Instruction for running matlab code.

The matlab codes implements the open-loop optimal control and closed-loop optimal control of a Markov process. For details, please refer to the orginal article:

"Optimal treatment strategies in the context of 'treatment for prevention' against HIV-1 in resource-poor settings." - S. Duwal, S. Winkelmann, C. Schütte and M. von Kleist, PLoS Comput. Biol., 11, e1004200, 2015

Open-Loop Optimal Control - Pro-active strategy.

The matlab codes for the open-loop optimal control are in the folder `Algorithm_I_OpenLoop`.

The detail mathematical derivation of open-loop optimal control can be found in the Supplementary Text 2.

The relation of different codes to one another and their brief description are shown as below :

- ExecuteSimulation - is the main wrapper script to run the example. Different parameters to run the algorithm are set here and it calls the following functions :
    - func_execute - the main function file. This function file calls following scripts and the function executing algorithm :
        - SystemParams_New2 - is a script file where different variables are initialized.
        - CalculationParams - is a script which performs the precomputations of cost vectors and constraints. This allows faster execution of the main algorithm. Following function files are used for precomputation :
            - MakeGenerator - is a function file which creates the generator matrices for different actions.
            - MakeTerminalCostFunction3 - is a function file which creates the terminal cost vector. The terminal cost vector is computed assuming that the first action is continued after the considered time horizon.
            - MakeTerminalCostFunction4 - is a function file which creates the terminal cost vector. The terminal cost vector is computed assuming that the second action is continued after the considered time horizon.
        - runBDA_parallel - is the function file which executes the dynamic programming algorithm with backward propagation. If the `matlab parallel computing toolbox` is available, the function allows the parallelization.
        - runBruteForceF - is the function which executes the brute force algorithm with forward propagation. This function can be used to cross-check the result of dynamic programming algorithm for cases with a small number of intervals.

- Plot_OpenLoopTrajectories - is a script that plots the propagation of probability state vector under a specified control (treatment strategy).

Selection of Algorithm

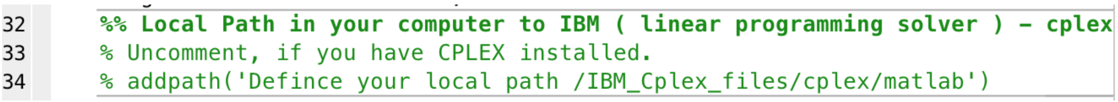
The main wrapper script allows selection between the dynamic programming algorithm with backward algorithm and the brute force algorithm with forward propagation. The execution time of brute force algorithm grows exponentially with the number of intervals. Thus, the script only allows the execution of brute force algorithm, if the number of interval is less or equal to 15. The algorithm selection can be done in ExecuteSimulation.

Linear Programming Solver

The dynamic programming algorithm with backward propagation requires solving series of linear programming problem. We recommend using the linear programming solver from `IBM ILOG Suite`. The default is the linear programming solver `linprog` of matlab (does not require IBM ILOG Suite).

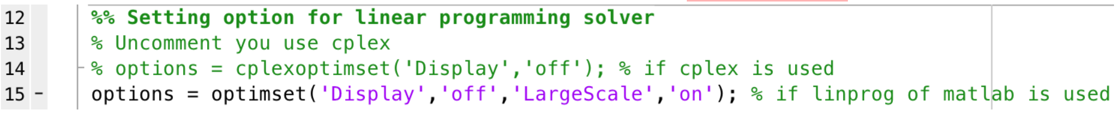
Following codes need to be changed in order to enable use of IBM ILOG Suite:

1. Uncomment and set the local path to the IBM Cplex folder with matlab files in ExecuteSimulation :



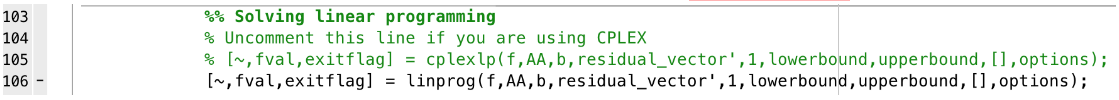
```
%% Local Path in your computer to IBM ( linear programming solver ) - cplex
% Uncomment, if you have CPLEX installed.
% addpath('Defince your local path /IBM_Cplex_files/cplex/matlab')
```

2. Uncomment the first line and comment out the second line in the file runBDA_Parallel.



```
%% Setting option for linear programming solver
% Uncomment you use cplex
% options = cplexoptimset('Display','off'); % if cplex is used
options = optimset('Display','off','LargeScale','on'); % if linprog of matlab is used
```

3. Uncomment the first line and comment out the second line in the file runBDA_Parallel.



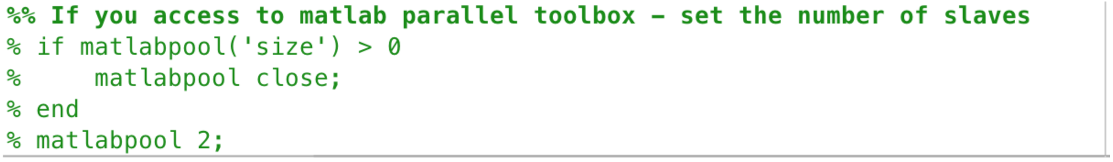
```
            %% Solving linear programming
            % Uncomment this line if you are using CPLEX
            % [~,fval,exitflag] = cplexlp(f,AA,b,residual_vector',1,lowerbound,upperbound,[],options);
            [~,fval,exitflag] = linprog(f,AA,b,residual_vector',1,lowerbound,upperbound,[],options);
```

Parallelization

For each interval, a number of linear programming problem needed to be solved, which can be parallelly performed for a faster execution. This requires the `parallel computing toolbox` package from Matlab. We recommend using the linear programming solver from IBM ILOG Suite. The default is the linear programming solver `linprog` of matlab.

Following codes need to be changed :

1. Uncomment the following part and set the number of slave cores in ExecuteSimulation file.



```
%% If you access to matlab parallel toolbox - set the number of slaves
% if matlabpool('size') > 0
%     matlabpool close;
% end
% matlabpool 2;
```

Closed-loop Optimal Control - Diagnostic Guided Strategy

The matlab codes for the closed-loop optimal can be found in the folder `Algorithm_II_ClosedLoop`.

The detailed mathematical derivation can be found in the "Markov Control Processes with Rare State Observation: Theory and Application to Treatment Scheduling in HIV-1" S. Winkelmann, C. Schütte and M. von Kleist. Communications in Mathematical Sciences 12, 859, 2014 and is also briefly described in Supplementary Text 1

- CompleteCalculation - is the main script which implements the closed-loop optimal control algorithm.
    - DisInfoPolicyFunction - is the function file that execute the policy iteration algorithm.
    - MakeGenerator - is the function file that creates the generator matrices for the actions.
    - MakeCostFunction - is the function file that makes the cost function.
    - NettoCosts - is the function that calculates the total cost.