = dySolution(d,k,c,h)
function [result,road] = dySolution(d,k,c,h)
    disp('旋疯□□ョ淮搴︾□□□□□□□');
    result = −1;
    road = −1;
    return
end
s = zeros(1,length(d));
for i = 1:length(d)
    r(i) = mToNCost(d,k,c,h,1,i+1);
    s(i) = 1;
```

```
    for j = 1:i-1
        temp = r(j) + mToNCost(d,k,c,h,j+1,i+1);
        if temp < r(i)
            r(i) = temp;
            s(i) = j+1;
        end
    end
end
result = r(length(d));
road = dyRoad(length(d),s);
end
```

步骤

1 基本的输入向量验证，例如验证维度是否符合要求。

2 预分配内存，提高效率: 一维数组 r 表示从第 1 期到第 n 期的最小成本，该问题是动态规划的子问题。一维数组 s 表示 r(n) 对应最优子方案中最后一次生产的时期是在第 s(n) 阶段。

3 迭代求解子问题，例如现在是第 i 次循环，依次遍历 s(i)= 1 to i，计算值，记录下最优路线方案，以及最有成本。

4 重复步骤 3，直到 r(n) 被算出，停止迭代。

- 输出从第 1 期开始到第 N 的最优生产路径 程序： *dyRoad.m*

```
function road = dyRoad(leng,s)
%dyRoad - □规□□儿 瑙□□缁□□□□锛□ 褰□□缁□□鏈 □□崇□□规
%
%璺 緞□□ 褰□□涔□細 1 ---> ... ---> s(s(n) - 1) --->  s(n) ---> N
%杈□□锛 %    leng(number):□舵 □扮□
%    s(vector):□ㄥ□鍋□ □□□娉□腑缁□□□□
%杈□□锛 %    road(vector) : 杈惧□鏈 □□□□□□□妗 0浠 h 〃涓□□浜 □1浠 h 〃□□骇)

road = zeros(1,leng);
point = leng;
while point > 0
    road(s(point)) = 1;
    point = s(point) - 1;
end
end
```

步骤

1 路径的递归定义：1 —> ... —> s(s(n) - 1) —> s(n) —> N

2 从 s(n) 开始，表示最后生产的阶段。

3 继续计算 s(s(n)-1)，表示倒数第二次的生产阶段。

4 重复下去，直到回到 s(i) - 1 < 0。

- 检查一个路线是否是最优路线

首先对为什么要编写程序检查一个路线是否是最优路线作出解释：由于对于每组 d,k,c,h 都有可能有多个最优解，所以我们有时候需要核查给定 road 是否是最优方案的方法。

程序： *checkOptRoad.m*

```matlab
%checkOptRoad - 判断给定路径是否是动态规划的最优解
%
%输入:
%    d(vector) : 各阶段的需求
%    k(vector) : 各阶段的固定成本
%    c(vector) : 各阶段的单位边际成本
%    h(vector) : 各阶段的持有库存的边际成本
%    road(vector) : 路线
%
%输出:
%    result(boolean) : 逻辑1或者逻辑0
% 根据路线给出成本
% 比较最优成本与sum_cost
function result = checkOptRoad(d,k,c,h,road)
plan = find(road == 1);
sum = 0;
for i = 1:length(plan)-1
    sum = sum + mToNCost(d,k,c,h,plan(i),plan(i+1));
end
sum = sum + mToNCost(d,k,c,h,plan(length(plan)),length(d)+1);
% disp(sum);
% disp(dySolution(d,k,c,h));
result = sum == dySolution(d,k,c,h);
end
```

测试主程序

测试程序伪代码

```matlab
% 随机生成d,k,c,h,以及c, h增加的同一个常数add
% 生成1000组测试用例
d,k,c,h,add = randi(10,1000,10);

for i = 1:1000
% 分别计算四种情况下的最优值
  [a1,b1] = dySolution(d(i,:),k(i,:),c(i,:),h(i,:));
  [a2,b2] = dySolution(d(i,:),k(i,:) + tem,c(i,:),h(i,:));
  [a3,b3] = dySolution(d(i,:),k(i,:),c(i,:) + tem,h(i,:));
  [a4,b4] = dySolution(d(i,:),k(i,:)+tem,c(i,:)+tem,h(i,:));
% 检验他们的最优决策方案是否相同
all([
  checkOptRoad(d(i,:),k(i,:),c(i,:) + tem,h(i,:),b1),
  checkOptRoad(d(i,:),k(i,:) + tem,c(i,:),h(i,:),b1),
  checkOptRoad(d(i,:),k(i,:) + tem,c(i,:)+tem,h(i,:),b1)
]) == 1
```

测试结果

测试用例	结果	要求
dySolution(1,1,1,1)	通过	结果为 2
dySolution(d,k,c,h)	通过	d,k 保持不变，c 或者 h 增加一个常数，最优方案不变
dySolution(d,k,c,h)	通过	k=0,c,h 保持不变，d 不影响最优决策

算法效率分析

对于该算法，在每次子问题的迭代过程中，都要计算 i-1 种方案进行对比，所以算法的复杂度是 $O(n^2)$。下面我们绘制出该算法解决问题所需时间随规模变化的曲线。(图 1, 图 2)

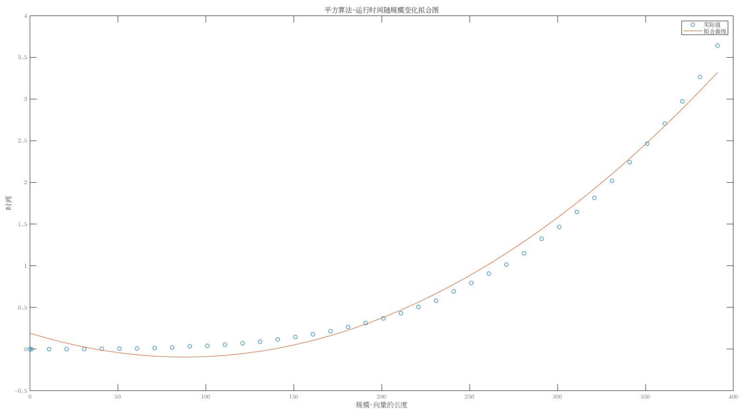

图 2: 密集点拟合

稀疏点拟合的参数分别为 $a = 0.0000 \cdot 10^{-4} \quad b = -0.0058 \cdot 10^{-4} \quad c = 0.1412 \cdot 10^{-4}$, 如图一：稀疏点拟合。
密集点拟合的参数分别为 $a = 0.0000 \cdot 10^{-4} \quad b = -0.0065 \cdot 10^{-4} \quad c = 0.1921 \cdot 10^{-4}$，如图二：密集点拟合。

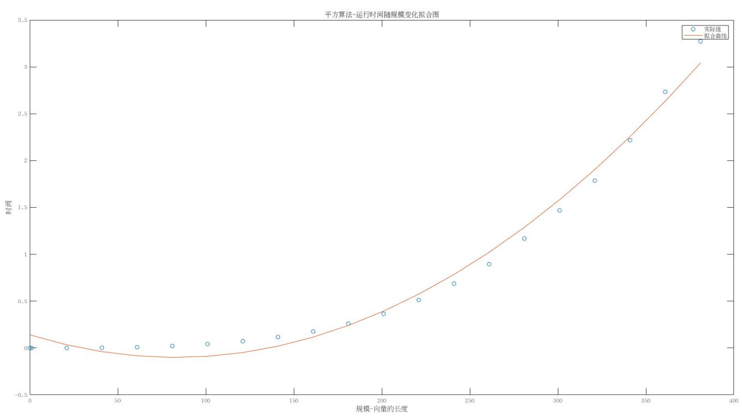

图 1: 稀疏点拟合

任务二

a 如果没有固定生产成本，那么最优生产序列与需求序列无关。

1 数值分析:

测试命题是否正确的伪代码

```
% 测试 k = 0时，最优决策与d无关。
d = randi(10,100,10);
k = zeros(1,10);
c = randi(10,1,10);
h = randi(10,1,10);
```

```
[opt,plan] = dySolution(d(1,:),k,c,h);
for i = 1:100
% 检验最优决策方案是否发生改变。
checkOptRoad(d(i,:),k,c,h,plan)
```

2 数学分析

假设在 m 期购买第 t 期所需的商品，总费用为：

$$c(m) = k_m + c_m d_t + (h_m + h_{m+1} + \cdots + h_{t-1}) d_t$$

所要做的决策是选择在某一期订购使得每件商品的平均成本最小。设所做出的最优决策为在第 p 期订购，则

$$p = \arg\min_m \{\frac{c(m)}{d_t}\} = \arg\min_m \{\frac{k_m}{d_t} + c_m + h_m + h_{m+1} + \cdots + h_{t-1}\}$$

当 $k_m = 0$ 时，最优决策为：

$$p = \arg\min_m \{c_m + h_m + h_{m+1} + \cdots + h_{t-1}\}$$

最优决策的表达式中不含 d_m，所以当 $k_m = 0$ 时，第 m 期的最优决策与 d_m 无关。

所以当 $k_n = 0, \forall n$ 时，最优生产计划与需求序列无关。

b 在计算第 m 期生产从第 m 期至第 n 期所需的所有商品的总成本的时，有两种思路，一种是分别计算从第 m 期至第 n-1 期间各期成本，然后再求和；另一种是除固定成本外，先分别计算满足各批次需求的成本，然后再求和。一般而言，用第 m 期的生产来满足第 $t \geq m$ 期需求的总成本可记录为 $c_{m.t} d_t$。请给出一般形式的 $c_{m,t}$ 表达式，然后再 Matlab 中实现，并用原来的代码测试。

$c_{m,t}$ 的表达式如下：

$$c_{m,t} = c_m + h_m + h_{m+1} + \cdots + h_{t-1}$$

使用这种思路写出的 totalcost 函数为：

```
function m_n_totalcost = totalcost_2(d,k,c,h,m,n)
%求出在第m期生产m-(n-1)期的所有产品的总花费
%输入:
%  d(vector) : 需求序列
%  k(vector) : 每一期生产的固定成本
%  c(vector) : 每一期生产单位商品的成本
%  h(vector) : 每一期期末单位商品的库存成本
%  m(number) : 开始的期数
%  n(number) : 结束期数的下一期。
%
%输出:
%  m_n_totalcost(number) : 第m期生产满足第m到n-1期的所有需求的总成本
m_n_totalcost = k(m) ;
for i = m:n-1
    m_n_totalcost =m_n_totalcost + c(m) .* d(i) + sum(h(m:i-1)) .* d(i);
end
end
```

下面对上述代码进行测试

测试思路：对于相同的 d,k,c，h 分别调用第一种思路下的 totalcost 函数，和第二种思路下的 totalcost_2 函数，比较得出的函数值是否相同，如果不同，那么函数会报错，并跳出循环，如果没有报错，重复 10000 次。

测试代码如下：

```
for i = 1 : 10000
d = randi([1,100],1,10) ;
c = randi([1,100],1,10) ;
k = randi([1,100],1,10) ;
h = randi([1,100],1,10) ;
a = totalcost(d,k,c,h,1,9);
b = totalcost_2(d,k,c,h,1,9);
if a ~= b
    disp('程序有误')
    break
end
end
```

运行结果：程序没有报错，说明上述程序 totalcost_2 是正确的。

c 如果把 $\ell(m.n)$ 的定义改成 $\ell(m,n) = k_m + c_{m,N+1}d_{1,n+1}$，是否会改变原问题的最优解。

在解决这个问题的时候，我们将老师给的式子代入程序运算后发现最优解会发生改变。

但是我们进一步猜测应该具有某个，甚至某一类表达式是的最优解不发生改变，我们尝试对老师给出的表达式进行变形，主要改变 c,d 的下标。我们试过的表达式有 $\ell(m,n) = k_m+c_{m,N+1}d_{1,n+1}, \ell(m,n) = 2k_m + c_{m,N+1}d_{1,n+1}, \ell(m,n) = k_m + c_{m,N+1}d_{m,n+1}, \ell(m,n) = k_m + c_{m,N+1}d_{m,n}, \ell(m,n) = k_m + c_{m,N+1}d_{m,n-1}$，最后发现当表达式变为 $\ell(m,n) = k_m + c_{m,N+1}d_{m,n-1} = k_m + (c_m + h_m + h_{m+1} + \cdots + h_N)(d_m + d_{m+1} + \cdots + d_{n-1})$ 最优解不发生改变。

下面我们证明：当表达式为 $\ell(m,n) = k_m+c_{m,N+1}d_{m,n-1} = k_m+(c_m+h_m+h_{m+1}+\cdots+h_N)(d_m+d_{m+1}+\cdots+d_{n-1})$ **最优解不发生改变**。

1 数值分析

我们将上述函数 totalcost 改为题目中的 $\ell(m,n) = k_m + c_{m,N+1}d_{m,n-1}$，得到一个新的函数，我们将其命名为 totalcost_3。

程序如下

```
function m_n_totalcost = totalcost_3(d,k,c,h,m,n)
%求出在第m期生产m-(n-1)期的所有产品的总花费
%输入:
%   d(vector) : 需求序列
%   k(vector) : 每一期生产的固定成本
%   c(vector) : 每一期生产单位商品的成本
%   h(vector) : 每一期期末单位商品的库存成本
%   m(number) : 开始的期数
%   n(number) : 结束期数的下一期。
%
%输出:
%   m_n_totalcost(number) : 第m期生产满足第m到n-1期的所有需求的总成本
m_n_totalcost = k(m) + (c(m)+sum(h(m:length(h))))  .* sum(d(m:n-2)) ;
end
```

当总成本为 totalcost_3 给出的成本时，我们测试最优解是否不变。

程序思路：

在总成本分别为 totalcost_3 和 totalcost 给出的成本时，我们分别调用 DELS 函数，如果最优解不同，那么程序报错并跳出循环；如果最优解相同，那么重复 10000 次。

程序:totalcost_3_text

```
for i = 1 :10000
    d = randi([1,100],1,10) ;
    k = randi([1,100],1,10) ;
    c = randi([1,100],1,10) ;
    h = randi([1,100],1,10) ;
    [a,b] = DELS(d,k,c,h) ;
    [c,d] = totalcost_3_DELS(d,k,c,h) ;
    if b~=d
        disp("命题错误")
        break
    end
end
```

运行结果：程序没有报错。

这就证明了在 $\ell(m,n)$ 换成新的表达式以后，最优解没有发生改变。

2 数学分析

在原来的 $\ell(m,n)$ 的定义下，

假设在 m 期购买第 n 期所需的商品，总费用为：

$$c(m,n) = k_m + c_m d_n + (h_m + h_{m+1} + \cdots + h_{n-1})d_n$$

所要做的决策是选择在某一期订购使得总成本最小。设所做出的最优决策为在第 p 期订购，则

$$p(n) = \arg\min_m \{c(m,n)\} = \arg\min_m \{k_m + (c_m + h_m + h_{m+1} + \cdots + h_{n-1})d_n\}$$

在新的 $\ell(m,n)$ 的定义下，在第 m 期生产第 m 期到第 n 期所需所有产品的总费用为：

$$\ell(m,n+1) = k_m + (c_m + h_m + \cdots + h_N)(d_m + d_{m+1} + \cdots + d_n)$$

其中 N 为问题所求的总得期数。

此时，在第 m 期生产第 n 期所需的产品的总费用为：

$$c(m,n) = k_m + (c_m + h_m + \cdots + h_N)d_n$$

所做出的决策为选择第 m 期生产第 n 期所需的产品，使得总成本最小，即

$$\begin{aligned}
p(n) &= \arg\min_m c(m,n) \\
&= \arg\min_m \{k_m + (c_m + h_m + \cdots + h_N)d_n\} \\
&= \arg\min_m \{k_t + (c_m + h_m + h_{m+1} + \cdots + h_{n-1})d_n + (h_n + h_{n+1} + \cdots + h_N)d_n\} \quad (1) \\
&= \arg\min_m \{k_t + (c_m + h_m + h_{m+1} + \cdots + h_{n-1})d_n\}
\end{aligned}$$

(1) 式中有两项构成，一项为 $k_t + (c_m + h_m + h_{m+1} + \cdots + h_{n-1})d_n$ 这正是在原来的 $\ell(m, n)$ 定义下在 m 期订购第 n 期所需商品的总费用, 而另一项为 $(h_n + h_{n+1} + \cdots + h_N)d_n$ 这是一个与 m 无关的常数。

所以在以前的 $\ell(m, n)$ 的定义下，如果在 $p(n)$ 期订购第 n 期所需商品使得总成本最小，那么在新的 $\ell(m, n)$ 的定义下，同样在 $p(n)$ 期订购第 n 期所需商品使得总成本最小。

由于 n 的任意性，该结论对 $1 - N$ 期都成立。

即，在改变 $\ell(m, n)$ 的定义下，最优解不会改变。

任务三

根据题目提示，如果已经得到了第 n 期之后的最优生产期分别为 $n = s_o < s_1 < s_2 < \cdots < s_m \leq N$，那么在计算第 n-1 期之后的最优生产计划时，$s_0, s_1$ 可能会更新，但是 s_2 及以后的计划期不会改变。

我们做出猜想：如果已经得到了第 m 期之前的最优生产期分别为 $n = s_o < s_1 < s_2 < \cdots < s_n \leq m$，那么在计算第 m+1 期之后的最优生产计划时，$s_m, s_{m-1}$ 可能会更新，但是 s_{n-2} 及之前的计划期不会改变。也就是说最多会改变最近两期的决策。

算例验证如下图所示.

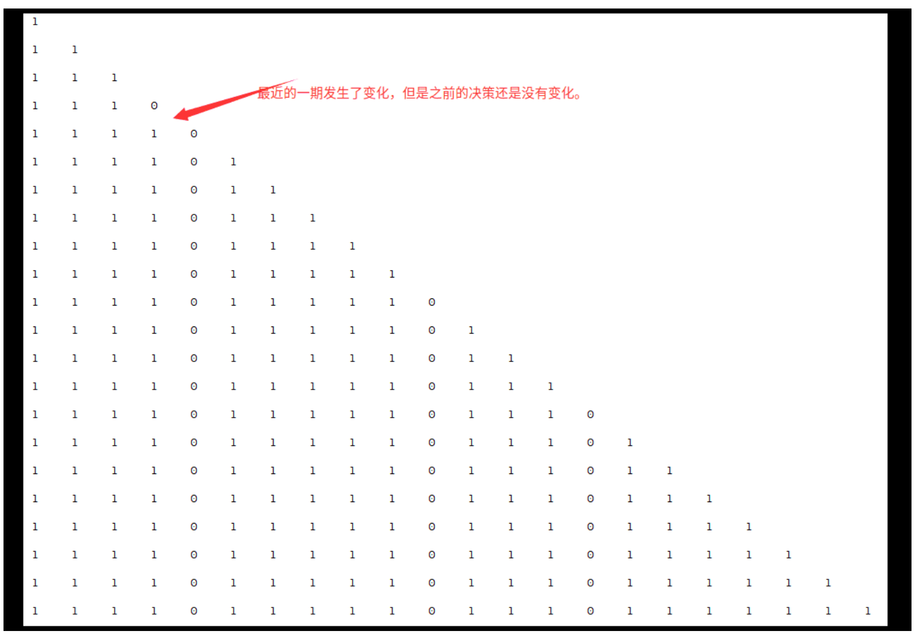

图 3: 稀疏点拟合

_197.png _197.bb _197.pngGraphic file (type bmp)

图 4: 密集点拟合

从图中可以看出最多只有最近两期的决策会发生改变。

据此，我们可以对上面已经提到的算法进行改进：

实现思路

1 先计算出第一期，以及第一期到第二期的最优决策值。

2 从第三期开始迭代，每次只用分四种情况计算最优值。1. 最近的两期都不生产。2. 最近的二期生产，最近的一期不生产。3. 最近的一期不生产，最近的二期生产。4. 最近的两期都生产。

3 比较上面四个结果，得出最优值以及方案。

具体实现过程

程序：*OnDySolution.m*

```
function [result,road] = OnDySolution(d,k,c,h)
    disp('璇疯□□ョ湅寮︹□□□□□□□');
    result = -1;
    road = -1;
    return
end
s = zeros(1,length(d));
r(1) = mToNCost(d,k,c,h,1,2);
s(1) = 1;
if length(d) <= 2
    result = r(1);
    road = 1;
    return
end
can = mToNCost(d,k,c,h,1,3);
bucan = mToNCost(d,k,c,h,1,2) + mToNCost(d,k,c,h,2,3);
if can <= bucan
    r(2) = can;
    s(2) = 1;
else
    r(2) = bucan;
    s(2) = 2;
for i = 3:length(d)
    z_z = mToNCost(d,k,c,h,s(i-2),i+1) + r(i-2) - mToNCost(d,k,c,h,s(i-2),i-1);
    z_o = mToNCost(d,k,c,h,s(i-2),i-1) + mToNCost(d,k,c,h,i-1,i+1) + r(i-2) - mToNCost(d,k,c,h,s(i
        -2),i-1);
    o_z = mToNCost(d,k,c,h,s(i-2),i) + mToNCost(d,k,c,h,i,i+1) + r(i-2) - mToNCost(d,k,c,h,s(i-2),
        i-1);
    o_o = mToNCost(d,k,c,h,s(i-2),i-1) + mToNCost(d,k,c,h,i-1,i) + mToNCost(d,k,c,h,i,i+1) + r(i
        -2) - mToNCost(d,k,c,h,s(i-2),i-1);
    r(i) = min([z_z,z_o,o_z,o_o]);
    switch r(i)
        case z_z
            s(i) = s(i-2);
        case z_o
            s(i) = i;
        case o_z
            s(i) = i-1;
        case o_o
            s(i) = i;
    end
    if mToNCost(d,k,c,h,s(i-1),i+1) + r(i-1) - mToNCost(d,k,c,h,s(i-1),i) <= r(i-1) + mToNCost(d,k
        ,c,h,i,i+1)
        s(i) = s(i-1);
        r(i) = mToNCost(d,k,c,h,s(i-1),i+1) + r(i-1) - mToNCost(d,k,c,h,s(i-1),i);
    else
        s(i) = i;
        r(i) = r(i-1) + mToNCost(d,k,c,h,i,i+1);
    end
end
```

```
result = r(length(d));
road = dyRoad(length(d),s);
end
```

正确性测试

测试程序伪代码:

```
% 随机生成d,k,c,h,以及c, h增加的同一个常数add
% 生成1000组测试用例
d,k,c,h,add = randi(10,1000,10);

for i = 1:1000
% 分别计算四种情况下的最优值
   [a1,b1] = dySolution(d(i,:),k(i,:),c(i,:),h(i,:));
   [a2,b2] = dySolution(d(i,:),k(i,:) + tem,c(i,:),h(i,:));
   [a3,b3] = dySolution(d(i,:),k(i,:),c(i,:) + tem,h(i,:));
   [a4,b4] = dySolution(d(i,:),k(i,:)+tem,c(i,:)+tem,h(i,:));
% 检验他们的最优决策方案是否相同
all([
  checkOptRoad(d(i,:),k(i,:),c(i,:) + tem,h(i,:),b1),
  checkOptRoad(d(i,:),k(i,:) + tem,c(i,:),h(i,:),b1),
  checkOptRoad(d(i,:),k(i,:) + tem,c(i,:)+tem,h(i,:),b1)
]) == 1
```

测试结果

测试用例	结果	要求
OnDySolution(1,1,1,1)	通过	结果为 2
OnDySolution(d,k,c,h)	通过	d,k 保持不变，c 或者 h 增加一个常数，最优方案不变
OnDySolution(d,k,c,h)	通过	k=0,c,h 保持不变，d 不影响最优决策

算法效率分析

使用该算法时，在计算更大一级的问题时，比其小两级的子问题的最优路线不会发生改变，这样就不用重复计算比较选择那个子问题。相当于所有阶段只计算了一次，算法的复杂度变为 O(n). 下面我们对其进行验证:

我们绘制出解决问题所需的时间随规模变化的曲线。(图 3, 图 4)

稀疏点拟合的参数分别为 $k = 0.1223 \cdot 10^{-4}$ 密集点拟合的参数分别为 $k = 0.6295 \cdot 10^{-4}$

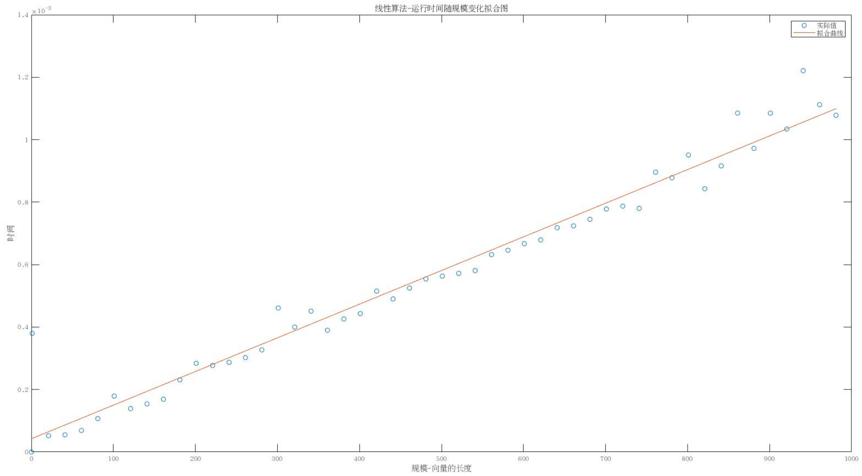

图 5: 稀疏点拟合

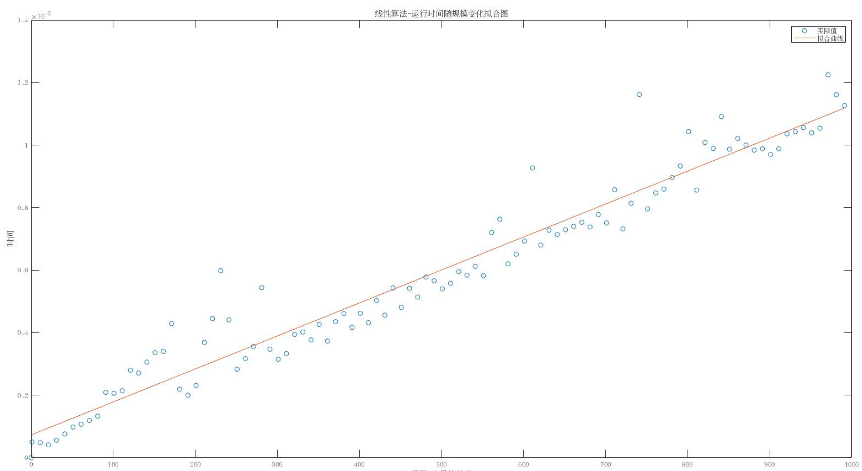

图 6: 密集点拟合

误差分析

使用上图中任一结果，分析其均方误差。$MSE = 0.00023$

数据来源如下图 5(每一项的误差值)。可知拟合的结果较好，所以不拒绝两个算法复杂度分别为 $O(n^2), O(n)$.