University of Helsinki, Master's Programme in Data Science
DATA20047 Probabilistic Cognitive Modelling - Spring 2023
Luigi Acerbi

Problem Set 2: Response distribution and model fitting

- This homework problem set focuses on **Week 3 and 4** of the course.
- This problem set is worth **25 points** in total (out of 100 for the full course).
- Check the submission deadline on Moodle! **Note that the deadline is at noon.**

Submission instructions

Submission must be perfomed entirely on Moodle (**not** by email).

1. When you have completed the exercises, save the notebook.
2. Report your solutions and answers on Moodle ("*Problem set 2 answer return*").
3. Submit two files on Moodle ("*Problem set 2 notebook return*"):
    - The notebook as `.ipynb`.
    - The same notebook downloaded as `.pdf` (there are various ways to save the file as PDF, the most general is "File" > "Print Preview" and then print the page to PDF using your browser - remember to enter the Print Preview first).

IMPORTANT

1. Do not share your code and answers with others. Contrary to the class exercises, which you can do with others, these problems are *not* group work and must be done individually.
2. It is allowed to use snippets of code from the lecture exercises and model solutions.
3. It is your responsibility to ensure that the notebook has fully finished running all the cells, all the plots view properly etc. before submitting it. However, the notebook should be runnable from scratch if needed ("Kernel > Restart & Run All").
4. Submit your work by the deadline.
5. Unless stated otherwise, please report your numerical answers in Moodle with full numerical precision (~14-15 digits), unless the answer is an integer.
6. If you are confused, think there is a mistake or find things too difficult, please ask on Moodle.

References

- [**MKG22**] Ma WJ, Körding K, and Goldreich D. "Bayesian Models of Perception and Action: An Introduction". MIT Press, 2022.
- [**AWV12**] Acerbi L, Wolpert DM, Vijayakumar S. "Internal Representations of Temporal Statistics and Feedback Calibrate Motor-Sensory Interval Timing". *PLoS Computational Biology*, 2012. [Link (https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1002771)](https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1002771)

In [2]:
```
# set-up -- do not change
import numpy as np
import numpy.random as npr
import scipy as sp
import scipy.stats as sps
import pandas as pd
import matplotlib.pyplot as plt
npr.seed(1)
```

Question 2.1 (7 pts)

> This question is about computing the total (root mean squared) error (RMSE) for a Bayesian observer, as seen in Week 3 of the course. The take-home message here is that the Bayesian observer whose prior matches the true empirical distribution of stimuli will perform best at the task (lower RMSE), compared to a Bayesian observer with an incorrect (aka *mismatched*) belief about the distribution of stimuli (i.e., whose prior does not match the true stimulus distribution). See Chapter 4.5 of [**MKG22**] and the lecture notes for Week 3.

A Bayesian observer is estimating a stimulus with empirical distribution $p(s) = \text{Uniform}(s; -5, 5)$. The measurement distribution and likelihood are Gaussian $p(x|s) = \mathcal{N}\left(x; s, \sigma^2\right)$ with $\sigma = 2$. We assume that the observer uses the posterior mean estimator $\hat{s}_{PM}$ and we ignore response noise. However, we consider the observer uses as prior a distribution $q(s)$ which might differ from the true prior (mismatched prior).

- a) Compute the total RMSE assuming $q(s) = p(s)$, i.e. the observer uses the true stimulus distribution as prior.
- b) Compute the total RMSE assuming the observer uses an approximate Gaussian prior, $q(s) = \mathcal{N}\left(s; \mu_s, \sigma_s^2\right)$ with mean μ_s and variance σ_s^2 equal to the mean and variance of the true stimulus distribution. *Hint*: You can find the variance of a continuous uniform distribution [here (https://en.wikipedia.org/wiki/Continuous_uniform_distribution)](https://en.wikipedia.org/wiki/Continuous_uniform_distribution).
- c) Compute the total RMSE assuming the observer uses as prior a mismatched, wider Uniform distribution, $q(s) = \text{Uniform}\left(s; -8, 8\right)$.

Report your results in Moodle. The accepted tolerance is ± 0.01 from the true value.

Hints:

- Remember that the (total) RMSE of an estimator $\hat{s}$ is computed as

$$\text{RMSE}[\hat{s}] = \sqrt{\int \text{MSE}\left[\hat{s}|s\right] p(s) ds}$$

  where $p(s)$ is the true empirical distribution and $\text{MSE}\left[\hat{s}|s\right]$ is the mean squared error at each stimulus, defined as

$$\text{MSE}\left[\hat{s}|s\right] = \mathbb{E}_{\hat{s}|s}\left[\left(\hat{s} - s\right)^2 |s\right] = \text{Bias}[\hat{s}|s]^2 + \text{Var}\left[\hat{s}|s\right],$$

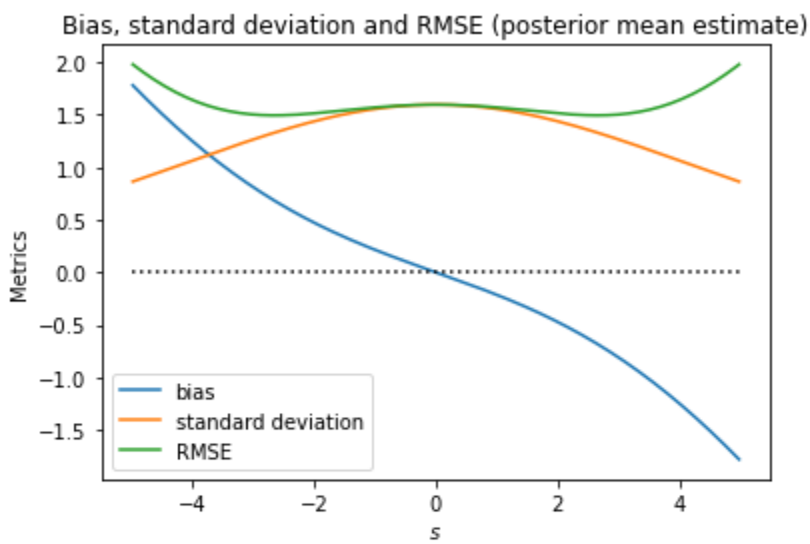
  where the definitions for bias and variance can be found in the textbook or lecture notes.
- Note that changing the prior $q(s)$ will change $\hat{s}(x)$, but nothing else! So once you manage to compute (a), you should be able to compute (b) and (c) with a small change to the code (only where $\hat{s}(x)$ is computed).
- You may want to check out Exercise 3.3 of the workshops.

In [3]:
```
def compute_posterior_mean_1d(s_grid, prior_pdf, likelihood):
    """Compute s_hat_PM (posterior mean) for an arbitrary prior and likelih
ood in 1d."""
    ds = s_grid.flatten()[1] - s_grid.flatten()[0] # grid spacing
    protoposterior = prior_pdf * likelihood
    normalization_constant = sp.integrate.romb(protoposterior, dx=ds, axis=
0)
    posterior_pdf = protoposterior / normalization_constant
    posterior_mean = sp.integrate.romb(s_grid * posterior_pdf, dx=ds, axis=
0)
    return posterior_mean

def compute_and_plot_metrics(r_col, s_hat, stimulus_pdf, label):
    """Compute bias, variance, conditional and overall MSE."""
    bias = sp.integrate.romb(s_hat * sps.norm.pdf(x_row,r_col,sigma),dx=dx,
axis=1) - r_col.flatten()
    std = np.sqrt(sp.integrate.romb(s_hat**2 * sps.norm.pdf(x_row,r_col,sig
ma),dx=dx,axis=1)
        - sp.integrate.romb(s_hat * sps.norm.pdf(x_row,r_col,sigma),dx=dx,a
xis=1)**2)
    rmse = np.sqrt(bias**2 + std**2)

    # Plot only where the support of the Uniform stimulus distribution is n
onzero
    s_range = r_col.copy()
    s_range[np.logical_or(s_range < a, s_range > b)] = np.nan

    plt.plot(s_range.flatten(),bias.flatten(),label='bias')
    plt.plot(s_range.flatten(),std.flatten(),label='standard deviation')
    plt.plot(s_range.flatten(),rmse.flatten(),label='RMSE')
    plt.plot((a,b),(0,0),':k')
    plt.xlabel(r'$s$')
    plt.ylabel(r'Metrics')
    plt.title('Bias, standard deviation and RMSE (' + label + ')')
    plt.legend()
    plt.show()

    total_rmse = np.sqrt(sp.integrate.romb(rmse**2 * stimulus_pdf,dx=ds))
    print('Total RMSE (' + label + '): {}'.format(total_rmse))
```

2.1

In [4]:
```
a = -5.
b = 5.
sigma = 2.

print('a)')
Nx = 2**9+1
Ns = 2**9+1

# define the grid
lb = a - sigma*5.
ub = b + sigma*5.
x_row = np.linspace(lb, ub, Nx).reshape((1,Nx)) # make x a row vector
r_col = np.linspace(lb, ub, Ns).reshape((Ns,1)) # make s a column vector
dx = x_row.flatten()[1] - x_row.flatten()[0]
ds = r_col.flatten()[1] - r_col.flatten()[0]

prior_pdf = sps.uniform.pdf(r_col, a, b-a)

likelihood = sps.norm.pdf(x_row, r_col, sigma)
s_hat_row = compute_posterior_mean_1d(r_col, prior_pdf, likelihood).reshape
((1,Nx)) # keep as a row vector

stimulus_pdf = prior_pdf.copy().flatten()# As in most cases we consider, th
ese two pdfs are the same
compute_and_plot_metrics(r_col, s_hat_row, stimulus_pdf, 'posterior mean es
timate')
```

```
a)
```

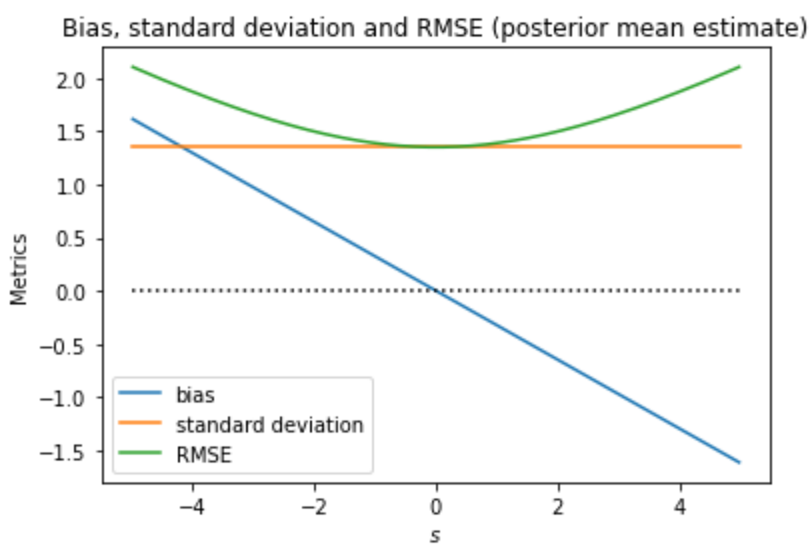



```
Total RMSE (posterior mean estimate): 1.6048573201415854
```

In [5]:

```
a = -5.
b = 5.
sigma = 2.

print('b)')
Nx = 2**9+1
Ns = 2**9+1

# define the grid
lb = a - sigma*5.
ub = b + sigma*5.
x_row = np.linspace(lb, ub, Nx).reshape((1,Nx)) # make x a row vector
r_col = np.linspace(lb, ub, Ns).reshape((Ns,1)) # make s a column vector
dx = x_row.flatten()[1] - x_row.flatten()[0]
ds = r_col.flatten()[1] - r_col.flatten()[0]

variance = ((b - a)**2)/12
mean = (b + a)/2
sigma = np.sqrt(variance)
prior_pdf = sps.norm.pdf(r_col, mean, sigma)

#New sigma for likelihood
sigma=2
likelihood = sps.norm.pdf(x_row, r_col, sigma)
s_hat_row = compute_posterior_mean_1d(r_col, prior_pdf, likelihood).reshape
((1,Nx)) # keep as a row vector

stimulus_pdf = prior_pdf.copy().flatten()# As in most cases we consider, th
ese two pdfs are the same
compute_and_plot_metrics(r_col, s_hat_row, stimulus_pdf, 'posterior mean es
timate')
```

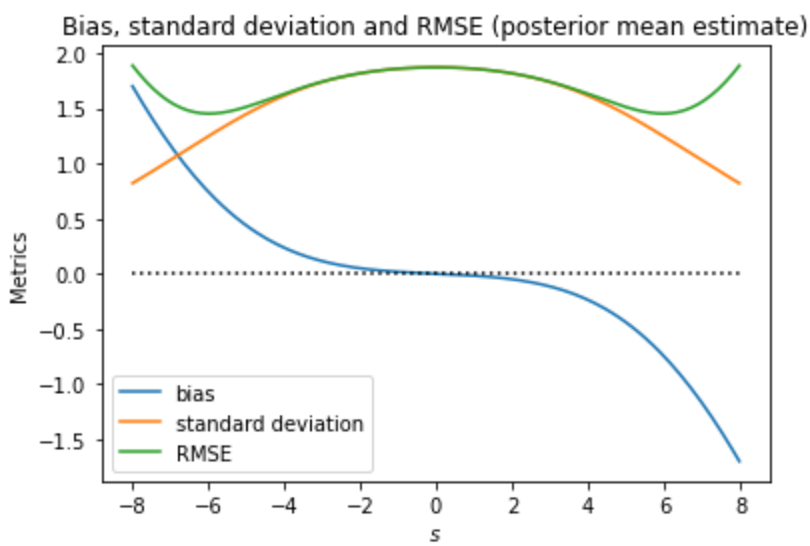
b)



Total RMSE (posterior mean estimate): 1.6446507755833726

In [26]:
```
a = -8.
b = 8.
sigma =1.9

print('c)')
Nx = 2**9+1
Ns = 2**9+1


# define the grid
lb = a - sigma*8.
ub = b + sigma*8.
x_row = np.linspace(lb, ub, Nx).reshape((1,Nx)) # make x a row vector
r_col = np.linspace(lb, ub, Ns).reshape((Ns,1)) # make s a column vector
dx = x_row.flatten()[1] - x_row.flatten()[0]
ds = r_col.flatten()[1] - r_col.flatten()[0]


prior_pdf = sps.uniform.pdf(r_col, a, b-a)

likelihood = sps.norm.pdf(x_row, r_col, sigma)
s_hat_row = compute_posterior_mean_1d(r_col, prior_pdf, likelihood).reshape
((1,Nx)) # keep as a row vector

stimulus_pdf = prior_pdf.copy().flatten()# As in most cases we consider, th
ese two pdfs are the same
compute_and_plot_metrics(r_col, s_hat_row, stimulus_pdf, 'posterior mean es
timate')
```

```
c)
```




```
Total RMSE (posterior mean estimate): 1.6842547306881315
```

Question 2.2 (6 pts)

> In this question, we compute the response distribution for [**JS10**] under different assumptions about the Bayesian observer.

Consider the time perception experiment from [**JS10**] which we analyzed in Exercise 3.4. We recall the setup below. Note that there are differences from Exercise 3.4 (marked as **NEW**):

- In this experiment, an observer is asked to judge the time interval s between two flashes, measured in milliseconds (ms). In each trial, the duration is drawn from an interval distribution $p(s)$.
- The experiment consist of three separate blocks of sessions run over multiple days. Each experimental block is identical except for the distribution of intervals $p(s)$. The distribution of time intervals in the three blocks are:
  - $p_{\text{short}}(s) = \text{Uniform}\,(s; 494, 847)$
  - $p_{\text{medium}}(s) = \text{Uniform}\,(s; 671, 1023)$
  - $p_{\text{long}}(s) = \text{Uniform}\,(s; 847, 1200)$
- The observer's measurement distribution follows *Weber's law* (known in time perception as the "scalar property" of temporal judgment). According to this empirical law, the measurement noise is roughly linearly proportional to the magnitude of the stimulus. In formulas,
  $$p(x|s) = \mathcal{N}\left(x|s, \sigma^2(s)\right) \qquad \text{with} \quad \sigma(s) = w_s \cdot s$$
  where w_s is known as *Weber's fraction*. Typical values of w_s in timing are around 0.05-0.2, here we assume $w_s = 0.1$.
- It is assumed that, after some practice, the observer develops a prior $p(s)$ which matches the stimulus distribution used in that block of sessions (and that the likelihood also matches the measurement distribution).
- **NEW**: The observer responds with a deterministic estimate $\hat{s}_{\text{MAP}}$ which we assume is the mode of the posterior (also known as *maximum-a-posteriori* or MAP estimate).
- **NEW**: The response is corrupted by motor noise which is proportional to the estimate:
  $$p(r|\hat{s}) = \mathcal{N}\left(r; \hat{s}, \sigma^2_{\text{m}}(\hat{s})\right) \qquad \text{with} \quad \sigma_{\text{m}}(\hat{s}) = w_{\text{m}} \cdot \hat{s}$$
  where w_{m} represents the Weber's fraction for the motor noise. Here we assume $w_{\text{m}} = 0.05$.

In this exercise, we look at the *distribution of responses* $p(r|s)$ that the experimenter would observe for a given stimulus in the three different experimental blocks (short, medium, or long). We consider the stimulus $s^\star = 847$ ms which appears in all three experimental blocks.

- a) Compute $p(r|s = s^\star)$ for the "short" block. Compute the mean and standard deviation of $p(r|s = s^\star)$ and report them on Moodle.
- b) Compute $p(r|s = s^\star)$ for the "medium" block. Compute the mean and standard deviation of $p(r|s = s^\star)$ and report them on Moodle.
- c) Compute $p(r|s = s^\star)$ for the "long" block. Compute the mean and standard deviation of $p(r|s = s^\star)$ and report them on Moodle.

The accepted tolerance for the solutions is ± 0.2 ms for (a) and (b), and ± 0.5 ms for (c).

Hints:

- Be careful that the likelihood, $p(x|s)$ as a function of s, is *not* Gaussian, because $\sigma(s)$ is not constant in s. As a consequence, the posterior will *not* be Gaussian. This affects the MAP estimate, $\hat{s}_{\mathrm{MAP}}$, which you will need to compute numerically.
- To compute the response distribution, remember the definition:
$$p(r|s) = \int p(r|\hat{s}(x))p(x|s)dx,$$
which you can solve via numerical integration.
- It is recommended that you first compute $\hat{s}_{\mathrm{MAP}}(x)$ for a grid of x, and then compute the response distribution numerically via the integral above.
- The MAP estimate $\hat{s}_{\mathrm{MAP}}$ is the value of s that maximizes the posterior $p(s|x)$. Note that this value does not depend on the normalization constant, so you can compute $p(s|x) \propto p(s)p(x|s)$ for a (fine) grid of values `s_grid` and take the argument s that maximizes this quantity.

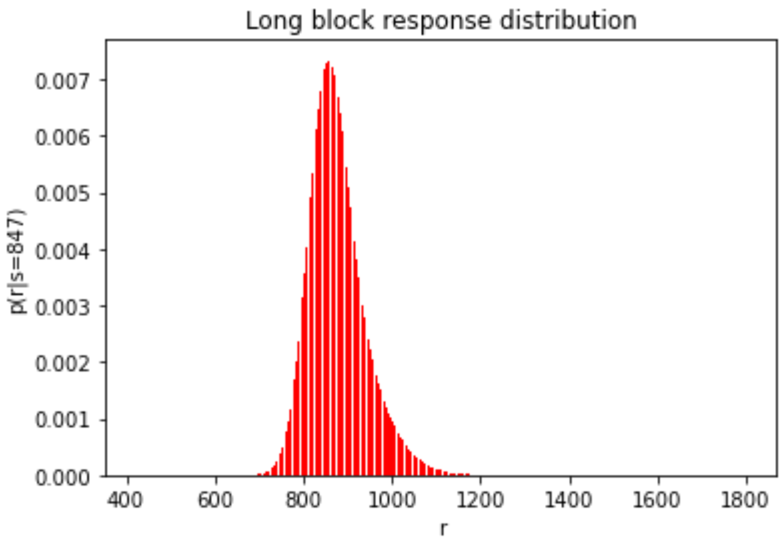
In [7]:
```
def sanity_check(x_row,r_col,integrand, s_hat_row, response_integral_pdf):
    if x_row[0][0] == 345.79999999999995:
        idx = (np.abs(x_row[0] - 759)).argmin()
        print("Sanity check: x=759, sMap=751.50")
        print(f"x={x_row[0][idx]}, sMAP={s_hat_row[0][idx]}")

        print('x_row.shape: {}'.format(x_row.shape))
        print('r_col.shape: {}'.format(r_col.shape))
        print('s_hat_row.shape: {}'.format(s_hat_row.shape))
        print('integrand.shape: {}'.format(integrand.shape))
        print('response_integral_pdf.shape: {}'.format(response_integral_pd
f.shape))
```

In [8]:
```
def plot_distribution(x, y, x_label, y_label, title):
    plt.bar(x, y , color='red', width=0.5)
    plt.xlabel(x_label)
    plt.ylabel(y_label)
    plt.title(title)
    plt.show()
```

In [62]:
```
a = np.array((494, 671, 847))
b = np.array((847,1023,1200))
label = ('short','medium','long')
w_s = 0.1
w_m = 0.05

#res=11, clearance=5 for correct results
res = 11
Nx = 2**res+1
Ns = 2**res+1

# calculate sMAP
def compute_maximum_a_posteriori_1d(s_grid, prior_pdf, likelihood):
    posterior = likelihood * prior_pdf
    map_estimate = s_grid[np.argmax(posterior, axis=0)]
    return np.transpose(map_estimate) #must be a row vector

# calculate response distribution
def calculate_response_dist(x_row, r_col, likelihood, prior_pdf, _dx):
    s_hat_row = compute_maximum_a_posteriori_1d(r_col, prior_pdf, likelihood)
    measurement_dist = sps.norm.pdf(x_row, 847, w_s*847)
    integrand = sps.norm.pdf(r_col, s_hat_row, w_m * s_hat_row) * measurement_dist
    response_dist = sp.integrate.romb(integrand, dx=_dx, axis=1)
    sanity_check(x_row,r_col,integrand, s_hat_row, response_dist)
    return response_dist

for iter in range(a.size):
    # define the grid
    clearance = 5
    lb = a[iter] - (w_s*a[iter])*clearance
    ub = b[iter] + (w_s*b[iter])*clearance
    x_row = np.linspace(lb, ub, Nx).reshape((1,Nx)) # make x a row vector
    r_col = np.linspace(lb, ub, Ns).reshape((Ns,1)) # make s a column vector
    _dx = x_row.flatten()[1] - x_row.flatten()[0]

    #calculate response distribution p(r|s=847)
    prior_pdf = sps.uniform.pdf(r_col, loc=a[iter], scale=b[iter]-a[iter])
    likelihood = sps.norm.pdf(x_row, r_col, w_s*r_col)
    response_dist = calculate_response_dist(x_row, r_col, likelihood, prior_pdf, _dx)

    #mean and SD calculation
    flat = r_col.flatten()
    mean_r = sp.integrate.romb(flat * response_dist, dx=_dx)
    var = sp.integrate.romb((flat - mean_r)**2 * response_dist, dx=_dx)
    std = np.sqrt(var)

    #results
    print("------------"+ label[iter] + "------------")
    print(f"mean= {mean_r}")
    print(f"std= {std}")
    print("1 ~=", sp.integrate.romb(response_dist, dx=_dx)) #should be appr
```

```
px 1

plot_distribution(r_col.flatten(), response_dist , "r", "p(r|s=847)",
                  "Long block response distribution")
```

```
------------short------------
mean= 809.0869476150397
std= 65.67980760174596
1 ~= 0.9999997133477272
------------medium------------
mean= 839.0043753778564
std= 91.38405383325505
1 ~= 0.9999999992244515
------------long------------
mean= 876.7796740577605
std= 63.521941134265894
1 ~= 0.9999997133484282
```



Question 2.3 (6 pts)

> The key quantity for model fitting is the log-likelihood for a dataset and some model parameters. In this exercise, we compute the log-likelihood for a Bayesian observer model which also includes the possibility of *lapses*, a common mechanism used in cognitive science to explain away "random" responses and subjects' mistakes.

In this question, we consider the datasets from Experiment 3 of [**AWV12**], as seen in Week 4.. The experimental setup which involves time perception and interval reproduction is very similar to [**JS10**], so we can consider the same type of models.

We analyze the data with the `gaussianobserverwithlapse` model, defines as follows:

- We assume the observer builds a (mismatched) Gaussian prior $p(s) = \mathcal{N}\left(s|\mu_{\text{prior}}, \sigma^2_{\text{prior}}\right)$ over the stimuli (time intervals).
- We assume that the measurement distribution and likelihood are also Gaussian, $p(x|s) = \mathcal{N}\left(x|s, \sigma^2\right)$.
- The observer uses the *posterior mean* estimator for the value of the stimulus, $\hat{s}_{\text{PM}}$.
- Gaussian motor response noise is added to the estimate, $p(r|\hat{s}) = \mathcal{N}\left(r|\hat{s}, \sigma^2_{\text{motor}}\right)$.
- In each trial, the observer lapses with probability λ (the *lapse rate*), in which case the response is drawn from $p_{\text{lapse}}(r) = \text{Uniform}\left(r; 0, 1500\right)$ ms. Otherwise, the observer responds normally (according to $p(r|\hat{s})$ described above) with probability $1 - \lambda$.
- The parameters of this model are $\theta = (\mu_{\text{prior}}, \sigma_{\text{prior}}, \sigma, \sigma_{\text{motor}}, \lambda)$.

For this question, we consider parameters
$\theta_\star = (\mu_{\text{prior}} = 780, \sigma_{\text{prior}} = 140, \sigma = 90, \sigma_{\text{motor}} = 60, \lambda = 0.02)$.

- a) Compute the log-likelihood of model parameter $\theta_\star$ for the dataset of subject 2.
- b) Compute the log-likelihood of model parameter $\theta_\star$ for the dataset of subject 5.

Report your results on Moodle with high precision.

Hint:

- If you use code from the lectures, be careful about the model definition, as there may be subtle differences.

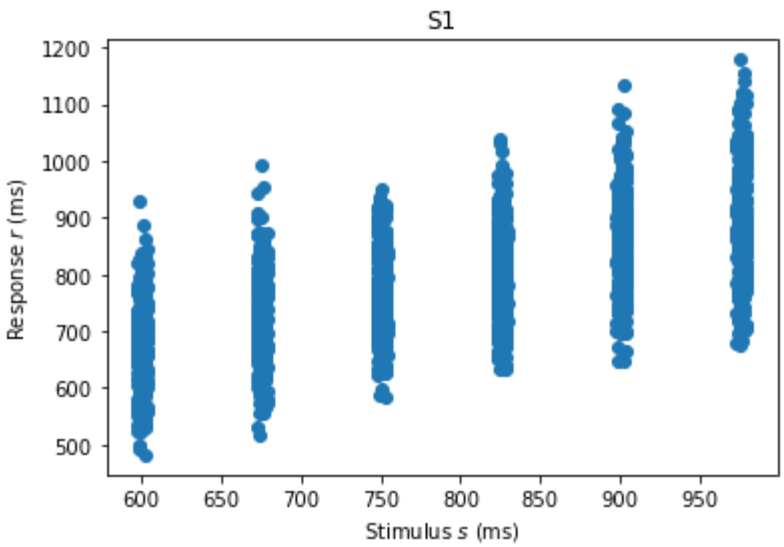
In [10]:
```
# Load data of Experiment 3 of [AWV12] from .csv file to a Pandas dataframe
df = pd.read_csv('https://www2.helsinki.fi/sites/default/files/atoms/files/
awv12_exp3.csv')

# Remove unused columns (they deal with performance feedback, which we igno
re in this lecture)
df.drop(df.columns[[6, 7, 8]], axis=1, inplace=True)

# Remove rows with NaNs
df.dropna(axis=0, inplace=True)

df.head()
```

Out[10]:

	Subject id	Session id	Run id	Stimulus (ms)	Response (ms)	Stimulus id
0	1	1	1	973.327049	862.947945	6.0
1	1	1	1	677.519900	574.920276	2.0
2	1	1	1	826.253049	870.995615	4.0
3	1	1	1	677.854859	695.055098	2.0
4	1	1	1	598.501198	632.981845	1.0

In [11]:
```
# Example code to extract stimuli and responses for a single subject (here
S1)
subject = 1
s = np.array(df['Stimulus (ms)'][df['Subject id'] == subject])
r = np.array(df['Response (ms)'][df['Subject id'] == subject])
print('s.shape:', s.shape)
print('r.shape:', r.shape)

plt.scatter(s, r)
plt.xlabel('Stimulus $s$ (ms)')
plt.ylabel('Response $r$ (ms)')
plt.title('S' + str(subject))
plt.show()
```

```
s.shape: (2520,)
r.shape: (2520,)
```



In [13]:
```
def gaussian_response(s,theta):
    """Compute mean and standard deviation of p(r|s; theta)."""
    # Unpack parameter vector theta
    mu_prior = theta[0]
    sigma_prior = theta[1]
    sigma = theta[2]
    sigma_motor = theta[3]
    # Compute mean and std of the response
    w = sigma_prior**2/(sigma_prior**2 + sigma**2)
    mu_resp = w*s + (1-w)*mu_prior
    sigma_resp = np.sqrt(w**2*sigma**2 + sigma_motor**2)
    return mu_resp, sigma_resp

def idealgaussianobserverwithlapse_loglike(theta,s_vec,r_vec):
    """Log-likelihood of ideal Gaussian observer with added lapse."""
    mu_prior = 780
    sigma_prior = 140

    sigma = theta[0]
    sigma_motor = theta[1]
    lapse_rate = theta[2]

    lapse_pdf = 1/1500.

    mu_resp, sigma_resp = gaussian_response(s_vec,np.array((mu_prior,sigma_
prior,sigma,sigma_motor)))
    # First, compute log-likelihood without probability of lapse
    loglike_vec = sps.norm.logpdf(r_vec,mu_resp,sigma_resp) # Vector of log
-likelihood per trials
    # Now, add the probability of lapse
    if lapse_rate > 0.:
        likelihood_vec = np.exp(loglike_vec) # Exponentiate back to the lik
elihood
        likelihood_with_lapse_vec = (1-lapse_rate)*likelihood_vec + lapse_r
ate*lapse_pdf
        loglike_vec = np.log(likelihood_with_lapse_vec)
        # This code snippet below uses the logsumexp trick, which is numeri
cally more stable
        # loglapse = np.log(lapse_rate*lapse_pdf)
        # M = np.maximum(loglike, loglapse)
        # loglike = np.log((1-lapse_rate)*np.exp(loglike-M) + np.exp(loglap
se-M)) + M
    return np.sum(loglike_vec)

theta1 = (90,60,0.02)

print('b)')

subject = 2
s = np.array(df['Stimulus (ms)'][df['Subject id'] == subject])
r = np.array(df['Response (ms)'][df['Subject id'] == subject])
loglike1 = idealgaussianobserverwithlapse_loglike(theta1,s,r)

print('SUBJECT 2: The log-likelihood of theta_1 = {} (dataset S{}) is: {}'.
format(
```

```
    theta1, subject, loglike1))


###################################################################

subject = 5
s = np.array(df['Stimulus (ms)'][df['Subject id'] == subject])
r = np.array(df['Response (ms)'][df['Subject id'] == subject])
loglike1 = idealgaussianobserverwithlapse_loglike(theta1,s,r)

print('SUBJECT 5: The log-likelihood of theta_1 = {} (dataset S{}) is: {}'.
format(
    theta1, subject, loglike1))
```

b)

```
SUBJECT 2: The log-likelihood of theta_1 = (90, 60, 0.02) (dataset S2) is:
-8577.318546123512
SUBJECT 5: The log-likelihood of theta_1 = (90, 60, 0.02) (dataset S5) is:
-9105.655523964238
```

Question 2.4 (6 pts)

> When fitting models to data, the experimenter may be interested in how model parameters are represented across the population (here represented by the group of subjects). A simple way to look at this is to look at the distribution of maximum-likelihood estimates for the parameters across subjects, in first instance by looking their mean and variability.

We consider here the `idealgaussianobserverwithlapse` model. This model is the same as the `gaussianobserverwithlapse` of Question 2.3, but with $\mu_{\text{prior}} = 787.5$ ms and $\sigma_{\text{prior}} = 128.1$ ms fixed. Thus, the model has three free parameters, $\theta = (\sigma, \sigma_{\text{motor}}, \lambda)$. Fit the model using maximum-likelihood estimation.

- a) First, fit the `idealgaussianobserverwithlapse` model to the six subjects' datasets (separately for each subject's data). For each maximum-likelihood estimate (MLE) of parameters $\sigma, \sigma_{\text{motor}}, \lambda$, report in Moodle the mean and standard deviation across the six subjects. For the standard deviation, use the correction for degrees of freedom (that is, `np.std(..., ddof=1)`).
- b) Now fit the pooled data of all subjects as a single dataset (as if all data were collected from a single uber-subject). Report the maximum-likelihood estimate of $\sigma, \sigma_{\text{motor}}, \lambda$ for the pooled data in Moodle.

Hints:

- If you use code for the `idealgaussianobserverwithlapse` model from the lectures, be careful about the model definition.
- As a sanity check that you have coded the log-likelihood function correctly, check that the log-likelihood of the dataset of subject 1 for $\theta_\star = (\sigma = 90, \sigma_{\text{motor}} = 80, \lambda = 0.02)$ is $\log \mathcal{L}(\theta_\star; \mathcal{D}_1) \approx -14709.795\ldots$

Note: Fitting individual subjects' data is the best approach to describe invidual behavior in cognitive science, but sometimes you will see studies only looking at pooled/group data. Be careful that pooling might hide what really happens, only giving a snapshot of the average behavior of the group, which might not correspond to what individuals do.

In [37]:
```
def idealgaussianobserverwithlapse_loglike(theta,s_vec,r_vec):
    """Log-likelihood of ideal Gaussian observer with added lapse."""
    mu_prior = 787.5
    sigma_prior = 128.1
    sigma = theta[0]
    sigma_motor = theta[1]
    lapse_rate = theta[2]
    lapse_pdf = 1/1500.
    mu_resp, sigma_resp = gaussian_response(s_vec,np.array((mu_prior,sigma_prior,sigma,sigma_motor)))
    # First, compute log-likelihood without probability of lapse
    loglike_vec = sps.norm.logpdf(r_vec,mu_resp,sigma_resp) # Vector of log-likelihood per trials
    # Now, add the probability of lapse
    if lapse_rate > 0.:
        likelihood_vec = np.exp(loglike_vec) # Exponentiate back to the likelihood
        likelihood_with_lapse_vec = (1-lapse_rate)*likelihood_vec + lapse_rate*lapse_pdf
        loglike_vec = np.log(likelihood_with_lapse_vec)
    return np.sum(loglike_vec)

def multioptimize(target_fun,lower_bounds,upper_bounds,plausible_lower_bounds,plausible_upper_bounds,num_runs=3):
    """Simple function for multi-start optimization."""
    # Run num_runs optimization runs from different starting points
    num_params = lower_bounds.shape[0]
    theta_res = np.zeros((num_runs,num_params))
    nll_res = np.zeros(num_runs)

    for index in range(num_runs):
        if index == 0:
            theta0 = 0.5*(plausible_lower_bounds + plausible_upper_bounds)
        else:
            theta0 = np.random.uniform(low=plausible_lower_bounds,high=plausible_upper_bounds)
        bounds = sp.optimize.Bounds(lower_bounds,upper_bounds,True) # Set hard bounds
        res = sp.optimize.minimize(target_fun, theta0, method='L-BFGS-B', bounds=bounds)
        nll_res[index] = res.fun
        theta_res[index] = res.x

    # Pick the best solution
    idx_best = np.argmin(nll_res)
    nll_best = nll_res[idx_best]
    theta_best = theta_res[idx_best]
    return nll_best,theta_best
```

In [53]:
```
thetas = []
num_runs = 10
num_subjects = 6

for i in range(num_subjects):
    subject = i + 1
    s = np.array(df['Stimulus (ms)'][df['Subject id'] == subject])
    r = np.array(df['Response (ms)'][df['Subject id'] == subject])
    target_fun = lambda theta_: -idealgaussianobserverwithlapse_loglike(np.
array(theta_),s,r)

    # Define hard parameter bounds
    lower_bounds = np.array([1.,1.,0.])
    upper_bounds = np.array([2000.,2000.,1.])

    # Define plausible range
    plausible_lower_bounds = np.array([np.mean(s)*0.05,np.mean(s)*0.05,0.0
1])
    plausible_upper_bounds = np.array([np.mean(s)*0.20,np.mean(s)*0.20,0.0
5])

    #call multioptimise  for subject
    nll2_best,theta2_best = multioptimize(target_fun,lower_bounds,upper_bou
nds,plausible_lower_bounds,plausible_upper_bounds,num_runs)
    thetas.append(theta2_best)

print(f"Thetas (= sigma, sigma_motor, lambda): {thetas}")
```

```
Thetas (= sigma, sigma_motor, lambda): [array([109.596,  42.538,   0.   ]),
array([68.835, 30.883,  0.005]), array([115.919,  71.987,   0.013]), array
([139.914,  91.651,   0.   ]), array([63.395, 76.46 ,  0.008]), array([93.8
98, 73.26 ,  0.007])]
```

In [54]:
```
#thetas = [sigma, ]
sigmas = [x[0] for x in thetas]
sigma_motors = [x[1] for x in thetas]
lambdas = [x[2] for x in thetas]

#calculate mles and stds
sigma_mle = np.mean(sigmas)
sigma_motors_mle = np.mean(sigma_motors)
lambdas_mle = np.mean(lambdas)
sigma_std = np.std(sigmas, ddof=1)
sigma_motors_std = np.std(sigma_motors, ddof=1)
lambdas_std = np.std(lambdas, ddof=1)

print(f"sigma:\n mle = {sigma_mle},\n std = {sigma_std}")
print(f"sigma motors:\n mle = {sigma_motors_mle},\n std = {sigma_motors_st
d}")
print(f"lambdas:\n mle = {lambdas_mle},\n std = {lambdas_std}")
```

```
sigma:
 mle = 98.59291952413031,
 std = 29.241153589295255
sigma motors:
 mle = 64.46303211477215,
 std = 22.91497518125363
lambdas:
 mle = 0.005600439788615318,
 std = 0.0050677241630049065
```

In [ ]:

In [51]:
```
def gaussianobserverwithlapse_loglike(theta,s_vec,r_vec):
    """Log-likelihood of ideal Gaussian observer with added lapse."""
    mu_prior = 787.5
    sigma_prior = 128.1
    sigma = theta[0]
    sigma_motor = theta[1]
    lapse_rate = theta[2]
    lapse_pdf = 1/1500
    mu_resp, sigma_resp = gaussian_response(s_vec,np.array((mu_prior,sigma_
prior,sigma,sigma_motor)))
    # First, compute log-likelihood without probability of lapse
    loglike_vec = sps.norm.logpdf(r_vec,mu_resp,sigma_resp) # Vector of log
-likelihood per trials
    # Now, add the probability of lapse
    if lapse_rate > 0.:
        likelihood_vec = np.exp(loglike_vec) # Exponentiate back to the lik
elihood
        likelihood_with_lapse_vec = (1-lapse_rate)*likelihood_vec + lapse_r
ate*lapse_pdf
        loglike_vec = np.log(likelihood_with_lapse_vec)
        # This code snippet below uses the logsumexp trick, which is numeri
cally more stable
        # loglapse = np.log(lapse_rate*lapse_pdf)
        # M = np.maximum(loglike, loglapse)
        # loglike = np.log((1-lapse_rate)*np.exp(loglike-M) + np.exp(loglap
se-M)) + M
    return np.sum(loglike_vec)
```

In [59]:
```
target_fun = lambda theta_: -gaussianobserverwithlapse_loglike(np.array(the
ta_),s,r)
nll2_best,theta2_best = multioptimize(target_fun,lower_bounds,upper_bounds,
plausible_lower_bounds,plausible_upper_bounds,num_runs=50)
print(theta2_best[:3])
```

```
[98.093 64.117  0.008]
```

In [ ]:

In [ ]: