Table of Contents

Define and simulate model

$SV = R - (E * k)$

Set up data to match experiment

```
reward = repmat(2:4, 1, 8)';
effort = repelem(2:5, 6)';

data = table(reward, effort);
```

Simulate choices using known discount (k) and beta parameters

```
% Fix parameters for simulating data
k_param = 0.5;
beta_param = 1;
% Define model formula
SV = reward - (effort .* k_param);
```

$SV = reward - (effort * 0.5)$

```
data.SV = SV
```

data = 24×3 table

	reward	effort	SV
1	2	2	1
2	3	2	2
3	4	2	3

	reward	effort	SV
4	2	2	1
5	3	2	2
6	4	2	3
7	2	3	0.5000
8	3	3	1.5000
9	4	3	2.5000
10	2	3	0.5000
11	3	3	1.5000
12	4	3	2.5000
13	2	4	0
14	3	4	1
15	4	4	2
16	2	4	0
17	3	4	1
18	4	4	2
19	2	5	-0.5000
20	3	5	0.5000
21	4	5	1.5000
22	2	5	-0.5000
23	3	5	0.5000
24	4	5	1.5000

```
% Calculate softmax
% (Predicted probabilities of choosing "work" option)
prob = exp(SV .* beta_param) ./ (exp(SV .* beta_param) + exp(1 .* beta_param));
```

$$prob = \frac{e^{SV*\beta}}{e^{SV*\beta} + e^{1*\beta}}$$

```
% Simulate choices from probabilities
random_num = rand(length(prob), 1); % list of 0-1 random numbers

choices = prob > random_num;

data.prob = prob;
data.random_num = random_num;
data.choices = choices;
data
```

data = 24×6 table

	reward	effort	SV	prob	random_num	choices
1	2	2	1	0.5000	0.7094	0
2	3	2	2	0.7311	0.7547	0
3	4	2	3	0.8808	0.2760	1
4	2	2	1	0.5000	0.6797	0
5	3	2	2	0.7311	0.6551	1
6	4	2	3	0.8808	0.1626	1
7	2	3	0.5000	0.3775	0.1190	1
8	3	3	1.5000	0.6225	0.4984	1
9	4	3	2.5000	0.8176	0.9597	0
10	2	3	0.5000	0.3775	0.3404	1
11	3	3	1.5000	0.6225	0.5853	1
12	4	3	2.5000	0.8176	0.2238	1
13	2	4	0	0.2689	0.7513	0
14	3	4	1	0.5000	0.2551	1
15	4	4	2	0.7311	0.5060	1
16	2	4	0	0.2689	0.6991	0
17	3	4	1	0.5000	0.8909	0
18	4	4	2	0.7311	0.9593	0
19	2	5	-0.5000	0.1824	0.5472	0
20	3	5	0.5000	0.3775	0.1386	1
21	4	5	1.5000	0.6225	0.1493	1
22	2	5	-0.5000	0.1824	0.2575	0
23	3	5	0.5000	0.3775	0.8407	0
24	4	5	1.5000	0.6225	0.2543	1

Turn the above code into a function for use later.

```
function data = simulateData(true_params)
reward = repmat(2:4, 1, 8)';
effort = repelem(2:5, 6)';
k_param_true = true_params(1);
beta_param_true = true_params(2);

% "Work" option subjective value
SV = reward - (effort .* k_param_true);

% Calculate probabilities of choosing work option
% (calculates probability for each row of matrix (i.e., trial) based on SV and baseline values)
```

```
    prob = exp(SV .* beta_param_true) ./ (exp(SV .* beta_param_true) + exp(1 .* beta_param_true));

    choices = prob > rand(length(prob), 1);

    data = table(choices, reward, effort);
    data.k_param = repmat(k_param_true, length(choices), 1);
    data.beta_param = repmat(beta_param_true, length(choices), 1);

    end
```

Fit model using MLE

Using the simulated data above, fit the model using maximum likelihood estimation (MLE).

Can be used to check if "true" parameters can be recovered by the model.

```
% Will use fmincon to fit model
%
% x = fmincon(fun,x0,[],[],[],[],lb,ub)
%
% x = solution (e.g., fitted params) to MLE fitting
% fun = function to minimize
% x0 = initial starting points
% lb = lower bounds of parameters being estimated
% ub = upper bounds of parameters being estimated
%
% fitted_params = fmincon(model_function, starting_values,...,lower_bounds,upper_bounds
recipient = repelem(["food", "climate"], 12)';
data.recipient = recipient;
```

First, create function that `fmincon` will try to minimize.

```
params = [k_param, beta_param];

negLL = fitModel(data, params); %(defined in "Functions" section at end of script)
fprintf('Minimised log-likelihood: %f', negLL);
```

```
Minimised log-likelihood: 14.504284
```

```
    function negLL = fitModel(data, params)
    % Fits parameters to model and calculates softmax and negative log-likelihood

    % Get reward, effort, and choices from data
    reward = data.reward;
    effort = data.effort;
    choices = data.choices;

    % Assign parameter values
    k_param = params(1);
    beta_param = params(2);

    % Compute subjective value of "work" option
    SV = reward - (effort .* k_param);
```

```
    % Calculate probabilities of choosing work option
    prob = exp(SV .* beta_param) ./ (exp(SV .* beta_param) + exp(1 .* beta_param));

    % Probability of choosing "rest" option
    prob(~choices) = 1 - prob(~choices);

    % Convert probabilities to log-likelihoods
    LL = log(prob);
    negLL = -sum(LL); %sum LL together and flip sign to make values positive (for fmin minimizing functi

    end
```

Then use `fmincon` to find parameters that minimize function—i.e., minimize the (negative) log-likelihood.

The `fmincon` function allows you to set lower/upper bounds for the parameters it estimates.

```
% Set parameter bounds and initial values
lb = [0, 0];
ub = [2, 5];
initial_params = rand(1, 2);
fminoptions = optimoptions('fmincon','Display','off');
```

```
% Define input function for fmin
inputfun = @(params) fitModel(data, params);

fitModel(data, params);
```

```
% Fit model to minimizing function
[fit_params, negLL] = fmincon(inputfun, ...
                              initial_params, [], [], [], [], ...
                              lb, ub, [], ...
                              fminoptions);
```

```
% Fitted parameters and negative log-likelihood (negLL)
fprintf('k_param: %f\nbeta_param: %f', fit_params);
```

```
k_param: 0.486073
beta_param: 0.920849
```

Note: The fitted parameters may not exactly match the "true" parameters we set above.

With so few trials, the randomly simulated data may not perfectly reflect the "true" parameters.

One solution is to increase the number of simulated trials. Another solution is to simulate a wide range of "true" parameters many times to see how well the model can recover these parameters.

Simulate range of parameters to check model robustness

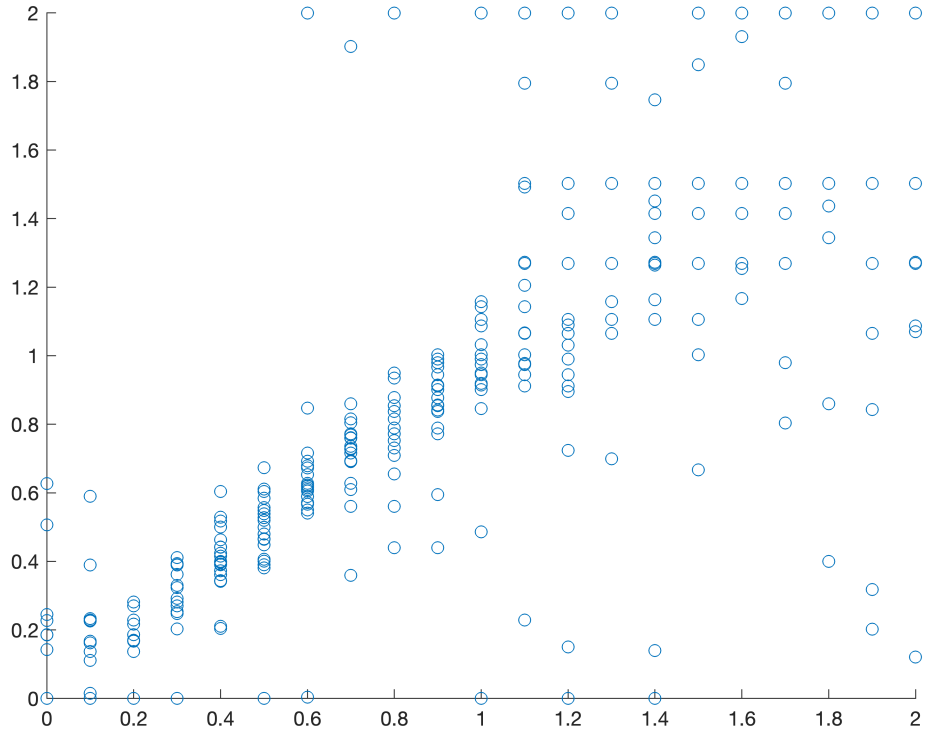
Set a range of known k and beta values and simulate "participants" with these values.

Simulate using a set range of parameter values

```
% Generate range of k and beta values
sim_k_values = 0 : 0.1 : 2; % min : increment : max
sim_beta_values =  0 : 0.25 : 5; % min : increment : max

% Create all combinations of these parameters
sim_params_allcombo = combvec(sim_k_values, sim_beta_values)';

% Total number of "participants" to simulate,
% each with unique combination of k and beta "true" values
nsims = length(sim_params_allcombo);

fit_params = zeros(nsims, 2); %(preallocate array to improve speed)

% Loop through each combination of parameters
for isim = 1:length(sim_params_allcombo)
    true_k = sim_params_allcombo(isim, 1);
    true_beta = sim_params_allcombo(isim, 2);

    % Generate simulated choices based on this combination of parameters
    sim_data = simulateData([true_k, true_beta]);

    % Fit the simulated data using MLE
    inputfun = @(params) fitModel(sim_data, params);
    initial_params = rand(1, 2);
    fit_params(isim, :) = fmincon(inputfun, ...
                                  initial_params, [], [], [], [], ...
                                  lb, ub, [], ...
                                  fminoptions);
end
```

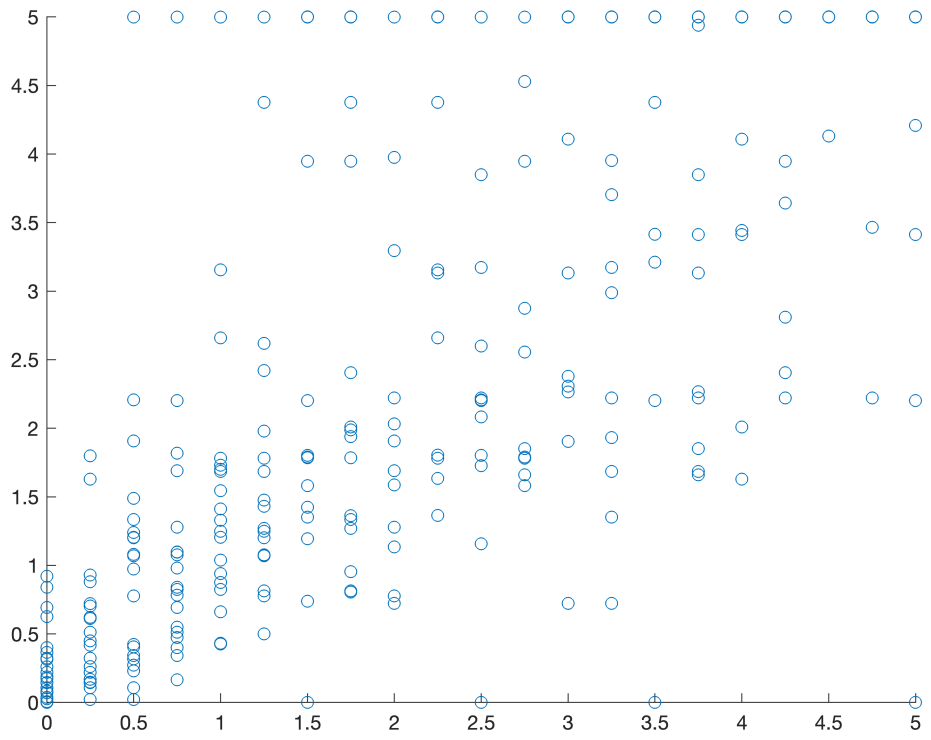
Fitted k parameter vs. "True" k parameter

```
scatter(sim_params_allcombo(:,1), fit_params(:,1));
```



Fitted beta parameter vs. "True" beta parameter

```
scatter(sim_params_allcombo(:,2), fit_params(:,2));
```



Simulate using uniform distribution of parameter values

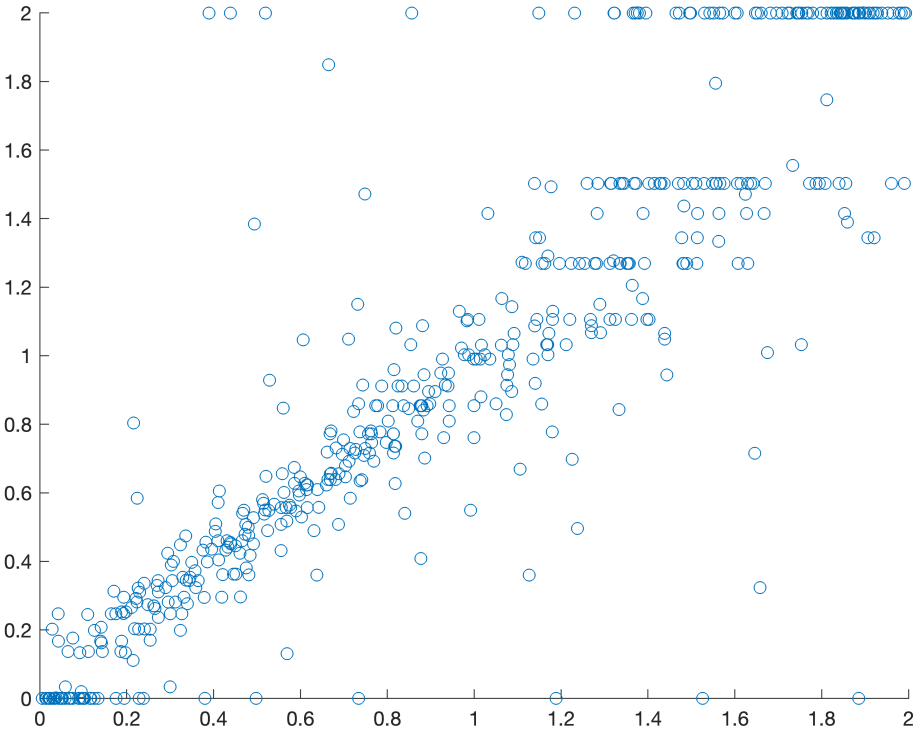
Rather than using predetermined k and beta values, we can generate a large number of values from a random distribution.

```
clear sim_data fit_params

nsims = 500; %number of "participants" to simulate

% Generate "true" k and beta values from random distribution
sim_k_values = rand(1, nsims) * ub(1);
sim_beta_values =  rand(1, nsims) * ub(2);

fit_params = zeros(nsims, 2); %(preallocate array to improve speed)

% Loop through each combination of parameters
for isim = 1:nsims
    true_k = sim_k_values(isim);
    true_beta = sim_beta_values(isim);

    % Generate simulated choices based on this combination of parameters
    sim_data = simulateData([true_k, true_beta]);

    % Fit the simulated data using MLE
    inputfun = @(params) fitModel(sim_data, params);
    initial_params = rand(1, 2);
```

```
    fit_params(isim, :) = fmincon(inputfun, ...
                                  initial_params, [], [], [], [], ...
                                  lb, ub, [], ...
                                  fminoptions);
end
```

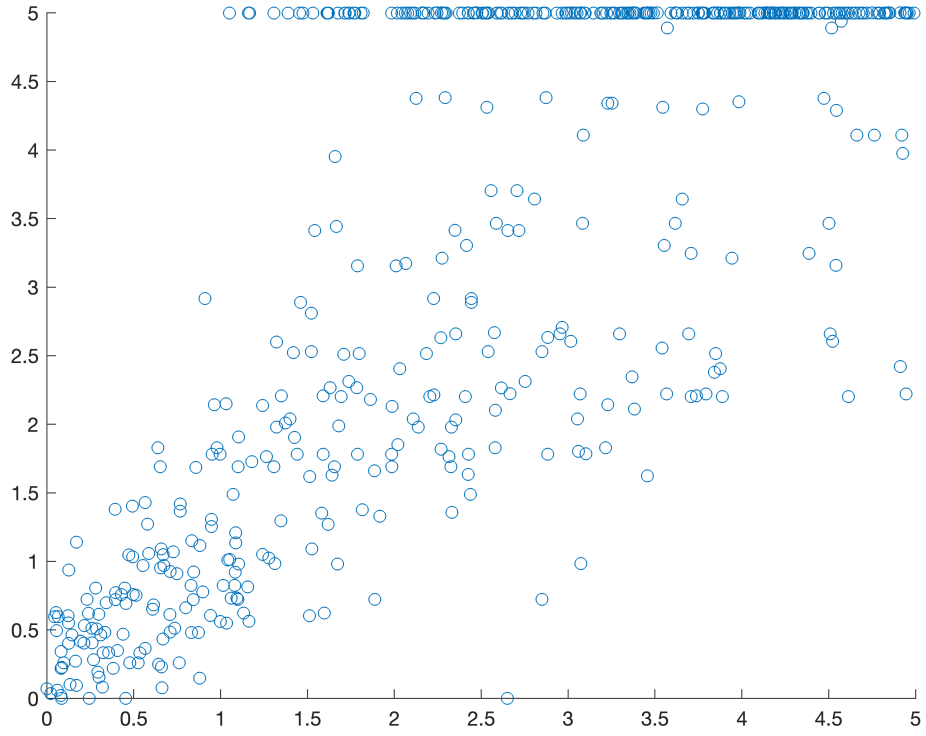
Fitted k parameter vs. "True" k parameter

```
scatter(sim_k_values', fit_params(:,1));
```



Fitted beta parameter vs. "True" beta parameter

```
scatter(sim_beta_values', fit_params(:,2));
```



Note: Sometimes the fitted parameters are at the bounds we set.

There can be various reasons for this, e.g.,

- Bounds that are too small/too large (bounds may be set based on theory, or sometimes to improve mathematical stability)
- Data are noisy/not enough data
- Model doesn't accurately reflect how data were generated

There are several ways to address this, e.g.,

- Use hierarchical fitting
- Reduce noise in data (e.g., collect more trials/participants)
- Try different bounds
- Try a different model

Fitting different types of models

Simulate data using a parabolic model. Will use this data to fit different types of models (linear, parabolic, hyperbolic) to see which type of model best describes the data (i.e., the one that best minimises the log-likelihood).

```
model_ids = {
    'one_k_one_beta_linear', ...
```

```
    'one_k_one_beta_para', ...
    'one_k_one_beta_hyper'
};


nsims = 100;
sim_data_para = cell(1, nsims);

true_k_params = rand(1, nsims) * 2;
true_beta_params = rand(1, nsims) * 3;

for i = 1:nsims
    sim_data_para{i} = simulateData([true_k_params(i),...
                                     true_beta_params(i)],...
                                     'one_k_one_beta_para');
end
```

Fit a linear model

$SV = R - (E * k)$

```
fit_linear = zeros(nsims, 2);
negLL_linear = zeros(nsims, 1);

model_id = 'one_k_one_beta_linear';
lb = [0, 0];
ub = [2, 5];

for isubj = 1:length(sim_data_para)

    subj_data = sim_data_para{isubj};

    % Fit the data using MLE
    inputfun = @(params) fitModel(subj_data, params, model_id);
    initial_params = rand(1, 2);
    [fit_linear(isubj, :), negLL_linear(isubj)] = fmincon(inputfun, ...
                                                          initial_params, [], [], [],
                                                          lb, ub, [], ...
                                                          fminoptions);
end
```

```
fprintf('Minimised log-likelihood: %f', sum(negLL_linear));
```

```
Minimised log-likelihood: 353.554734
```

Fitted k parameter vs. "True" k parameter

```
% scatter(true_k_params', fit_linear(:,1));
```

Fitted beta parameter vs. "True" beta parameter

```
% scatter(true_beta_params', fit_linear(:,2));
```

Fit a parabolic model

$SV = R - (E^2 * k)$

```
fit_para = zeros(nsims, 2);
negLL_para = zeros(nsims, 1);

model_id = 'one_k_one_beta_para';
lb = [0, 0];
ub = [1, 5];

for isubj = 1:length(sim_data_para)

    subj_data = sim_data_para{isubj};

    % Fit the data using MLE
    inputfun = @(params) fitModel(subj_data, params, model_id);
    initial_params = rand(1, 2);
    [fit_para(isubj, :), negLL_para(isubj)] = fmincon(inputfun, ...
                                                      initial_params, [], [], [], [],
                                                      lb, ub, [], ...
                                                      fminoptions);
end
```

```
fprintf('Minimised log-likelihood: %f', sum(negLL_para));
```

```
Minimised log-likelihood: 292.261278
```

Fitted k parameter vs. "True" k parameter

```
% scatter(true_k_params', fit_para(:,1));
```

Fitted beta parameter vs. "True" beta parameter

```
% scatter(true_beta_params', fit_para(:,2));
```

Fit a hyperbolic model

$$\frac{R}{1 + (E * k)}$$

```
fit_hyper = zeros(nsims, 2);
negLL_hyper = zeros(nsims, 1);
```

```
model_id = 'one_k_one_beta_hyper';
lb = [0, 0];
ub = [2, 5];

for isubj = 1:length(sim_data_para)

    subj_data = sim_data_para{isubj};

    % Fit the data using MLE
    inputfun = @(params) fitModel(subj_data, params, model_id);
    initial_params = rand(1, 2);
    [fit_hyper(isubj, :), negLL_hyper(isubj)] = fmincon(inputfun, ...
                                                        initial_params, [], [], [], []
                                                        lb, ub, [], ...
                                                        fminoptions);
end
```

```
fprintf('Minimised log-likelihood: %f', sum(negLL_hyper));
```

```
Minimised log-likelihood: 482.808489
```

Fitted k parameter vs. "True" k parameter

```
% scatter(true_k_params', fit_hyper(:,1));
```

Fitted beta parameter vs. "True" beta parameter

```
% scatter(true_beta_params', fit_hyper(:,2));
```

Calculate AIC/BIC to compare models

Compare Log-Likelihoods

```
negLL_table = table(sum(negLL_linear), sum(negLL_para), sum(negLL_hyper), 'VariableNam
negLL_table
```

```
negLL_table = 1×3 table
```

	Linear	Parabolic	Hyperbolic
1	353.5547	292.2613	482.8085

Calculate AIC

$AIC = -2 * LL + 2 * numParam$

```
numParam = 2; % 1 k, 1 beta parameter

aic_linear = -2 .* (-negLL_linear) + 2 * numParam;
aic_para = -2 .* (-negLL_para) + 2 * numParam;
```

```
aic_hyper = -2 .* (-negLL_hyper) + 2 * numParam;
```

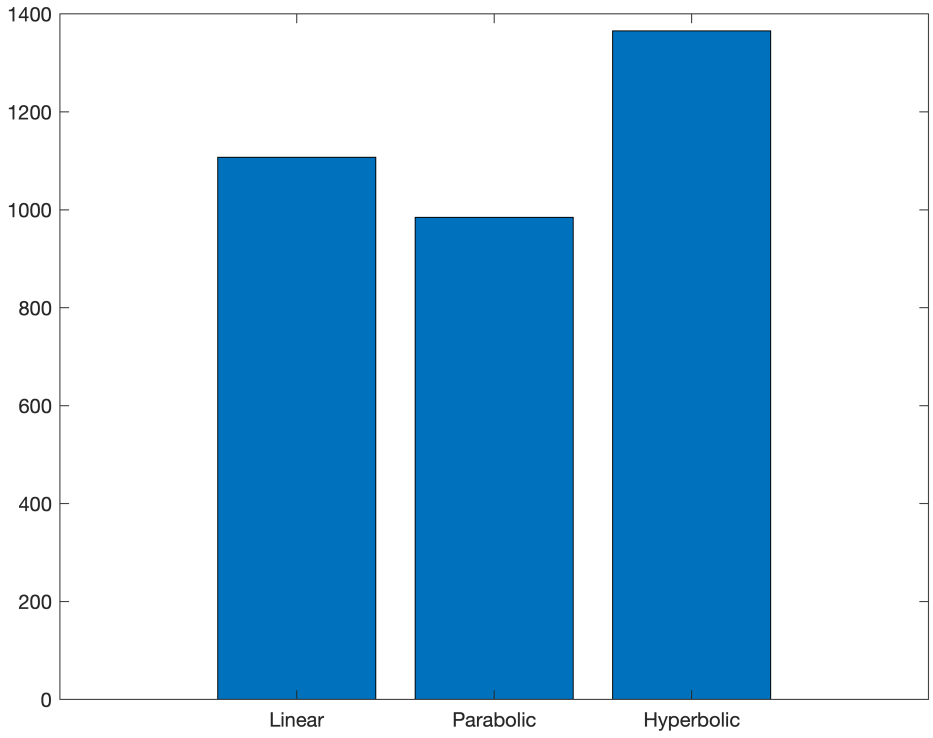
Note: the negLL parameter is actually the log-likelihood with its sign flipped to make it positive (to help the fmin function minimise the LL). Here we flip the sign back to get the actual log-likelihood (which confusingly, is a negative number).

```
aic_table = table(sum(aic_linear), sum(aic_para), sum(aic_hyper), 'VariableNames', ["L
aic_table
```

aic_table = 1×3 table

	Linear	Parabolic	Hyperbolic
1	1.1071e+03	984.5226	1.3656e+03

```
bar(1:3,[sum(aic_linear), sum(aic_para), sum(aic_hyper)])
set(gca, 'xticklabel', ["Linear", "Parabolic", "Hyperbolic"])
```



Calculate BIC

$BIC = -2 * LL + log(numObs) * numParam$

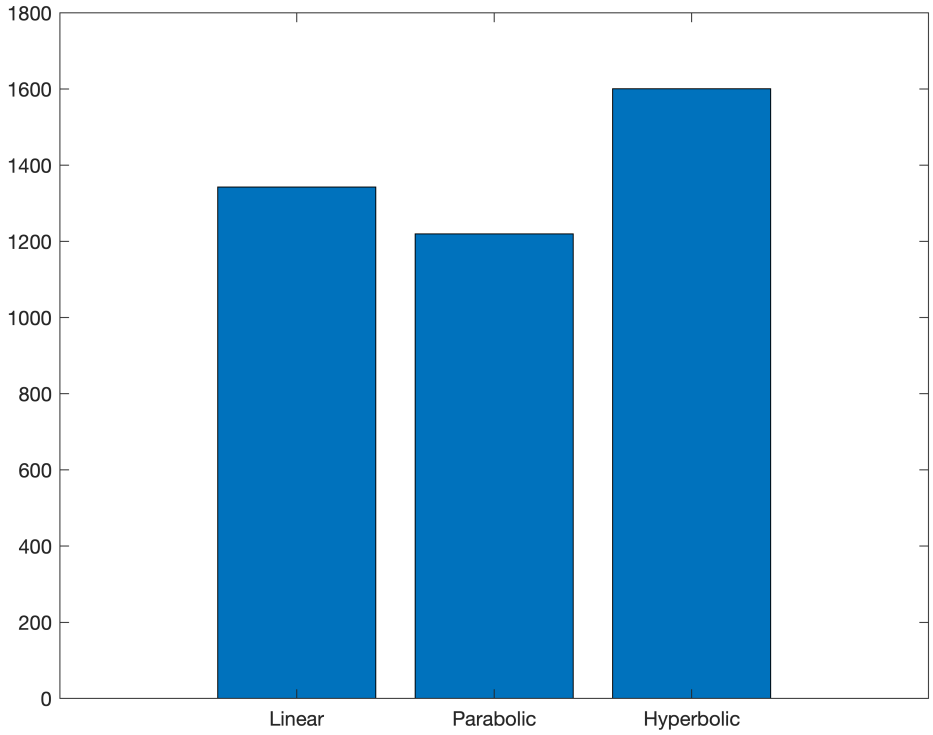
```
numParam = 2; % 1 k, 1 beta parameter
numObs = height(subj_data); % 24, number of observations (trials) for each participant

bic_linear = -2 .* (-negLL_linear) + log(numObs) * numParam;
bic_para = -2 .* (-negLL_para) + log(numObs) * numParam;
bic_hyper = -2 .* (-negLL_hyper) + log(numObs) * numParam;
```

```
bic_table = table(sum(bic_linear), sum(bic_para), sum(bic_hyper), 'VariableNames', ["L
bic_table
```

bic_table = 1×3 table

	Linear	Parabolic	Hyperbolic
1	1.3427e+03	1.2201e+03	1.6012e+03

```
bar(1:3,[sum(bic_linear), sum(bic_para), sum(bic_hyper)])
set(gca, 'xticklabel', ["Linear", "Parabolic", "Hyperbolic"])
```



Fitting multiple k and beta parameters

Simulate data based on 1 k and 1 beta parameter

```
nsims = 100;
data_one_k_one_b_lin = cell(1, nsims);

true_k_params = rand(1, nsims) * 2;
true_beta_params = rand(1, nsims) * 3;

for isubj = 1:nsims
    data_one_k_one_b_lin{isubj} = simulateData([true_k_params(isubj), ...
                                                true_beta_params(isubj)], ...
                                                'one_k_one_beta_linear');
```

```
end
```

Fit the data to a model assuming 1 k and 1 beta

```
model_id = 'one_k_one_beta_linear';
lb = [0, 0];
ub = [2, 5];
nparams = 2;

fit_smallmodel = zeros(nsims, nparams);
negLL_smallmodel = zeros(1, nsims);

for isubj = 1:length(data_one_k_one_b_lin)

    subj_data = data_one_k_one_b_lin{isubj};

    % Fit the data using MLE
    inputfun = @(params) fitModel(subj_data, params, model_id);
    initial_params = rand(1, nparams);
    [fit_smallmodel(isubj, :), negLL_smallmodel(isubj)] = fmincon(inputfun, ...
                                                          initial_params, [], [], [],
                                                          lb, ub, [], ...
                                                          fminoptions);
end

sumnegLL_smallmodel = sum(negLL_smallmodel);
aic_smallmodel = sum(-2 .* (-negLL_smallmodel) + 2 .* nparams);
bic_smallmodel = sum(-2 .* (-negLL_smallmodel) + log(height(subj_data)) .* nparams);
```

Fit the data to a model assuming 2 k and 2 beta

```
model_id = 'two_k_two_beta_linear';
lb = [0, 0, 0, 0];
ub = [2, 2, 5, 5];
nparams = 4;

fit_largemodel = zeros(nsims, nparams);
negLL_largemodel = zeros(1, nsims);

for isubj = 1:length(data_one_k_one_b_lin)

    subj_data = data_one_k_one_b_lin{isubj};

    % Fit the data using MLE
    inputfun = @(params) fitModel(subj_data, params, model_id);
    initial_params = rand(1, nparams);
    [fit_largemodel(isubj, :), negLL_largemodel(isubj)] = fmincon(inputfun, ...
                                                          initial_params, [],
                                                          lb, ub, [], ...
```

```
                                                                    fminoptions);
end

sumnegLL_largemodel = sum(negLL_largemodel);
aic_largemodel = sum(-2 .* (-negLL_largemodel) + 2 .* nparams);
bic_largemodel = sum(-2 .* (-negLL_largemodel) + log(height(subj_data)) .* nparams);
```

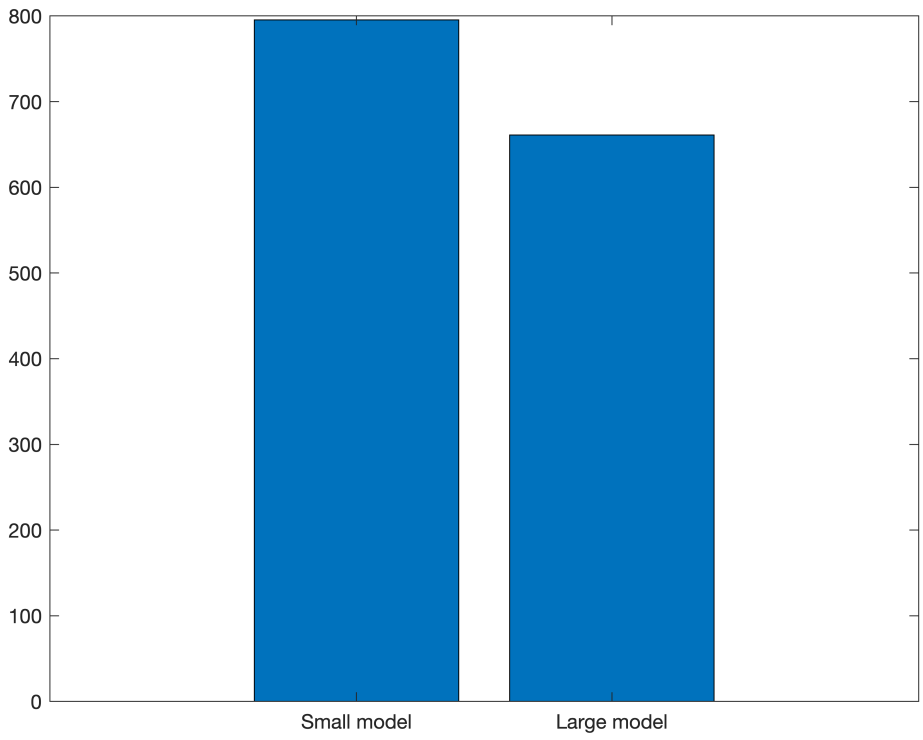
Compare negLL, AIC, and BIC between models

negLL

```
disp(table(sumnegLL_smallmodel, sumnegLL_largemodel))
```

sumnegLL_smallmodel	sumnegLL_largemodel
795.43	661.15

```
bar(1:2, [sum(negLL_smallmodel), sum(negLL_largemodel)])
set(gca, 'xticklabels', ["Small model", "Large model"])
```

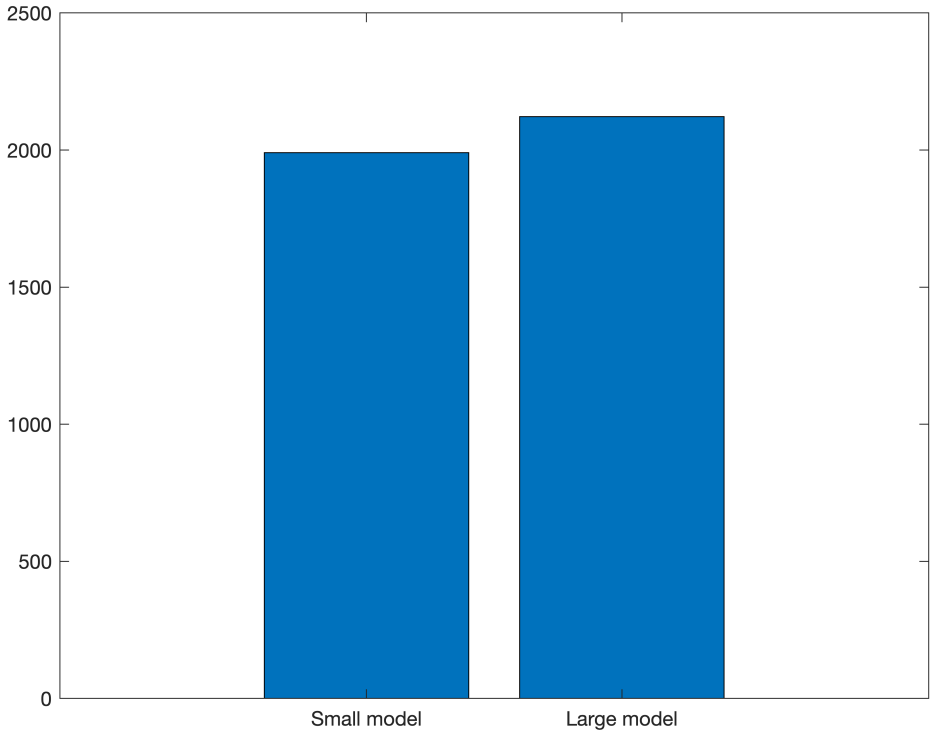


AIC

```
disp(table(aic_smallmodel, aic_largemodel))
```

aic_smallmodel	aic_largemodel
1990.9	2122.3

```
bar(1:2, [aic_smallmodel, aic_largemodel])
set(gca, 'xticklabels', ["Small model", "Large model"])
```

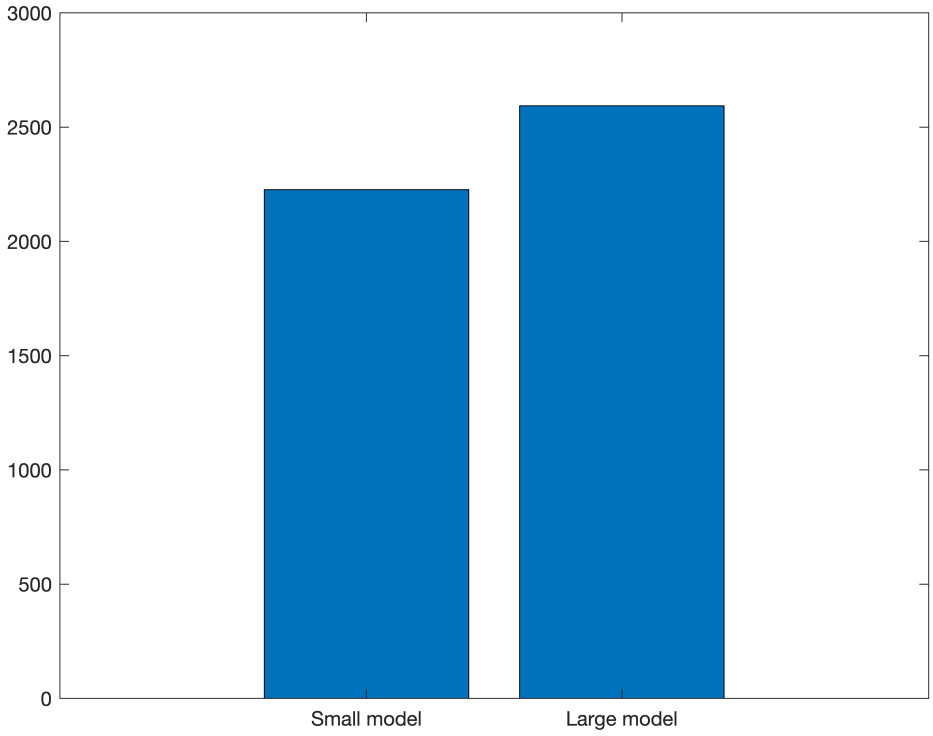


BIC

```
disp(table(bic_smallmodel, bic_largemodel))
```

bic_smallmodel	bic_largemodel
2226.5	2593.5

```
bar(1:2, [bic_smallmodel, bic_largemodel])
set(gca, 'xticklabels', ["Small model", "Large model"])
```



Functions

```
function negLL = fitModel(data, params, model_id)
% Fits parameters to model and calculates softmax and negative log-likelihood

arguments
    data table
    params (1, :) double
    model_id string = 'one_k_one_beta_linear' %defaults to linear if model_id not spec
end

% Get reward, effort, and choices from data
reward = data.reward;
effort = data.effort;
choices = data.choices;
recipient = data.recipient;


% Assign k parameter values
if contains(model_id, 'one_k')
    n_ks = 1;
    k_param = repelem(params(1), length(recipient), 1);
elseif contains(model_id, 'two_k')
    n_ks = 2;
    k_param = (recipient=="food").*params(1) + (recipient=="climate").*params(2);
```

```
else
    error('k parameters in %s misspecified', model_id);
end

% Assign beta parameter values
if contains(model_id, 'one_beta')
    beta_param = repelem(params(1 + n_ks), length(recipient), 1);
elseif contains(model_id, 'two_beta')
    beta_param = (recipient=="food").*params(1 + n_ks) + (recipient=="climate").*param
else
    error('beta parameters in %s misspecified', model_id);
end


% Compute subjective value of "work" option
% (Based on type of model specified by 'model_id')
if contains(model_id, 'linear')
    SV = reward - (effort .* k_param);
elseif contains(model_id, 'para')
    SV = reward - ((effort.^2) .* k_param);
elseif contains(model_id, 'hyper')
    SV = reward ./ (1 + (effort .* k_param));
else
    error("Model ID: %s is incorrectly specified.", model_id)
end


% Calculate probabilities of choosing work option
prob = exp(SV .* beta_param) ./ (exp(SV .* beta_param) + exp(1 .* beta_param));

% Probability of choosing "rest" option
prob(~choices) = 1 - prob(~choices);

% Convert probabilities to log-likelihoods
LL = log(prob);
negLL = -sum(LL); %sum LL together and flip sign to make values positive (for fmin min

end


function data = simulateData(true_params, model_id)
% Generates simulated data for a participant based on the "true" parameters supplied

arguments
    true_params (1, :) double
    model_id string = 'one_k_one_beta_linear' %defaults to linear if model_id not spec
end

reward = repmat(2:4, 1, 8)';
effort = repmat(repelem(2:5, 3), 1, 2)'; % repeat same 12 trials for each recipient
```

```
recipient = repelem(["food", "climate"], 12)';

% Define k parameters
if contains(model_id, 'one_k')
    n_ks = 1;
    k_param_true = repelem(true_params(1), length(recipient), 1);
elseif contains(model_id, 'two_k')
    n_ks = 2;
    k_param_true = (recipient=="food").*true_params(1) + (recipient=="climate").*true_
else
    error('k parameters in %s misspecified', model_id);
end

% Define beta parameters
if contains(model_id, 'one_beta')
    beta_param_true = repelem(true_params(1 + n_ks), length(recipient), 1);
elseif contains(model_id, 'two_beta')
    beta_param_true = (recipient=="food").*true_params(1 + n_ks) + (recipient=="climat
else
    error('beta parameters in %s misspecified', model_id);
end

% Compute subjective value of "work" option
% (Based on type of model specified by 'model_id')
if contains(model_id, 'linear')
    SV = reward - (effort .* k_param_true);
elseif contains(model_id, 'para')
    SV = reward - ((effort.^2) .* k_param_true);
elseif contains(model_id, 'hyper')
    SV = reward ./ (1 + (effort .* k_param_true));
else
    error("Model ID: %s is incorrectly specified.", model_id)
end

% Calculate probabilities of choosing work option
% (calculates probability for each row of matrix (i.e., trial) based on SV and baselin
prob = exp(SV .* beta_param_true) ./ (exp(SV .* beta_param_true) + exp(1 .* beta_param_

choices = prob > rand(length(prob), 1);

data = table(choices, reward, effort, recipient);
data.k_param_true = k_param_true;
data.beta_param_true = beta_param_true;

end
```