```
clear
clc
close all
```

Dynamic Goals_Based Wealth Management Using Dynamic Programming

Building constant variables

Here We define all the variables for the model.

1. cov_mat: a n by n covariance matrix for n assets' returns, which can be generated from historical data.
2. avg_arr: a n by 1 vector for n assets's expected returns, which can be generated from historical data.
3. m: int, deciding how many points to sample from efficient frontier.
4. mus: a m by 1 vector for m expected portfolio returns, equally spaced between the min-max expected return.
5. T: int, total time period.
6. cash: a T by 1 vector for cash flow at every timestep.
7. rho: double, deciding how many wealth grid points to generate.
8. G: double, target wealth at the terminal period.
9. w0: double, initial wealth at the initial period.
10. total_itr: int, param for visualization.

```
cov_mat = [0.0017, -0.0017, -0.0021; -0.0017, 0.0396, 0.03086; -0.0021, 0.0309, 0.0392
avg_arr = [0.0493; 0.0770; 0.0886];
m = 15; % a vector of the n expected returns
mus = reshape(linspace(.0526, .0886, m), [], 1); % the maximum portfolio expected retu
T = 11; % years
cash = zeros(T, 1); % cash flows
rho = 1;
G = 200;
w0 = 100;
total_itr = 100;
```

Dynamic Programming Algorithm

This is the whole pipeline for calculating the probability that the target will be realized at the terminal period.

The pipeline is as follow:

1. caculate efficient frontier ***ef***
2. generate wealth grid point ***grid*** and the index of w0 in the grid ***w0_idx***
3. caculate transition probability from each grid point at t timestep to every grid point at t+1 timestep, and store it as ***tp_tables***
4. caculate ***V*** and best actions(best expected portfolio return mu at each timestep) ***best_ms***
5. caculate the probability distribution for the wealth grid points in every timestep t ***p_table***

6. caculate the probability that the target will be realized at the terminal period.

```
tic
%calculated the constants, a, b, and c for efficient frontier
[a, b, c] = cal_efficient_frontier_params(avg_arr, cov_mat);
%calculated the efficient frontier
ef = cal_efficient_frontier(mus, a, b, c);
% create the state space with wealth grid points
[grid, w0_idx] = gen_grid(w0, cash, ef, rho, {-3, 3}, 1);
```

```
Grid size: 117
```

```
tp_tables = get_trans_prob_tables(ef, grid, T, cash);
[best_ms, v_tables] = V(ef, grid, tp_tables, G, T);
p_table = calc_wealth_prob(grid, w0_idx, best_ms, tp_tables, T);
[p_len, p_width] = size(p_table);
G_idx = sum(grid < G) + 1;
prob = sum([p_table{p_len, G_idx:p_width}]);
prob
```

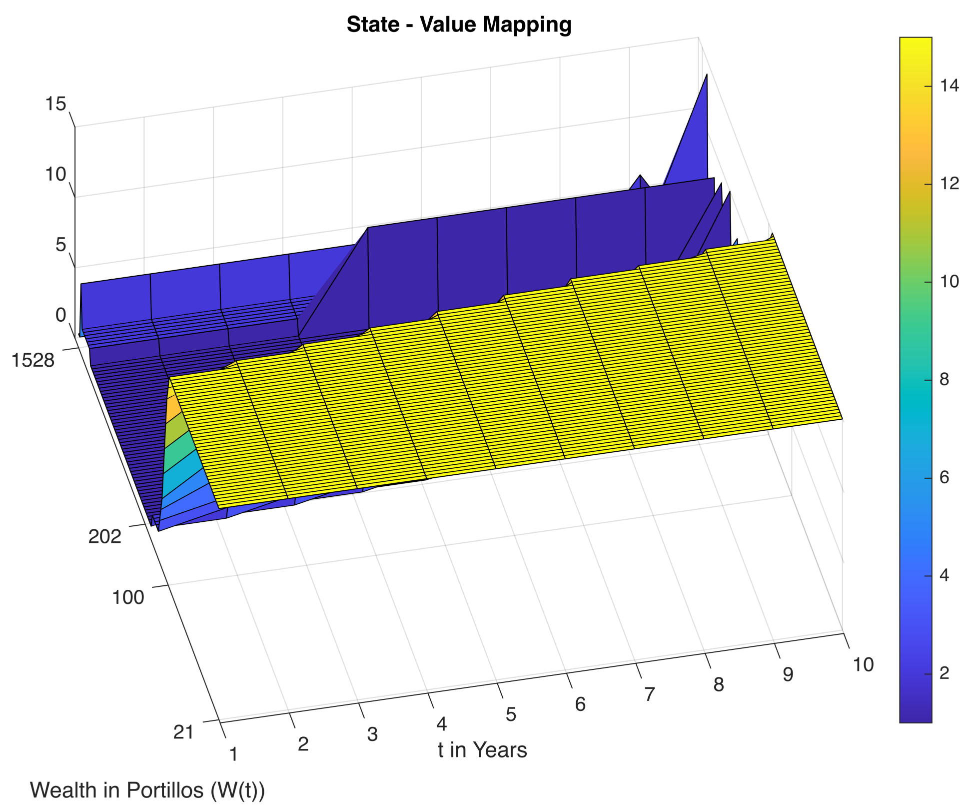
```
prob = 0.6756
```

```
toc
```

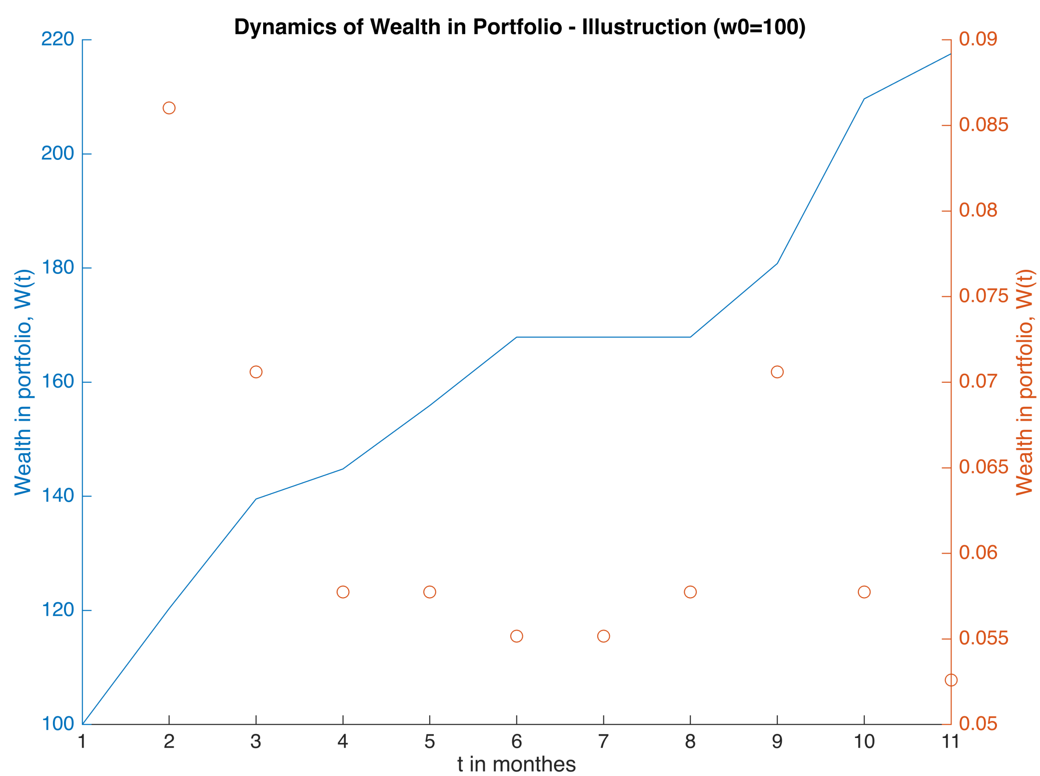
```
Elapsed time is 10.459771 seconds.
```

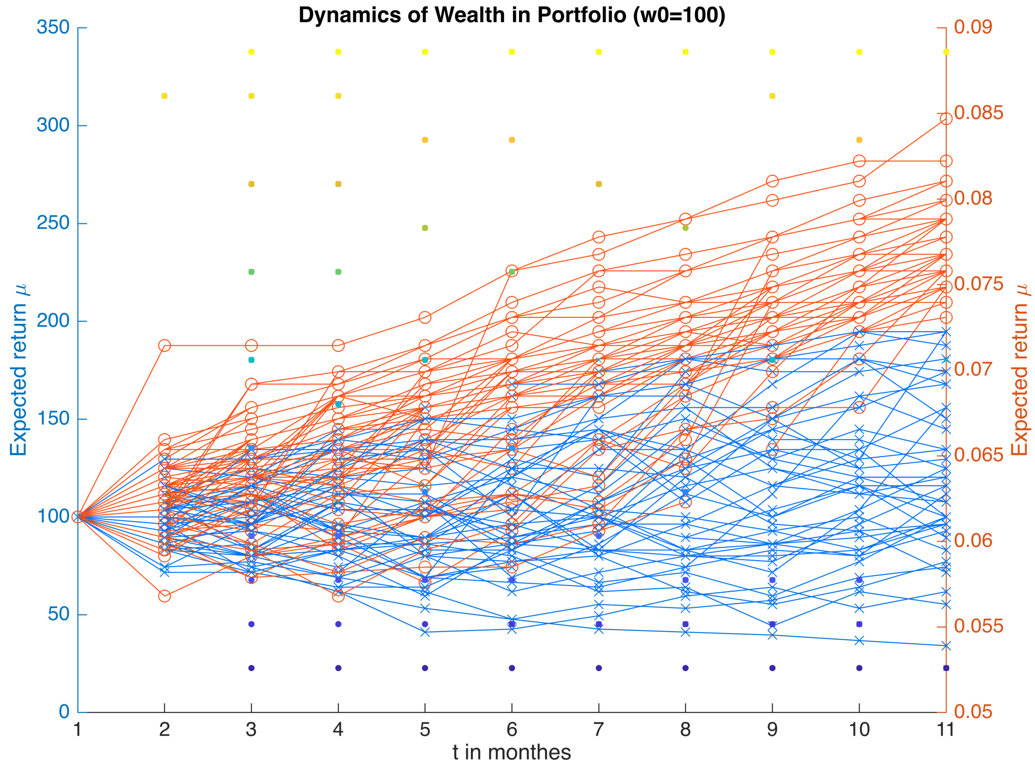
```
%  best action demonstration

best_action = dp_plot_1(grid, v_tables, T, w0_idx, G_idx, ef);
```




```
traces = dp_plot_2(ef, total_itr, T, w0_idx, G, best_ms, tp_tables, grid);
```





Efficient Frontiersize(grid, 1)

- For any given portfolio volatility, σ, it is always optimal to maximize the portfolio expected return, μ
- $\sigma = \sqrt{a\mu^2 + b\mu + c}$
- $a = h^\top \Sigma h$
- $b = 2g^\top \Sigma h$
- $c = g^\top \Sigma g$
- The constants, a, b, and c are defined by m, which is a vector of the n expected returns; o, which is a vector of n ones
- $g = \dfrac{l\Sigma^{-1}o - k\Sigma^{-1}m}{\mathrm{lp} - k^2}$
- $h = \dfrac{p\Sigma^{-1}m - k\Sigma^{-1}o}{\mathrm{lp} - k^2}$
- $\boldsymbol{\Sigma}$, which is the $n \times n$ covariance matrix of the n assets
- $k = m^\top \Sigma^{-1} o$
- $l = m^\top \Sigma^{-1} m$
- $p = o^\top \Sigma^{-1} o$

```
function [a, b, c] = cal_efficient_frontier_params(avg_arr, cov_mat)
    % we first find the constants, a, b, and c are defined by m, which is a
```

```
    % vector of the n expected returns.
    %
    % Args:
    %   avg_arr: a n by 1 vector for n assets's expected returns.
    %   cov_mat: a n by n covariance matrix for n assets' returns.
    % Returns:
    %   a, b, c: parameters for caculating portfolio's variance on the
    %          efficient frontier given expected returns.
    o = ones(size(avg_arr));
    inv_cov = inv(cov_mat);
    k = avg_arr.' * inv_cov * o;
    l = avg_arr.' * inv_cov * avg_arr;
    p = o.' * inv_cov * o;

    g = (l .* inv_cov * o - k .* inv_cov * avg_arr) ./ (l .* p - k .^ 2);
    h = (p .* inv_cov * avg_arr - k .* inv_cov * o) ./ (l .* p - k .^ 2);

    a = h.' * cov_mat * h;
    b = 2 * g.' * cov_mat * h;
    c = g.' * cov_mat * g;
end

function ef = cal_efficient_frontier(mus, a, b, c)
    % caculates the efficient frontier of given size.
    %
    % Args:
    %   mus: a m by 1 vector for m expected portfolio returns.
    %   a, b, c: returns from function cal_efficient_frontier_params.
    %            parameters for caculating portfolio's variance on the
    %            efficient frontier given expected returns.
    % Returns:
    %   ef: efficient frontier, which is a m by 2 array, first
    %       column for expected returns, second column for corresponding
    %       variance.
    cal_sig = @(mu) sqrt(a .* mu .^ 2 + b .* mu + c);
    sigs = arrayfun(cal_sig, mus);%Apply function to each element of array
    ef = cat(2, mus, sigs);%first column mus, second column sigs
end
```

The State Space Gridpoints

Evolution of W(0)

- The paper has chosen to use geometric Brownian motion as stochastic model for the growth of W(0)
- �
- *Z* is a standard normal random variable
- The paper assumes that Z realistically takes values between –3 and 3

- The smallest realistic value for W(t) corresponds to computing equation (2) after setting Z = −3, $\mu = \mu$min, and σ equal to σmax
- �
- The largest realistic value is computed by again using σ = σmax, but replacing Z with 3 and μ with μmax
- �

Grid points

- Starting with ln(ˆWmin), we add a grid point every σmin/ρgrid units, stopping once we reach or surpass ln(ˆWmax)
- $\frac{\sigma_{\min}}{\rho_{\text{gird}}}$
- This yields a total of imax + 1 grid points where imax equals (ln(ˆWmax)−ln(ˆWmin))ρgrid /σmin after rounding up to the nearest integer.
- We equally shift all of these imax + 1 values downward by the smallest amount necessary to match one of these values to ln(W(0))
- Finally, we exponentiate all imax + 1 values to obtain our wealth grid values, *W*0 through *Wi*max

```
function [grid, w0_idx] = gen_grid(w0, cash, efficient_frontier, rho, Z_range, h)
    % Generates wealth grid according to gaussian distribution.
    %
    % Args:
    %   w0: double, initial wealth at the initial period.
    %   cash: a T by 1 vector for cash flow at every timestep.
    %   efficient_frontier: a m by 2 array, each row represents a (mu, sig)
    %                       pair on the efficient frontier.
    %   rho: double, deciding how many wealth grid points to generate.
    %   Z_range: [z_min, z_max], Z is a standard normal random variable,
    %            and z_min, z_max are the range manually set.
    %   h: param for decision frequency in the time period.
    %
    % Returns:
    %   grid: g by 1 array, stores every wealth level at ascending order.
    %   w0_idx: int, index of w0 in the grid. That is, w0 = grid(w0_idx).
    [z_min, z_max] = Z_range{:};%Z is standard normal random variable from -3 to 3
    t = size(cash, 1);%gain from T and cash, now T = 11 so t = 11
    mu_tmp = efficient_frontier(:, 1); %extract mu from ef
    sig_tmp = efficient_frontier(:, 2);%extract sigma from ef
    mu_min = min(mu_tmp);
    mu_max = max(mu_tmp);
    sig_max = max(sig_tmp);
    sig_min = min(sig_tmp);

    emap_max = containers.Map('KeyType', 'int32', 'ValueType', 'double');
    emap_min = containers.Map('KeyType', 'int32', 'ValueType', 'double');
```

```
    for i = 1:t
        emap_max(i) = exp((mu_max - sig_max .^ 2 ./ 2) .* h .* i + z_max .* sig_max .*
        emap_min(i) = exp((mu_min - sig_max .^ 2 ./ 2) .* h .* i + z_min .* sig_max .*
    end

    w_min = Inf;
    w_max = -Inf;
    for i = 1:t
        cur_min = ( ...
            w0 .* emap_min(i) + ...
            sum(arrayfun(@(j) cash(j) .* emap_min(i - j + 1), 1:i)) ...
            );
        cur_max = ( ...
            w0 .* emap_max(i) + ...
            sum(arrayfun(@(j) cash(j) .* emap_max(i - j + 1), 1:i)) ...
            );
        w_min = min(cur_min, w_min);
        w_max = max(cur_max, w_max);
    end

    grid = [];
    cur = log(w_min);
    log_max = log(w_max);
    while cur < log_max %construct gird
        grid = [grid; cur];
        cur = cur + sqrt(h) .* sig_min ./ rho;% add a grid point for every sjgma /rho
    end
    grid = [grid; log_max];

    log_w0 = log(w0);
    shift = Inf;
    w0_idx = 1;
    for i = 1:size(grid, 1)%locating index of w0
        x = grid(i);
        if x - log_w0 >= shift && x - log_w0 >= 0
            continue
        else
            shift = x - log_w0;
            w0_idx = i;
        end
    end
    grid = grid - shift;
    grid = exp(grid);
    disp(['Grid size: ' num2str(size(grid, 1))]);
end
```

Transition probability

- Beginning by determining the transition probabilities, p(Wj (t + 1)|Wi(t), μ)
- The transition probability is the normalized relative probability that we will be at the wealth node Wj at time t + 1 if we start at the wealth node Wi at time t and, between times t and t + 1, our portfolio is run with an expected return of μ and its corresponding volatility, σ
- $$p^{\sim}\left(W_j\left(t+1\right)\middle|W_i\left(t\right),\mu\right)=\phi\left(\frac{1}{\sigma}\left(\ln\left(\frac{W_j}{W_j+C(t)}\right)-\left(\mu-\frac{\sigma^2}{2}\right)\right)\right)$$
- Defining $\phi(z)$ to be the value of the probability density function of the standard normal random variable at $Z=z$
- Normalizing these probability density function values yields the desired transition probabilities
- $$\frac{p^{\sim}(W_j(t+1)|W_i(t),\mu)}{\sum_{k=0}^{i_{\max}} p^{\sim}(W_k(t+1)|W_i(t),\mu)}$$

```
function transfer_prob_ = cal_transfer_prob(t, i, j, ef_pair, grid, cash, h)
    stn_pdf = @(x) exp(-x .^ 2 ./ 2) ./ sqrt(2 .* pi);

    mu = ef_pair(1);
    sig = ef_pair(2);

    transfer_prob_fn = @(j) stn_pdf(1 ./ sig .* (log(grid(j) ./ (grid(i) + cash(t) )) -
    transfer_prob_ = transfer_prob_fn(j);
end

function tp_tables = get_trans_prob_tables(ef, grid, T, cash)
    tp_tables = cell(size(ef, 1), 1);
    for ei = 1:size(ef, 1)
        ef_pair = ef(ei, :);
        tp_table = cell(T - 1, 1);
        for t = 1:T - 1
            row_t = cell(1, size(grid, 1));
            for i = 1:size(grid, 1)
                fn = @(j) cal_transfer_prob(t, i, j, ef_pair, grid, cash);
                trans_p = arrayfun(fn, 1:size(grid, 1));
                trans_p = trans_p ./ sum(trans_p);
                row_t{i} = trans_p;
            end
            tp_table{t} = row_t;
        end
        tp_tables{ei} = tp_table;
    end
end
```

$$V(W_i(T)) = \begin{cases} 0 & \text{if } W_i(T) < G \\ 1 & \text{if } W_i(T) \geq G \end{cases} \quad (5)$$

$$V(W_i(t)) = \max_{\mu \in [\mu_{\min}, \mu_{\max}]} \left[\sum_{j=0}^{i_{\max}} V(W_j(t+1)) p(W_j(t+1) | W_j(t), \mu) \right] \quad (7)$$

```
% Create the table of V which is the probability that the investor will attain their g
% wealth, G, or more at the time horizon T, given they have a worth W(t) at time t
% grid: the wealth grid points, tp-table: the table of transition probabilities

% Generate the V table for one of m equally spaced values from mu_min to mu_max
function v_dp = get_Vtable(grid, tp_table, G, T)
    v_dp = zeros(T, size(grid, 1)); % create the initial table with all 0s
    fullfill_idx = grid >= G; % distinguish which wealth grid points have values great
    v_dp(T, fullfill_idx) = 1;
    v_dp(T, ~fullfill_idx) = 0; % assign the corresponding value to v table

    % obtain all V values for this m
    for t = T - 1:-1:1
        for i = 1:size(grid, 1) % rows of grid
            trans_p = tp_table{t}{i};
            v_dp(t, i) = sum(v_dp(t + 1, :) .* trans_p);
        end
    end
end

% Obtain the entire V(W(T)) table and the value of m corresponding to V(Wi(t))
function [best_ms, v_tables] = V(ef, grid, tp_tables, G, T)
    v_tables = cell(size(ef, 1), 1); % Create a cell array, where each cell can contai
    % Obtain the entire V(W(T)) table
    for ei = 1: size(ef, 1) % rows of ef = m = 15
        v_table = get_Vtable(grid, tp_tables{ei}, G, T);
        v_tables{ei} = v_table;
    end

    best_ms = cell(T, size(grid, 1)); % python max( ,key) equivalent?
    % Find the value of m corresponding to the maximum V (W(T))
    % by fixing the year and the wealth grid point
    for t = 1: T
        for i = 1: size(grid, 1)
            best_ef_idx = 0;
            cur_best = -Inf;
            for j = 1: size(ef, 1)
                v = v_tables{j}(t, i);
                if v > cur_best
                    best_ef_idx = j;
                    cur_best = v;
                end
```

```
            end
            best_ms{t, i} = best_ef_idx;
        end
    end
end
```

$$p(W_j(t+1)) = \sum_{i=0}^{i_{\max}} p(W_j(t+1)|W_j(t), \mu_{i+t}) \cdot p(W_j(t)) \quad (8)$$

```
% calculate the table of probability distribution for the investor's wealth at future t
function p_table = calc_wealth_prob(grid, w0_idx, best_ms, tp_tables, T)
    p_table = cell(T, size(grid, 1));
    p_table(1:T, :) = {0}; % When t = 0, the proability of all wealth node are 0
    p_table{1, w0_idx} = 1; % without the wealth node that equals W(0) = 1
    for t = 2: T
        for j = 1: size(grid, 1)
            % create an anonymous function fn to firstly find the best m giving time an
            % and then find the transition probabilities
            fn = @(i) tp_tables{best_ms{t - 1, i}}{t - 1}{i}(j);
            % apply fn to each number of rows of grid,
            % and find the transition probabilities for each wealth grid points
            trans_p = arrayfun(fn, 1: size(grid, 1));
            % calcualte p(Wj(t + 1))
            p_table{t, j} = sum([p_table{t - 1, :}] .* trans_p);
        end
    end
end

% This is a function that integrates all of the above functions
function prob = gen_G_prob_pipeline(w0, T, cash, ef, rho, G)
    [grid, w0_idx] = gen_grid(w0, cash, ef, rho, {-3, 3}, 1);
    tp_tables = get_trans_prob_tables(ef, grid, T, cash);
    [best_ms, v_tables] = V(ef, grid, tp_tables, G, T);
    p_table = calc_wealth_prob(grid, w0_idx, best_ms, tp_tables, T);
    [p_len, p_width] = size(p_table);
    G_idx = sum(grid < G) + 1;
    prob = sum([p_table{p_len, G_idx:p_width}]);
end







% value function for each state
function dp_1 = dp_plot_1(grid, v_tables, T, w0_idx, G_idx, ef)
    dp_1 = figure(3);
    best_action = zeros(size(grid, 1), T-1); % python max( ,key) equivalent?
```

```
    % Find the value of m corresponding to the maximum V (W(T))
    % by fixing the year and the wealth grid point
    for t = 1: T-1
        for i = 1: size(grid, 1)
            best_ef_idx = 0;
            cur_best = -Inf;
            for j = 1: size(ef, 1)
                v = v_tables{j}(t, i);
                if v > cur_best
                    best_ef_idx = j;
                    cur_best = v;
                end
            end
            best_action(i, t) =  best_ef_idx;
        end
    end

    %imagesc(best_action)
    surf(best_action)

    yticks([1, w0_idx, G_idx, length(grid)]);
    s1 = int2str(grid(1));
    s2 = int2str(grid(w0_idx));
    s3 = int2str(grid(G_idx));
    s4 = int2str(grid(end));
    yticklabels({s1,s2,s3,s4})
    xlabel("t in Years");
    ylabel("Wealth in Portillos (W(t))");
    title("State - Value Mapping");

    %set(gca, 'XDir','reverse');
    colorbar
    rotate3d


    end



% decision making part
function dp_2 = dp_plot_2(ef, total_itr, T, w0_idx, G, best_ms, tp_tables, grid)
    %% suggestions
    dp_2 = [];
    [mus_qln, trace] = suggestion(T, w0_idx, best_ms, tp_tables);

    mus = ef(:,1);
    a_1 = figure(1);

    hold on
    yyaxis left
```

```
    plot(1:T, grid(trace));
    xlabel("t in monthes");
    ylabel("Wealth in portfolio, W(t)");

    yyaxis right
    scatter(2:T,mus(mus_qln));
    xlabel("t in monthes");
    ylabel("Wealth in portfolio, W(t)");

    title(" Dynamics of Wealth in Portfolio - Illustruction (w0=100)")
    hold off




    a_2 = figure(2);
    hold on

    trace_matrix = zeros(total_itr, T);
    mus_qln_matrix = zeros(total_itr, T-1);

    for itr = 1:total_itr
        [mus_qln, trace] = suggestion(T, w0_idx, best_ms, tp_tables);
        trace_matrix(itr,:) = grid(trace)';

        c = mus(mus_qln);
        mus_qln_matrix(itr,:) = c';
    end


    yyaxis left
    for itr = 1:total_itr
        trace = trace_matrix(itr,:);
        trace = trace';

        mus = ef(:,1);

        if trace(end) >= G
            plot(1:T, trace, '-o', 'Color', [min(0.8500+trace(end)/400, 1) 0.3250 0.098
        else
            plot(1:T, trace, '-x', 'Color', [0 0.4470 min(0.7410+trace(end)/400, 1)]);
        end
    end
    xlabel("t in monthes");
    ylabel("Expected return \mu");
    yyaxis right
    for itr = 1:total_itr
        trace = trace_matrix(itr,:);
        c = mus_qln_matrix(itr,:);
```

```
        c = c';
        if trace(end) >= G
            scatter(2:T, c, 10, c, 'filled');
        else
            scatter(2:T, c, 10, c, "x");
        end
    end
    xlabel("t in monthes");
    ylabel("Expected return \mu");
    title("Dynamics of Wealth in Portfolio (w0=100)");
    hold off


end



function [mus_qln, trace] = suggestion(T, w0_idx, best_ms, tp_tables)
    mus_qln = [];
    trace = [];
    cs = w0_idx;
    t = 1;

    while(t<T)      % do not update for t = T
        trace = [trace cs];
        chosen_action = best_ms{t, cs};
        mus_qln = [mus_qln, chosen_action];
        %% state simulation
        seed_state = rand;
        cdf = cumsum(tp_tables{chosen_action}{t}{cs});
        ns = find(cdf >= seed_state);
        ns = ns(1);
        cs = ns;
        t = t + 1;
    end
    trace = [trace ns];
end
```