研究生《大数据优化建模与优化算法》大作业

姓　　名：　樊路林

学　　号：　23171214470

任课教师：　王宇平

一、若考虑处理机的释放时间，即，设从处理机 P_i 的释放时间为r_i（即从开始到时刻 r_i 从处理机 P_i 是非空闲的，从时刻 r_i 开始空闲，可以给其安排任务）。请

1. 叙述带有释放时间的同构网络可分任务调度问题；

主处理机和从处理机通过星形网络连接，主处理机负责数据的切分和传输，从处理机负责数据的处理。P_0 将大小为 W_{total} 的数据切分为为 N个子任务块并传输给 $P_1, P_2, \dots, P_n$, 同一时刻只能给一个从处理机 P_i 传输任务，P_i 只有完全接收该任务后才开始处理该任务，且只能在 r_i 时刻后才可以开始接受。令 z 表示从 P_0 传输单位数据到 P_i 所需要的时间，则传输大小为 α_i 的数据所需的时间为 $z\alpha_i$ ， w表示处理机 P_i 处理单位数据的时间,则处理 α_i 的时间为 $w\alpha_i$,由于考虑启动开销,所以子任务 α_i 的传输时间和处理时间分别为 $z\alpha_i$ 和 $w\alpha_i$ 。该问题就是求主处理机如何分配数据 P，使得所有从处理机处理完所有任务所用时间最短最短。

2. 建立有释放时间的同构网络可分任务调度问题的数学模型。

从处理机的调度顺序遵循释放时间递增的顺序，即$r_1 \le r_2 \le \cdots \le r_n$
记处理机P_i开始接收任务的时刻为S_i,由于考虑释放时间，所以$S_i \ge r_i$。S_i与前一个处理机的开始时间和数据传输时间有关，$S_i \ge S_{i-1} + z\alpha_{i-1}$。即$S_i = MAX(r_i, S_{i-1} + z\alpha_{i-1})$
P_0向P_i发送的数据量为α_i，所以子任务P_i的传输时间和处理时间分别为 $z\alpha_i$ 和 $w\alpha_i$,总共花费的时间为$S_i + z\alpha_i + w\alpha_i$。所有子任务传输的数据量加起来为总数据量，即$\alpha_1 + \alpha_2 + \cdots + \alpha_n = W_{total}$。故想要最后的总处理时间最小，只有当所有处理机同时完成，否则完全可以将后完成任务的处理机上的部分数据分配给先完成任务的处理机上执行,即$S_i + z\alpha_i + w\alpha_i = S_{i+1} + z\alpha_{i+1} + w\alpha_{i+1}$

$$T = r_1 + z\alpha_1 + w\alpha_1$$

$$s.t. \begin{cases} S_i = MAX(r_i, S_{i-1} + z\alpha_{i-1}) \\ \alpha_1 + \alpha_2 + \cdots + \alpha_n = W_{total} \end{cases}$$

将上述的一个约束优化问题求解出来就是有释放时间的同构网络的可分调度问题的解。

(1) 若$r_i \le S_{i-1} + z\alpha_{i-1}$恒成立，

$$S_i = S_{i-1} + z\alpha_{i-1} = r_1 + z\sum_{j=1}^{i-1}\alpha_j$$

代入得$\alpha_i = q\alpha_{i-1}$，$q = w/(z+w)$

代入约束优化可解得

$$\alpha_1 = W_{total} / \sum_{i=1}^{n} q^{i-1}$$

$$T = r_1 + z\alpha_1 + w\alpha_1 = (z+w)\alpha_1 + r_1 = \frac{(z+w)W_{total}}{\sum_{i=1}^{n} q^{i-1}} + r_1$$

(2) $r_i > S_{i-1} + z\alpha_{i-1}$ *恒成立，则*$S_i = r_i$*，则*

$$\alpha_i = \alpha_1 + (r_1 - r_1)/(z+w)$$
$$\alpha_1 + \alpha_2 + \cdots + \alpha_n = W_{total}$$

可以求解出α_1*，接着带入*

$$T = r_1 + z\alpha_1 + w\alpha_1$$

就可以解出 T。

(3) 混合时序约束，任意两个相邻的处理机之间可能满足约束条件（1），也可能满足约束条件（2）。若有 n 台处理机参与计算，所有的情况有 2^{n-1} *种。*

将处理机 P_{i-1} *和* P_i *满足约束条件（1）和满足约束条件（2）的情况分别记为* $M_i(1)$ *和* $M_i(2)$ *。其中 i=2, 3, …, n. 用 C=（*$c_2, c_3, \ldots c_n$*) 表示。若*c_i*等于*$M_i(1)$*，表示主处理机*P_0*在给从处理机*P_{i-1}*传输数据后紧接着给*P_i*传输数据，中间没有空闲，此时*$S_i = S_{i-1} + z\alpha_{i-1}$*。若*$c_i$*等于*$M_i(2)$*，表示主处理机*$P_0$*在给从处理机*$P_{i-1}$*传输数据后，需要等待*$P_i$*从忙碌转为空闲，中间存在空闲时间，此时*$S_i = r_i$*。*

带入求解线性方程组，即可得到分配方案的解。

二、对函数$f(x) = \sum_{i=1}^{9}[(1-x_i)^2 + 100(x_{i+1} - {x_i}^2)^2]$，初始点取为$x^0 = (0,0,\ldots,0) \in R^{10}$，分别用最速下降法和牛顿法编程迭代 10 次，把结果总结在如下形式的一张表里，记录各次迭代的函数值最终 CPU。比较两个方法所得到的结果，并分析结果。

迭代次数 k	最速下降法	牛顿法
	$f(x^k)$	$f(x^k)$
1	8.8976	8.7380
2	8.8062	8.1462
3	8.7235	7.4690
4	8.6517	6.7322

5	8.5854	5.9291
6	8. 5295	5.0902
7	8. 4770	4.2423
8	8. 4310	3.4208
9	8. 3866	2.6271
10	8. 3451	1.8429
10 次迭代两个方法所花时间	3.369 秒	14.3825 秒

最速下降法只使用了一阶导数，迭代 10 次所用到的时间较少，但是其函数值下降的也不如牛顿法下降的多。
牛顿法使用了二阶导数，迭代 10 次所用的时间相比于最速下降法较多，但是其函数值下降的也更多。

计算程序如下：

```
% 编程语言 Matlab

% 函数 get_f()用于得到目标函数
function f = get_f()
    x = sym('x',[1,10]);
    f = 0;
    for i = 1:9
        f = f + (1-x(i))^2 +100*(x(i+1)-x(i)^2)^2;
    end
end

% 函数 df1()用于对目标函数求一阶导
function dx = df1()
    f = get_f();
    x = sym('x',[1,10]);
    for i=1:10
        dx(i) = diff(f,x(i));
    end
end

```

```
% 函数 df2()用于对目标函数求二阶导
function d2x = df2()
    x = sym('x',[1,10]);
    dx = df1();
    for i = 1:10
        for j = 1:10
            d2x(i,j) = diff(dx(i),x(j));
        end
    end
end

% 函数 fast()为使用最速下降法的一次迭代
function  x_new =fast(x_value)
    x = sym('x',[1,10]);
    syms t;
    f = get_f();
    dx = subs(df1(),x,x_value);
    m = x_value - t * dx;
    fun_t = subs(f,x,m);
    diff_t = diff(fun_t);
    t_all = real(double(solve(diff_t)));
    t_1 = t_all(1);
    x_new = double(subs(m,t,t_1));
    y_new = double(subs(f,x,x_new));
    for i = 1:length(t_all)
        x_test = double(subs(m,t,t_all(i)));
        y_test = double(subs(f,x,x_test));
        if(y_test < y_new)
            x_new = x_test;
            y_new = y_test;
        end
    end
    disp(y_new);
end

% 函数 newton()为使用牛顿法的一次迭代
function  x_new =newton(x_value)
    x = sym('x',[1,10]);
    syms t;
    f = get_f();
    dx = subs(df1(),x,x_value);
    d2x = subs(df2(),x,x_value);
    d2x_1 = inv(d2x);
    % 下降方向 d
```

```
    d = - d2x_1 * dx.';
    % 求最佳步长
    m = x_value - t * d.';
    fun_t = subs(f,x,m);
    diff_t = diff(fun_t);
    t_all = real(double(solve(diff_t)));
    t_1 = t_all(1);
    x_new = double(subs(m,t,t_1));
    y_new = double(subs(f,x,x_new));
    for i = 1:length(t_all)
        x_test = double(subs(m,t,t_all(i)));
        y_test = double(subs(f,x,x_test));
        if(y_test < y_new)
            x_new = x_test;
            y_new = y_test;
        end
    end
    disp(y_new);
end

% 使用最速下降法迭代 10 次，并记录时间
x_value = [0,0,0,0,0,0,0,0,0,0];
tic
for i =1:10
    x_value = fast(x_value);
end
toc
disp(['最速下降法的运行时间: ',num2str(toc)]);

% 使用牛顿法迭代 10 次，并记录时间
x_value = [0,0,0,0,0,0,0,0,0,0];
tic
for i =1:10
    x_value = newton(x_value);
end
toc
disp(['牛顿法的运行时间: ',num2str(toc)]);
```