Table of Contents


```
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%
% Computational Problem Set, Enviro I, Problem 2
% Zachary Kuloszewski
%
% Last Edit Date: Nov 7, 2022
%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
```

set directories

```
clear; clc;

cd '/Users/zachkuloszewski/Dropbox/My Mac (Zachs-MBP.lan)/Documents/';
cd 'GitHub/phd_psets/year2/environmental';
addpath(genpath('figures'));
addpath(genpath('functions'));
```

Problem 2

Interpolating the state space

part 2a - set up state space, parameters

```
N     = 501;
S_tot = 1000;
r     = 0.05;

delta = 1/(1+r);
S     = linspace(0,S_tot,N);

% set up action space
A     = linspace(0,sqrt(S_tot),N);
A     = A.^2;
nA    = numel(A);

% define utility of extraction
```

```
% pick utility function of interest
u_fun_flag = 2; % choose 1 or 2

if u_fun_flag == 1
    u = @(x) 2*x.^0.5;
elseif u_fun_flag == 2
    u = @(x) 5*x-0.05*x.^2;
end

% define flow utility matrix
U = u(repmat(A,nA,1));

% def -inf upper triangular matrix
infs    = repmat(-Inf, N);
flow_UT = triu(infs,1);

% sum to get final flow utility matrix
U   = U + flow_UT;
```

part 2b - define interpolated transition matrix

```
% identify state in next period
% init matrix w same dimension as flow utility
transition = nan(N,nA);

for i=1:N % iter through rows
    for j=1:nA % iter through columns
        if j > i % if extracting more than full stock
            transition(i,j) = 1;
        else
            transition(i,j) = i-j+1;
        end
    end
end

% state transition matrix
T = nan(N,N*nA);
rows = 1:501;

for k=1:nA

    inds = zeros(N,2);
    wts  = zeros(N,2);

    %here we need to find the weights
    for i=1:N
        [inds(i,:),wts(i,:)] = interp_actions(S(i),k,S,A);
    end

    inds_stack = [inds(:,1); inds(:,2)];
    wts_stack  = [wts(:,1) ; wts(:,2)];
    rows_stack = [rows'; rows'];
```

```
    T(:,1+(k-1)*N:k*N) = sparse(rows_stack(wts_stack > 0),inds_stack(wts_stack
 > 0), ...
        wts_stack(wts_stack > 0),N,nA);
end

T = sparse(T);
```

part 2c - solve the model

```
% init parameters
error     = 1e12;
error_tol = 1e-8;

% init value and optim choice
V_hist = nan(N,1000);
C_hist = nan(N,1000);

% iteration counter
n_iter = 0;

V     = repelem(0,N)';

Vnext = nan(N,nA);

while error > error_tol

    % count number iterations
    n_iter = n_iter + 1;
    Vold  = V;

    Vnext = zeros(N,nA);
    for k=1:nA %looping thru action space
        Vnext(:,k) = T(:,1+(k-1)*N:k*N)*V;
    end

    % grab optimized value and action column
    [V, C] = max(U + delta .* Vnext,[],2);

    % store values and choices
    V_hist(:,n_iter) = V;
    C_hist(:,n_iter) = C;

    error = max(abs(V-Vold));

end
```

part 2d - simulate the model

```
% find optimal transition matrix
Topt = zeros(N,nA);

for i=1:N
    Topt(i, C_hist(i,n_iter)) = 1;
```

```
end

% init parameters for simulation
t      = 80;
st     = S_tot;

% init storage for output
V_hist = nan(t,1);
C_hist = nan(t,1);
S_hist = nan(t,1);

for i=1:80

    st_ind  = find(S <= st,1,'last'); % find lower bound state index
    act_ind = find(Topt(st_ind,:) > 0);

    if st_ind == N
        action = A(act_ind);
    else
        wt = (st - S(st_ind))/(S(st_ind+1) - S(st_ind));
        action = wt * A(act_ind) + (1-wt) * A(act_ind+1);
    end

    st = st - action; % update stock

    S_hist(i) = st;
    C_hist(i) = action;
    V_hist(i) = u(action);

end
```

part 2e - plots

```
figure;
plot([V_hist, C_hist]);
legend("Price", "Extraction", Location="northeast");
legend box off
xlabel("Period")

if u_fun_flag == 1
    u_txt = "2\sqrt{y}$";
elseif u_fun_flag == 2
    u_txt = "5y - 0.05y^2$";
end

title(strcat("Compressed State Space Simulations, $u=", u_txt), ...
    'Interpreter','latex');

saveas(gcf, ['figures/part2e_u' num2str(u_fun_flag) '.png']);
```

Published with MATLAB® R2022b