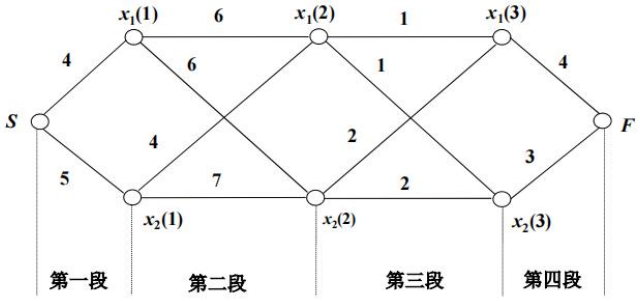
研究生课 最优化与最优控制 期末作业

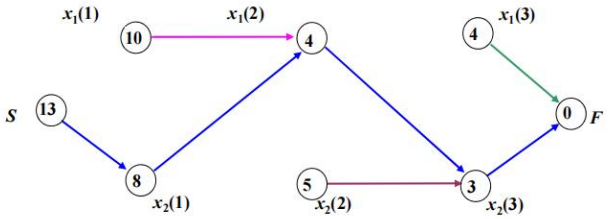
个人关于动态规划的记录、思考、问题汇总

课堂记录与拓展思考：

1. 动态规划是一种将复杂多级决策问题转化为一些列简单单级决策问题的算法。其本质在于状态的跳转和状态函数的传递；
2. 动态规划优化的核心和理论基础是最优性原理，即一个多级决策的最优问题可以分解为一个多级决策的最优子问题和决定子问题的决策问题；
3. 符合最优性原理的问题可以被抽象为一个有向无环图，即先前的决策可以影响此后的状态但此后的决策不会影响先前的状态。
4. 动态规划的思路可以用作上述有向无环图问题的求解。不止于优化问题，比如最简单的上楼梯问题也可以视作状态的转换和状态函数的传递，再此不再做展开；
5. 对于实用动态规划求解优化问题，每一个状态的代价可以由值函数表示 J(x)，每一步由原问题向子问题发展决策的代价也可以由决策函数 Q(x,u)表示。每一个状态值函数是其可做的最优决策的决策代价函数 J(x)=min(or max)Q(x,u)。
6. 决策代价函数包含当前决策造成的代价 j[x*(k-1),u(k-1)]，也包含这一决策所造成的此后状态的代价 J[x*(k),u*(k)]，而让此后状态的代价最小就是原始最优问题的子问题；
7. 相比起暴力穷举法穷举的对象是整个过程所有的决策路线，动态规划穷举的是每个状态的状态函数；
8. 在进行每一步的最优过程中，动态规划还是要枚举所有的状态跳转的可能。当每一步状态跳转由离散向连续转变时，可能的状态跳转呈指数级增加；
9. 维数灾产生的原因是动态规划本质还是一种优化了的穷举方法；
10. 上述动态规划的讨论均是基于状态确定、代价计算方法已知的基础上的，如果对于问题中状态跳转的代价不清楚，则需要引入学习策略来描绘模型。



最短行车路线多段决策路网



从各站开始的最短行车路线

机械工程学院 19 级本科生 李佩泽

研究生课 最优化与最优控制 期末作业

关于 DP 我自己也做了一些未成体系的工作：

首先，我根据 B 站作者 DR_CAN 的教学视频：

【最优控制】2_动态规划_Dynamic Programming_基本概念_哔哩哔哩_bilibili

【最优控制】3_动态规划_Dynamic Programming_代码详解_哔哩哔哩_bilibili

仿照其程序思路进行了一个简单运动问题动态规划优化方法的程序案例

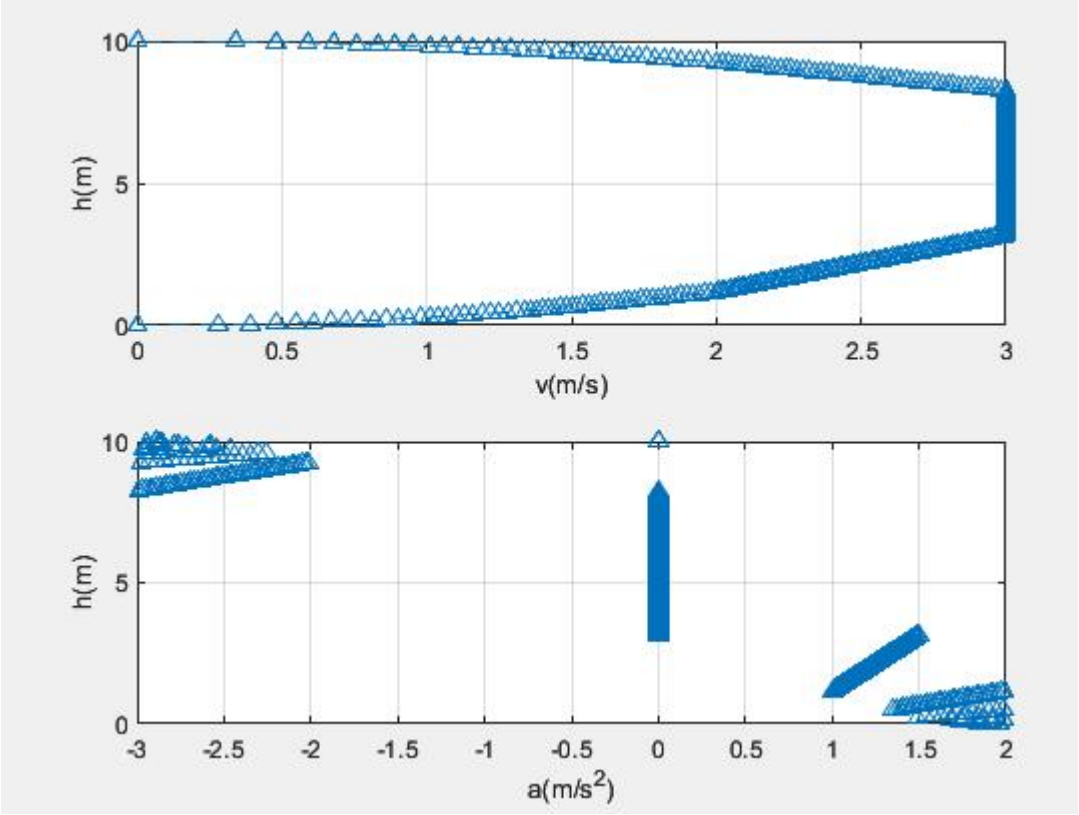
这一问题的具体描述是：

一个一维运动的点起始位置和终止位置距离已知，规定了运动的速度和加速度上限，试以动态规划求解使点运动时间最短轨迹（trajectory）。

其算法思路为以这一点距离起始位置的距离作为求解的各个步，每一个不同距离上的运动速度作为这一步下可选的状态。点在两步之间的运动过程视为一个匀加速运动过程，这样可以根据“步”中提供的位置信息和每一步状态中提供的速度信息计算每一步运动中所花费的时间，并将这一步与此前运动过程的时间之和作为决策代价函数进行动态规划。

在程序实现层面，将 J 以求解步骤为行、每步内运动状态为列储存于一个矩阵中，将每两步之间的决策代价 J_temp 按照决策前状态为行、决策后状态为列的格式储存在一个临时矩阵中，并以 J_temp 每一行的最小值更新 J 中最下方的一列。在完成动态规划的求解后根据的代价自减逐步回溯决策过程。

在 DR_CAN 原程序的基础上，我进行了程序理解、对每一“步”迭代过程的函数进行了封装，并尝试了将点的加速度大小也作为代价函数一部分的尝试。



机械工程学院 19 级本科生 李佩泽

实现代码：

myDPmain.m

```
% create state array
clc; clear all; close all;

% Define IC
h_init = 0; % h: height
v_init = 0; % v: velocity
% Final state
h_final = 10;
v_final = 0;
% Boundary condition
h_min = 0;
h_max = 10; N_h = 500;
v_min = 0;
v_max = 3; N_v = 300;
% Create state array
Hd = h_min:(h_max-h_min)/N_h:h_max;
Vd = v_min:(v_max-v_min)/N_v:v_max;
% Input constraint, input is the system acceleration
u_min = -3; u_max = 2;
% Define cost to go matrix
J_costtogo = zeros(N_h+1,N_v+1);
% Define input acceleration matrix
Input_acc = zeros(N_h+1,N_v+1);

%%%% From 10m to 8m %%%%
[ J_costtogo , Input_acc ] = DynaPlanLayer( 2 , J_costtogo , Input_acc , ...
(h_max-h_min)/(N_h) , [0] , Vd , u_min , u_max );
%%%% From 8m to 2m %%%%
for k = 3:1:N_h
[ J_costtogo , Input_acc ] = DynaPlanLayer( k , J_costtogo , Input_acc , ...
(h_max-h_min)/(N_h) , Vd , Vd , u_min , u_max );
end
%%%% From 2m to 0m %%%%
[ J_costtogo , Input_acc ] = DynaPlanLayer( N_h+1 , J_costtogo , Input_acc , ...
(h_max-h_min)/(N_h) , Vd , [0] , u_min , u_max );

%%%%%%%%%%%%%% Plot %%%%%%%%%%%%%%%%%%%%
```

```
% 作者：DR_CAN
https://www.bilibili.com/read/cv20060597?spm_id_from=333.999.0.0 出处：
bilibili
h_plot_init = 0; % Initial height
v_plot_init = 0; % Initial velocity
acc_plot = zeros(length(Hd),1); % Define acc plot array
h_plot = zeros(length(Hd),1); % Define height plot array
v_plot = zeros(length(Hd),1); % Define velocity plot array
h_plot (1) = h_plot_init; % First value
v_plot (1) = v_plot_init; % First value

for k = 1 : 1 : N_h
[min_h,h_plot_index] = min(abs(h_plot(k) - Hd)); % Table look up
[min_v,v_plot_index] = min(abs(v_plot(k) - Vd)); % Table look up
acc_index = sub2ind(size(Input_acc), N_h+2-h_plot_index, v_plot_index); % Find
control input acceleration
acc_plot (k) = Input_acc(acc_index); % Save acceleration to the matrix
v_plot (k + 1) = sqrt((2 * (h_max - h_min)/N_h * acc_plot(k))+ v_plot (k)^2); %
Calculate speed and height
h_plot (k + 1) = h_plot(k) + (h_max - h_min)/N_h;
end

% Plot
subplot(2,1,1);
plot(v_plot,h_plot,'--^'),grid on;
ylabel('h(m)');
xlabel('v(m/s)');
subplot(2,1,2);
plot(acc_plot,h_plot,'^'),grid on;
ylabel('h(m)');
xlabel('a(m/s^2)');
```

DynaPlanLayer.m

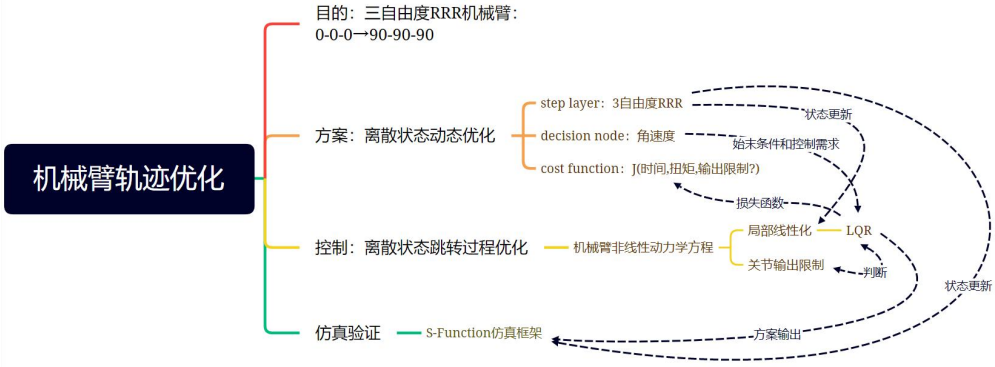
```
% Dynamics Planning Layer Fuction
% used to calculate the itration of DP
% created by StvLi 2022-12-10
% based on idea from Dr.CAN
% renew: general form that be able to handle links between two layers
% which could contain any number of nodes
%
% Inputs: k - recipocal number of the current layer
% J - loss fuction matrix
```

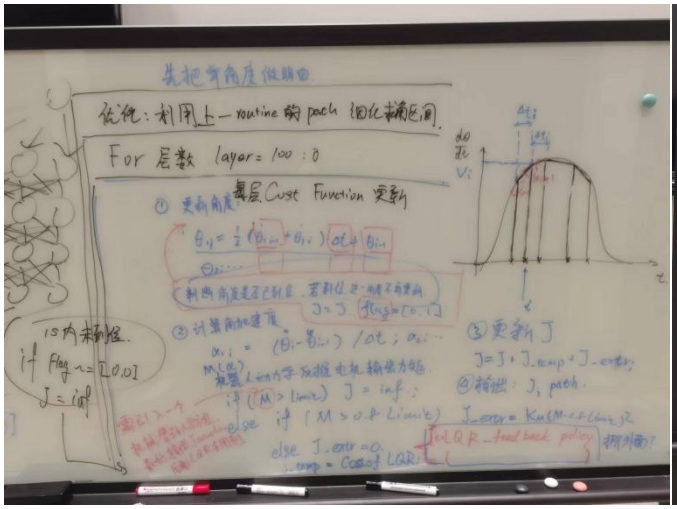
```
% I - input matrix
% h_delta - height difference between two layers
% Vd_up - vector of upper layer
% Vd_dn - vector of lower layer
% u_min - minimum of input value
% u_max - maximum of input value
%
% Outputs:J - loss fuction matrix
% I - input matrix
function [ J , I ] = DynaPlanLayer( k , J , I , ...
h_delta , Vd_up , Vd_lw , u_min , u_max )
[vd_x,vd_y] = meshgrid(Vd_up,Vd_lw); % Prepare the matrix
v_avg = 0.5*(vd_x+vd_y); % Calculate average time
T_delta = (h_delta)./v_avg; % Calculate travel time, this is the cost
inp = (vd_x - vd_y)./T_delta; % Calculate acceleration
J_temp = T_delta; % Assign delta T to cost to go
[acc_x,acc_y] = find(inp<u_min|inp>u_max); % FIne Acc is over the limit
Ind_lin_acc = sub2ind(size(inp),acc_x,acc_y); % Fine linear index
J_temp(Ind_lin_acc) = inf; % Let certain elements to infinity
%%%% Very Important Step!  %%%%
[~,m] = size(Vd_up);
[~,n] = size(Vd_lw);
for row = 1:n
for col =1:m
J_temp(row,col) = J_temp(row,col) + J(k-1,col);
end
end
for row = 1:length(Vd_lw)
[J(k,row),pos(row)] = min(J_temp(row,:)); % Save to cost to go matrix
I(k,row) = inp(row,pos(row)); % Save to acceleration matrix
end
end
```

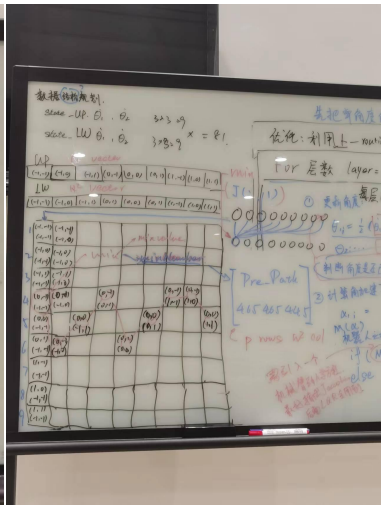
机械工程学院 19 级本科生 李佩泽

研究生课 最优化与最优控制 期末作业

在此基础上，我自己产生了一个使用动态规划进行机械臂移动路径整体规划的的想法，但因为感染新冠的身体原因和本科毕业设计开题答辩没有完成：







这一部分因为开题答辩和感染新冠的原因没有完成动力学、代价函数、惩罚函数和主函数的代码，只完成了动态规划迭代函数部分的代码：

```
% Dynamics Planning Layer Fuction
% used to calculate the itration of DP
% created by StvLi 2022-12-10
% based on idea from Dr.CAN
% renew: general form that be able to handle links between two layers
% which could contain any number of nodes
%
% Inputs: k - recipocal number of the current layer
% J - matrix of loss fuction (p,n*n)
% Path - matrix of scheme matrix (p,n*n)
% Theta_up - matrix of upper angles (2,n*n)
% Vd_up - matrix of upper layer (2,n*n)
% Vd_lw - matrix of lower layer (2,n*n)

%
% Outputs:J - loss fuction matrix
```

机械工程学院 19 级本科生 李佩泽

```
% Path - scheme matrix
%
function [J,Path] = myArmDPLayer( k , J , Path , Theta_up , ...
Vd_up , Vd_lw , dt )
% Judge whether the angle motion has been finished
flag = [1 1];
if Theta_up(1) > 90
flag(1) = 0;
end
if Theta_up(2) > 90
flag(2) = 0;
end

if flag(1) ~= 0 && flag(2) ~= 0 % motion hasn`t been finished
% Matrix Arrangement
% Theta_up → a_1 a_2
[~,a_1] = meshgrid( Vd_lw(1,:) , Theta_up(1,:) );
[~,a_2] = meshgrid( Vd_lw(2,:) , Theta_up(2,:) );
% Vd_up → da_1p da_2p Vd_lw → da_1c da_2c
[da_1c,da_1p] = meshgrid( Vd_lw(1,:) , Vd_up(1,:)' );
[da_2c,da_2p] = meshgrid( Vd_lw(2,:) , Vd_up(2,:)' );

% Calculate present angle to go in future steps
a_1 = 0.5*( da_1p + da_1c )*dt + a_1;
a_2 = 0.5*( da_2p + da_2c )*dt + a_2;

% Calculate angular accelerations
dda_1 = ( da_1c - da_1p)/dt;
dda_2 = ( da_2c - da_2p)/dt;
% Calcutate motors` output moments
[ m_1 , m_2 ] = armDynamicModel(a_1,da_1c,dda_1,a_2,da_2c,dda_2);
% Calculate COST FUNCTION
J_temp = costOutput( m_1 , m_2 ); % Output cost
J_temp = J_temp + costPenalty( m_1 , m_2 ); % Penalty cost

% Renew the PLAN COST & PATH with least COST scheme
for col = 1:length(Vd_lw)
[J(k,col),pos(col)] = min(J_temp(:,col)); % renew cost
Path(k,col) = pos(col); % renew path
end
else
% No need to renew parameters & direct Output
end
end
```

研究生课 最优化与最优控制 期末作业

在完成这一方案方案的过程中，我思考了如下几个问题：

1. 状态跳转的“步”如何选择？是否有必要将其与某一状态绑定？

我最初的思路是沿用上面点运动的思路，将机械臂一个轴的转角作为“步”的划分依据，但之后我发现从 0-0 到 90-90 的机械臂运动过程中，单取一个轴的转角作为划分依据首先会造成两个角度在每一步状态上的不对称。之后又想到在动态规划中，步的概念也只是作为求解起止和数据组织的工具，并不直接参与 DP 算法。所以，我思考之后的结论是以两轴不同的转速组合作为可选的状态，将两轴的角度作为状态函数矢量的一部分在每一次迭代计算中更新并传递（优化运算中也要用角度构建动力学模型计算代价函数）；

2. 保存所有状态的代价函数，在完成 DP 求解后再通过自减寻找最优方案的方法如何改进？

我一个想法是，在利用上面的以矩阵（数组）的方式记录数据的基础上，在每一轮迭代过程中再传递一个记录下一步每一状态最优的上一步状态的方案矩阵（类似于指针矩阵数组），在完成 DP 后以第一步（最后一行）方案矩阵指向的（倒数第二行）的相应位置并以此类推去检索最优方案。之后我又想到，这种数据的组织形式本质上是一种链表。如果我能通过一个结构体，在结构体中储存状态函数信息，并在结构体内指向最优的下一步状态，便可组织起一个已知状态出发并终止于已知状态的链表。在实际计算中，每一步都有大量的状态对应的结构体会位于上一步状态的决策链表之外，及时地释放掉这些没有其他结构体指向的结构体可以节约硬件存储空间。

3. 在不引入复杂方法的前提下如何使用离散状态求解连续问题并避免维数灾？

避免维数灾的最简单办法是使每一步的可选状态不要过多。在完成各个状态的代价计算后，可以用各个取到的状态的代价进行插值，估计其未取到的中间状态的代价，从而得到可能的更优方案。我还想到，如果在分析实际的连续问题时，可以在第一轮粗分网格求解后，以粗分的最优方案去指导下一轮求解网格细化方向。

比如上述机械臂问题中，限制两轴转速分别在-0.5rad/s ~ +0.5rad/s 以每 0.05rad/s 为单位划分状态网格进行以第一次求解，每一步共有 21*21=441 种可能的状态。在这一轮求解中，若求得某一步的最优解为 1 轴转速 0.15rad/s，2 轴转速 0.05rad/s，因为物理解的连续性，可以在下一轮中把当前步的可能状态设为 1 轴转速 0.100rad/s ~ 0.200rad/s，2 轴转速 0.000rad/s~0.100rad/s 以 0.005rad/s 为单位划分状态网格进行第二轮计算。这样通过多轮回滚实现某一精度要求下的连续状态优化求解。

4. “从后向前”进行决策的意义是什么？

我最一开始惯性地以为，“从后向前”对应的是时间顺序层面的从后向前，正如上面 DR_CAN 的运动点运动控制案例中代码实现是从运动的末状态向之前状态进行递推。但是我之后发现在这一案例中时间顺序的“前”和“后”是完全对称的，这个问题从起始点开始求解和从终止点开始求解时没有本质区别的。

回想起最优原理的概念，我意识到“从后向前”指的是因果关系的从果向因，即从被决策的子问题向决定子问题的决策进行递推，是从有向无环图被指向的节点逆着指向其的有向线进行递推。而在运动点的案例中，每一步的决策并没有对除了这一决策前后各一步之外的状态产生影响，或者说因为其将运动坐标取作“步”，实际上解除了决策对此后状态的因果关系。可以说这种巧妙的方式虽然没有必要，但使 DP 问题大为简化。（这一思考实际上回答了我的问题 1.）

机械工程学院 19 级本科生 李佩泽