```
% Parameters: alpha, beta, sigma, q
% Platform: fixed fee T, ad valorem tax t
% alpha: price-quality tradeoff in utility
% beta: price-quality tradeoff in platform choice
% sigma: technology parameter about how tightly the platform can make
 the
% choice

% Parameters

qstart      = 1;          % enter fixed value of q1, ignore the
 normalized q2
alphastart  = .5;
sigmastart  = 1;
fees        = [.3; .1];         % ordered as a 2x1 vector of T first,
 then t.
cstart      = [.5; 0];          % ordered as a 2x1 vector of c1 first,
 then the normalized c2.
```

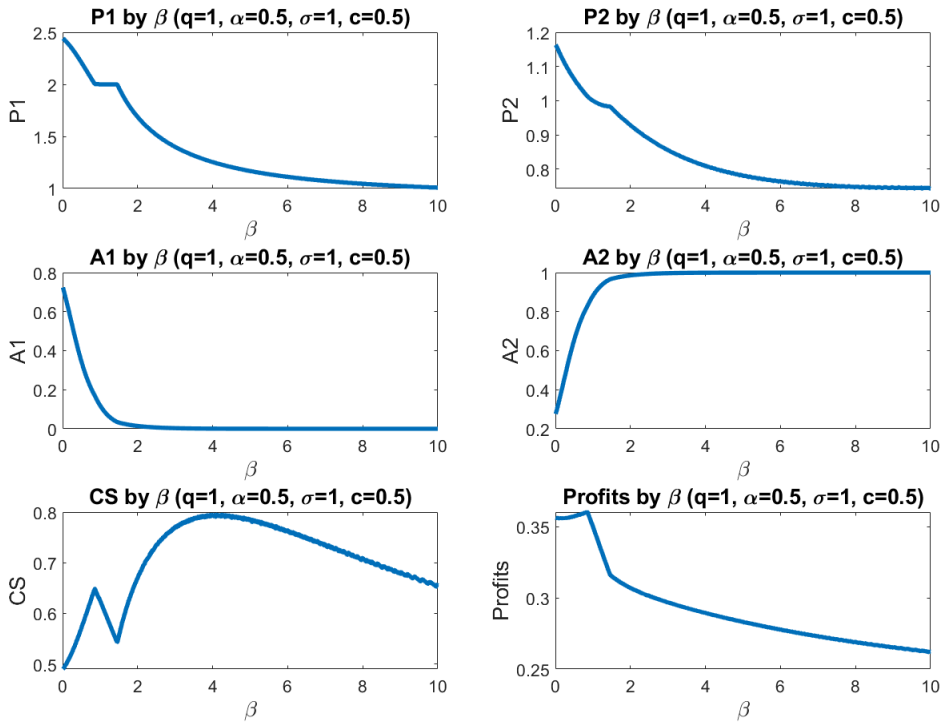
Part 1 - Recreate the six figures of how various values differ across

values of beta

```
% Beta evaluation points for six figures:
numbeta = 1000;
numvec = linspace(1/100,10,numbeta)';
% Evaluate 1,000 points evenly spaced from 1/100 to 10.
% Below I refer to the number of beta points as 'numbeta' and the
% evaluation points themselves 'numvec'


% create placeholder vectors of zeros where you will fill in the
 resulting
% value for each beta you're evaluating.
p1holder = zeros(numbeta,1);    % P1 placeholder
p2holder = zeros(numbeta,1);    % P2 placeholder
objholder = zeros(numbeta,1);  % Objective function (satisfy
 optimality) placeholder
csholder = zeros(numbeta,1);    % Consumer Surplus placeholder
piholder = zeros(numbeta,1);    % Profit placeholder

% Loop over each value of beta
for ii=1:numbeta

vectry = [qstart;alphastart;numvec(ii);sigmastart;fees]; % arrange
 the vector of parameters you're evaluating at: q/alpha/beta/sigma/T/
t <-- MUST be in th is order because subsequent functions assume this
 order.
```

```
h = @(x) func_foc_costs(x,vectry,cstart); % define objective and
 performance to solve FOC=0 with given parameters and compute
 corresponding Consumer Surplus and profit associated with it. x is
 pair of prices, vectry is the params considering, cstart is costs
tempp = func_find_prices(1000,vectry,cstart); % function solves for
 equilibrium prices [p1,p2] given parameters

[tempobj,tempcs,temppi] = h(tempp); % given the equilibrium prices
 (tempp), compute the outputs to store and compare across betas.

p1holder(ii)    = tempp(1);
p2holder(ii)    = tempp(2);
objholder(ii)   = tempobj;
csholder(ii)    = tempcs;
piholder(ii)    = temppi;
end

% Solve for a1
a = exp(qstart - numvec.*p1holder)./(exp(qstart - numvec.*p1holder) +
 exp(numvec.*p2holder));


% Example of how to plot to look like in assignment (for p_1)
f = figure(1);
f.Position = [100,100,1400,1000];
subplot(3,2,1)
plot(numvec,p1holder,'LineWidth',4)
title(['\fontsize{20}P1 by \beta (q=1, \alpha=0.5, \sigma=1, c=0.5)'])
xlabel(['\fontsize{20}\beta'])
ylabel(['\fontsize{20}P1'])
set(gca,'fontsize',16)


subplot(3,2,2)
plot(numvec,p2holder,'LineWidth',4)
title(['\fontsize{20}P2 by \beta (q=1, \alpha=0.5, \sigma=1, c=0.5)'])
xlabel(['\fontsize{20}\beta'])
ylabel(['\fontsize{20}P2'])
set(gca,'fontsize',16)


subplot(3,2,3)
plot(numvec,a,'LineWidth',4)
title(['\fontsize{20}A1 by \beta (q=1, \alpha=0.5, \sigma=1, c=0.5)'])
xlabel(['\fontsize{20}\beta'])
ylabel(['\fontsize{20}A1'])
set(gca,'fontsize',16)


subplot(3,2,4)
plot(numvec,1-a,'LineWidth',4)
title(['\fontsize{20}A2 by \beta (q=1, \alpha=0.5, \sigma=1, c=0.5)'])
xlabel(['\fontsize{20}\beta'])
ylabel(['\fontsize{20}A2'])
```

```
set(gca,'fontsize',16)


subplot(3,2,5)
plot(numvec,csholder,'LineWidth',4)
title(['\fontsize{20}CS by \beta (q=1, \alpha=0.5, \sigma=1, c=0.5)'])
xlabel(['\fontsize{20}\beta'])
ylabel(['\fontsize{20}CS'])
set(gca,'fontsize',16)

subplot(3,2,6)
plot(numvec,piholder,'LineWidth',4)
title(['\fontsize{20}Profits by \beta (q=1, \alpha=0.5, \sigma=1,
 c=0.5)'])
xlabel(['\fontsize{20}\beta'])
ylabel(['\fontsize{20}Profits'])
set(gca,'fontsize',16)
saveas(gcf,'p1_by_beta.png')
```



Part 2: Recreate four subfigures of optimal beta (maximizes CS).

Vary q, alpha, sigma, c and find the optimal beta for each. Hold the other paramters fixed at their standard values

```
% This takes forever, so we are gonna use a parpool
```

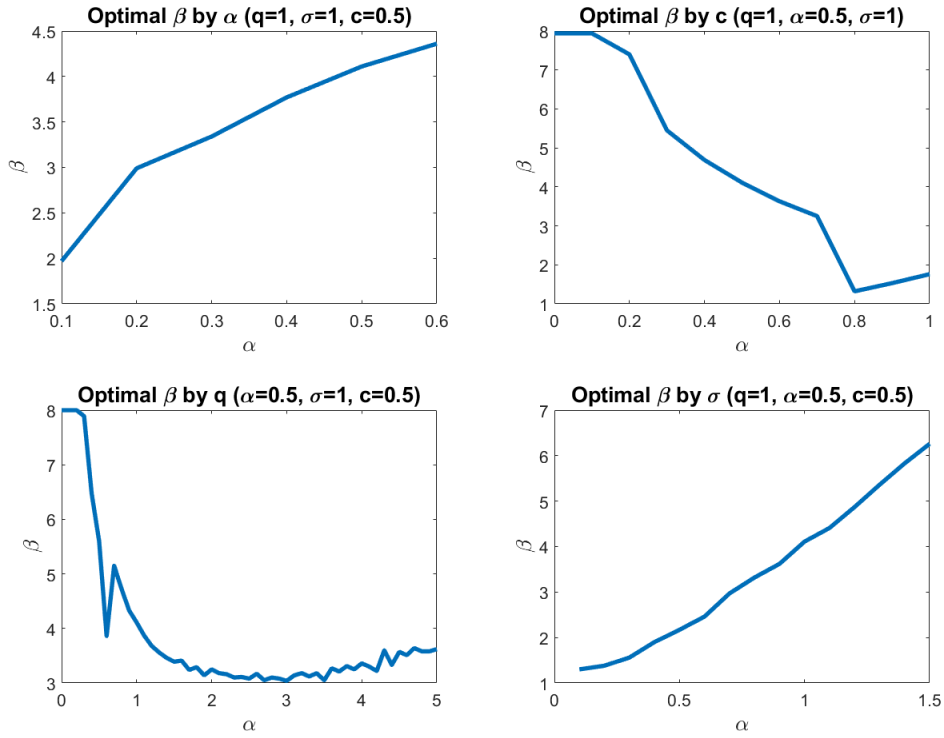
```
pool = parpool(10);



% This time, evaluate 800 points of beta, evenly spaced from 1/100 to
 8.
numbeta2 = 800;
numvec2 = linspace(1/100,8,numbeta2)';


% alpha - evaluate from .1 to .6 with step size of .1.
alphavec = .1:.1:.6;
% for each pair of evaluation points (alpha,beta), along with all
 other params standard, save the CS here.
alphastore  = zeros(length(alphavec),numbeta2);
n1 = size(alphavec,2);

% You'll want to create a double for loop to do this.
parfor i=1:n1
    for j = 1:numbeta2
        alpha2 = alphavec(i);
        beta2 = numvec2(j,1);

        vectry = [qstart;alpha2;beta2;sigmastart;fees]; % arrange the
 vector of parameters you're evaluating at: q/alpha/beta/sigma/T/t
 <-- MUST be in th is order because subsequent functions assume this
 order.

        h = @(x) func_foc_costs(x,vectry,cstart); % define objective
 and performance to solve FOC=0 with given parameters and compute
 corresponding Consumer Surplus and profit associated with it. x is
 pair of prices, vectry is the params considering, cstart is costs
        tempp = func_find_prices(1000,vectry,cstart); % function
 solves for equilibrium prices [p1,p2] given parameters

        [tempobj,tempcs,temppi] = h(tempp); % given the equilibrium
 prices (tempp), compute the outputs to store and compare across
 betas.

        alphastore(i,j) = tempcs;

    end

end
% Then, for each value of alpha, find the optimal beta (in terms of
% maximizing CS) and save that to this next object.
[m0,i] = max(alphastore,[],2);
alphaoptbeta =numvec(i);


% c - vary c from 0 to 1 with step size of .1. Then follow similar
 approach
% as alpha
cvec = 0:.1:1;
```

```
cstore = zeros(length(cvec),numbeta2);

% You'll want to create a double for loop to do this.
n = size(cvec,2);
parfor i=1:n
    for j = 1:numbeta2
        c2 = cvec(i);
        beta2 = numvec2(j,1);

        vectry = [qstart;alphastart;beta2;sigmastart;fees]; % arrange
 the vector of parameters you're evaluating at: q/alpha/beta/sigma/T/
t <-- MUST be in th is order because subsequent functions assume this
 order.

        h = @(x) func_foc_costs(x,vectry,[c2; 0]); % define objective
 and performance to solve FOC=0 with given parameters and compute
 corresponding Consumer Surplus and profit associated with it. x is
 pair of prices, vectry is the params considering, cstart is costs
        tempp = func_find_prices(1000,vectry,[c2; 0]); % function
 solves for equilibrium prices [p1,p2] given parameters

        [tempobj,tempcs,temppi] = h(tempp); % given the equilibrium
 prices (tempp), compute the outputs to store and compare across
 betas.

        cstore(i,j) = tempcs;


    end
end
[m1,i] = max(cstore,[],2);
coptbeta = numvec(i);

% q - vary from 0 to 5 with step size of .1. Then follow similar
 approach
% as above. Note this is a little slower as it is more evaluation
 points.
qvec = 0:.1:5;
qcstore     = zeros(length(qvec),numbeta2);
n = size(qvec,2);
parfor i=1:n
    for j = 1:numbeta2
        qstart2 = qvec(i);
        beta2 = numvec2(j,1);

        vectry = [qstart2;alphastart;beta2;sigmastart;fees]; % arrange
 the vector of parameters you're evaluating at: q/alpha/beta/sigma/T/
t <-- MUST be in th is order because subsequent functions assume this
 order.

        h = @(x) func_foc_costs(x,vectry,cstart); % define objective
 and performance to solve FOC=0 with given parameters and compute
 corresponding Consumer Surplus and profit associated with it. x is
 pair of prices, vectry is the params considering, cstart is costs
```

```
        tempp = func_find_prices(1000,vectry,cstart); % function
 solves for equilibrium prices [p1,p2] given parameters

        [tempobj,tempcs,temppi] = h(tempp); % given the equilibrium
 prices (tempp), compute the outputs to store and compare across
 betas.

        qcstore(i,j) = tempcs;
    end
end

[m2,i] = max(qcstore,[],2);
qcoptbeta   = numvec(i);


% sigma - vary from .1 to 1.5 with step size of .1. Then follow
 similar
% approach as above.
sigmavec = .1:.1:1.5;
sigmastore  = zeros(length(sigmavec),numbeta2);
n = size(sigmavec,2);
parfor i=1:n
    for j = 1:numbeta2
        sigmastart2 = sigmavec(i);
        beta2 = numvec2(j,1);

        vectry = [qstart;alphastart;beta2;sigmastart2;fees]; % arrange
 the vector of parameters you're evaluating at: q/alpha/beta/sigma/T/
t <-- MUST be in th is order because subsequent functions assume this
 order.

        h = @(x) func_foc_costs(x,vectry,cstart); % define objective
 and performance to solve FOC=0 with given parameters and compute
 corresponding Consumer Surplus and profit associated with it. x is
 pair of prices, vectry is the params considering, cstart is costs
        tempp = func_find_prices(1000,vectry,cstart); % function
 solves for equilibrium prices [p1,p2] given parameters

        [tempobj,tempcs,temppi] = h(tempp); % given the equilibrium
 prices (tempp), compute the outputs to store and compare across
 betas.

        sigmastore(i,j) = tempcs;
    end
end

delete(pool);

[m3,i] = max(sigmastore,[],2);
sigmaoptbeta = numvec(i);


% Plotting
f = figure(2);
```

```
f.Position = [100,100,1400,1000];
subplot(2,2,1)
plot(alphavec,alphaoptbeta,'LineWidth',4)
title(['\fontsize{20}Optimal \beta by \alpha (q=1, \sigma=1, c=0.5)'])
ylabel(['\fontsize{20}\beta'])
xlabel(['\fontsize{20}\alpha'])
set(gca,'fontsize',16)

% Plotting
subplot(2,2,2)
plot(cvec,coptbeta,'LineWidth',4)
title(['\fontsize{20}Optimal \beta by c (q=1, \alpha=0.5, \sigma=1)'])
ylabel(['\fontsize{20}\beta'])
xlabel(['\fontsize{20}\alpha'])
set(gca,'fontsize',16)

% Plotting

subplot(2,2,3)
plot(qvec,qcoptbeta,'LineWidth',4)
title(['\fontsize{20}Optimal \beta by q (\alpha=0.5, \sigma=1,
 c=0.5)'])
ylabel(['\fontsize{20}\beta'])
xlabel(['\fontsize{20}\alpha'])
set(gca,'fontsize',16)

% Plotting
subplot(2,2,4)
plot(sigmavec,sigmaoptbeta,'LineWidth',4)
title(['\fontsize{20}Optimal \beta by \sigma (q=1, \alpha=0.5,
 c=0.5)'])
ylabel(['\fontsize{20}\beta'])
xlabel(['\fontsize{20}\alpha'])
set(gca,'fontsize',16)
saveas(gcf,'p2_by_beta.png')
```

```
Starting parallel pool (parpool) using the 'local' profile ...
Connected to the parallel pool (number of workers: 10).
Parallel pool using the 'local' profile is shutting down.
```



Published with MATLAB® R2020b