Bayesian Inference Project

Carlos Enmanuel Soto López[1,2]
carlos.soto362@gmail.com

`Supervisors: Paolo Lazzari`[2] `and Fabio Anselmi`[1,2].
1. Università degli Studi di Trieste, 31127, Italy
2. Istituto nazionale di oceanografia e di geofisica sperimentale - OGS, Trieste, 34010, Italy

1 Problem statement: Bayesian experiment optimization

We investigate the use of a Bayesian approach to optimization of the experimental configuration to maximize a suitably defined utility. We assume that data d have been collected, so that our current knowledge of the system when we build the new experiment is described by the posterior $Pr(\theta|d)$. We introduce the utility $U(\theta, d, e)$ as an inverse loss function, which depends on what we have observed so far (the data d), on the value of the model's parameters θ, and on the experimental configuration of the future experiment, e. From the utility we can build the expected utility, by averaging the utility over the posterior obtained from the past observations, $Pr(\theta|d)$,

$$\mathbb{E}[d|e] = \int d\theta U(\theta, d, e) Pr(\theta|d) \tag{1}$$

Consider the linear model,

$$y = \theta x + \epsilon \tag{2}$$

where ϵ is normally distributed. The past experiment has been performed by measuring the quantity y at two locations, x_0 , x_1 , resulting in data $d = \{y_0, y_1\}$. We assume that the two measurements are uncorrelated, and that the noise from the past experiment is described by a Gaussian with zero mean and variance σ^2 . The standard deviation of the future experiment, e, is described by a noise function $\tau(x)$, describing how the future experiment's accuracy varies with distance along the horizontal axis.

- Adopt as utility function the inverse of the posterior variance on θ from both the past experiment and e, and assuming constant noise, $\tau(x) = \tau$, determine what is the best choice of a single future measurement, x_f , in the range $[0, x_{max}]$.
- Consider now the case where we could build a more accurate experiment, a, which could measure y with noise $\tau^* << \tau$, but only if the dependent variable falls around a certain value, y^*. We can model this by writing for the noise of a as:

$$\tau_a^2 = \tau^* \exp\left(\frac{(y - y^*)^2}{2\Delta^2}\right) \tag{3}$$

  We consider the parameters y^* , τ^* , Δ as fixed quantities. Find the optimal location x_f for a new measurement and compare the performance of a with the experiment considered above, e. Discuss how and why the result depends on the ratio $\Delta/\Delta y$, where Δy is the present-day uncertainty in the value of y at the location x_f. Produce numerical results for the following choices: $x_0 = 0.5$, $x_1 = 1.0$, $x_{max} = 3$, $\sigma = 0.1$, $\theta = 1.0$, $\tau = 0.1$, $\tau^* = \sigma/5$, $y^* = 1.5$, $\Delta = 0.1$. Plot the expected utility as a function of Δ (all other quantities constant) and determine the cross-over value for Δ at which the experimental configuration to be preferred changes from e to a.

- Consider now the task of optimizing the future experiment a to distinguish between two models for the data: Model 1 is the linear model above, Model 2 has an additional quadratic term and is given by:

$$y = \theta x + \psi x^2 + \epsilon \tag{4}$$

  with ψ an additional parameter with a suitably chosen prior. As a utility function, adopt the volume of parameter space where the absolute value of the logarithm of the Bayes factor between the two models exceeds the threshold value of 2.5. This means that in such a region of parameter space the outcome of model comparison (using both past data and the future data point from a) would be above the 'moderate evidence' threshold (for either model) under the Jeffreys' scale. Determine the value of y^* (for fixed Δ) that maximizes such a utility function and discuss the result by comparing with maximization of the previous utility function.

2 Bayesian experiment design

Before diving on how to solve the problem, I'll make a small description of what is Bayesian experiment design, followed by placing the problem in the context of experiment design.

Bayesian experiment design deals with designing an experiment with the attempt of maximize a utility function. It could be an experiment with the desire of gaining as much new information as possible (the utility function could be the Kullback-Leibler Divergence between our prior knowledge, and the expected knowledge after performing the experiment), where the quantitative aspects of it can be chosen with the goal in mind.

An example is a drug testing design, in which two possible doses will be tested in two groups of people and a control group. The quantitative aspect to be decided would be the fraction of the total number of people involved who will take each dose. It could be that given prior knowledge, dividing the groups in equal sizes would not give the maximum information. For a wider exposition of these examples and more, see [1].

Main components: A design η has to be chosen from a set $\mathcal{H}$ (e.g. the fraction η_i of people taking the drug i). Given the design, we will have an output y. A stochastic function, with parameters θ, maps the design to the outputs y. We want to optimize the design with respect to the expected value of a utility function U (our expected gain in information from the experiment). Minimizing the Kullback-Leibler Divergence in a linear model leads to a D-optimality criteria, where the function to be optimized is $\Phi = \det|XX^T(\eta) + R|$, where X is the measurements to be taken given the design η, and R is related to a Gaussian prior with covariance matrix $\sigma^2 R$. Starting from different utility functions can also lead to a D-optimality criteria, or to different criteria.

While experiment design attempts to find the probability distribution of η that minimizes the expected utility, our problem in question assumes that the probability function is a delta function $p(\eta) = \delta(\eta)$, and the problem reduces to finding the η that maximises the expected utilizy.

3 Solving point one

Adopt as utility function the inverse of the posterior variance on θ from both the past experiment and e, and assuming constant noise, $\tau(x) = \tau$, determine what is the best choice of a single future measurement, x_f , in the range $[0, x_{max}]$:

The problem states that the model is linear, with ϵ being the noise. I will assume that these noise is the measurement noise he is talking about, so there is an exact relation between y and x, but it will be obscured by the noise of my measurements. Since I'm working with a Bayesian approach, I'll also assume that the past experiment was performed with a Jeffrey's prior, where the variance of the noise is known (since comes from my measurement instrument, that is usually the case).

3.1 Computation of Jeffrey's Prior

The likelihood is a normal distribution, with mean θx, and standard deviation σ,

$$p(y_1|\theta;\sigma^2) \propto e^{\frac{-(\theta x_1 - y_1)^2}{2\sigma^2}} \tag{5}$$

and then, the Jeffrey's prior is,

$$\begin{aligned} p(\theta) &= \sqrt{-\mathbb{E}\left[\left(\frac{d}{d\theta^2}\log p(y_1|\theta;\sigma)\right)\right]} \\ &= \sqrt{-\mathbb{E}\left[\left(\frac{d}{d\theta^2}\frac{-(\theta x_1 - y_1)^2}{2\sigma^2}\right)\right]} \\ &= \left|\frac{x_1}{\sigma}\right| \end{aligned} \tag{6}$$

3.2 Computation of first and second experiment posteriors

with this computation, the posterior of θ after the first measurement is,

$$\begin{aligned} p(\theta|\{y_1\};\sigma) &\propto \sqrt{\frac{x^2}{\sigma^2}}\exp\left(\frac{-(\theta x_1 - y_1)^2}{2\sigma^2}\right) \\ &\propto \sqrt{\frac{x^2}{\sigma^2}}\exp\left(\frac{-(\theta - \frac{y_1}{x_1})^2}{2\frac{\sigma^2}{x_1^2}}\right) \end{aligned} \tag{7}$$

Using these output as prior for the second measurement, we get the posterior of the first experiment as

$$\begin{aligned} p(\theta|\{y_1,y_2\};\sigma) &\propto \exp\left(\frac{-(\theta - \frac{y_1}{x_1})^2}{2\frac{\sigma^2}{x_1^2}}\right)\exp\left(\frac{-(\theta - \frac{y_2}{x_2})^2}{2\frac{\sigma^2}{x_2^2}}\right) \\ &\propto \exp\left(-\frac{(\theta - m_{1,2})^2}{2\sigma_{1,2}^2}\right), \qquad m_{1,2} = \frac{x_1 y_1 + x_2 y_2}{x_1^2 + x_2^2}, \qquad \sigma_{1,2}^2 = \frac{\sigma^2}{x_1^2 + x_2^2}. \end{aligned} \tag{8}$$

Now, the second experiment has a different standard deviation τ, so the posterior for the second experiment is,

$$\begin{aligned} p(\theta|\{y_1,y_2,y_3\};\sigma,\tau) &\propto \exp\left(-\frac{(\theta - m_{1,2})^2}{2\sigma_{1,2}^2}\right)\exp\left(-\frac{(\theta x_3 - y_3)^2}{2\tau^2}\right) \\ &\propto \exp\left(-\frac{(\theta - m_{3,\tau})^2}{2\sigma_{3,\tau}^2}\right), \qquad m_{3,\tau} = \frac{m_{1,2}\tau^2 + x_3 y_3 \sigma_{1,2}}{\tau^2 + \sigma_{1,2}^2 x_3^2}, \qquad \sigma_{3,\tau}^2 = \frac{\tau^2 + \sigma_{1,2}^2 x_3^2}{\sigma_{1,2}^2 \tau^2}. \end{aligned} \tag{9}$$

3.3 Computation of the Utility function and determination of the best measurement for experiment 2

As stated in the problem, the utility function is the inverse of the variance after performing the two experiments,

$$U(x_3) = \sigma_{3,\tau}^{-2} = \left[\frac{x_1^2 + x_2^2}{\sigma^2} + \frac{x_3^2}{\tau^2}\right] \tag{10}$$

and since it doesn't depend on θ, this is also the expression for the expected utility. Since we want to maximize it, the best value we should use in the range $[0, x_{max}]$ is x_{max}. These choice minimize the variance of the final posterior probability.

4 Solving Point two

Consider now the case where we could build a more accurate experiment, a, which could measure y with noise $\tau^* << \tau$, but only if the dependent variable falls around a certain value, y^*. We can model this by writing for the noise of a as:

$$\tau_a^2 = \tau^* exp\left(\frac{(y-y^*)^2}{2\Delta^2}\right) \tag{11}$$

We consider the parameters y^* , τ^* , Δ as fixed quantities. Find the optimal location x_f for a new measurement and compare the performance of a with the experiment considered above, e. Discuss how and why the result depends on the ratio $\Delta/\Delta y$, where Δy is the present-day uncertainty in the value of y at the location x_f. Produce numerical results for the following choices: x_0 = 0.5, x_1 = 1.0, x_{max} = 3, σ = 0.1, θ = 1.0, τ = 0.1, τ^* = $\sigma/5$, y^* = 1.5, Δ = 0.1. Plot the expected utility as a function of Δ (all other quantities constant) and determine the cross-over value for Δ at which the experimental configuration to be preferred changes from e to a.:

4.1 Computation of the Utility function and optimal location x_f

The utility function is the same as in the previous exercise, but now, τ is a function of y_3, and as a consequence, a function of x_3 and θ.

$$\begin{aligned} U(x_3,\theta) &= \left[\frac{x_1^2+x_2^2}{\sigma^2} + \frac{x_3^2}{\tau(x_3,\theta)^2}\right] \\ &= \left[\frac{x_1^2+x_2^2}{\sigma^2} + \frac{x_3^2}{\tau^*}\exp\left(\frac{-(\theta x_3 - y^*)^2}{2\Delta^2}\right)\right] \end{aligned} \tag{12}$$

Since now the utility function is also a function of θ, we have to compute the expected, over the prior,

$$\begin{aligned} \mathbb{E}[U(x_3,\theta)] &= \frac{x_1^2+x_2^2}{\sigma^2} + \frac{x_3^2}{\tau^*}\int_{-\infty}^{\infty}\exp\left(\frac{-(\theta x_3-y^*)^2}{2\Delta^2}\right)p(\theta|\{y_1,y_2\},\sigma)d\theta \\ &= \frac{x_1^2+x_2^2}{\sigma^2} + \frac{x_3^2}{\sqrt{2\pi\tau^{*2}\sigma_{1,2}^2}}\int_{-\infty}^{\infty}\exp\left(\frac{-(\theta x_3-y^*)^2}{2\Delta^2}\right)\exp\left(\frac{-(\theta-m_{1,2})^2}{2\sigma_{1,2}^2}\right)d\theta \\ &= \frac{x_1^2+x_2^2}{\sigma^2} + \frac{x_3^2}{\sqrt{\tau^{*2}(1+\frac{\sigma_{1,2}^2x_3^2}{\Delta^2})}}\exp\left(-\frac{1}{2}\left[\left(\frac{m_{1,2}}{\sigma_{1,2}}\right)^2+\left(\frac{y^*}{\Delta}\right)^2 - \frac{(\Delta^2+x_3^2\sigma_{1,2}^2)(m_{1,2}\Delta^2+x_3y^*\sigma_{1,2}^2)}{\sigma_{1,2}^2\Delta^2}\right]\right) \end{aligned} \tag{13}$$

We want to maximize this expression, as a function of x_3. After deriving, and equating to zero, the exponent and denominator factors cancel, and we end up with

$$\sigma_{1,2}^6x_3^5y^* + \sigma_{1,2}^4\Delta^2x_3^2(x_3y^*-1) + m_{1,2}^2\left(\sigma_{1,2}^2\Delta^2x_3^2+\Delta^4\right) - m_{1,2}\left(\sigma_{1,2}^2\Delta^4x_3^2+\Delta^6\right) = 0 \tag{14}$$

These is a polynomial of degree 5, which can have 5 different zeros, and will depend on the value of all the parameters. No more can be done from an analytical point of view without adding values to the equation.

4.2 Discussion about what to expect

The present day uncertainty in the value of y at the location of x_f is $\tau_a^2(x_f,\theta_{MPE})$, where θ_{MPE} is the maximum posterior estimate of θ after the first experiment. If we know with high accuracy the value of θ, then we would be able to compute an x_f such that y_f is close to y^*. The uncertainty of our prediction would be

$$\Delta y = \sqrt{\mathbb{V}[\theta x_3]} = \sigma_{1,2}x_3 \propto \sigma_{1,2} \tag{15}$$

Since we want to maximize $U(x_3)$, we want $\Delta y/\Delta$ to be as small as possible to make the term in 13 in the denominator, inside the root square, as small as possible, and the overall expression increase. Looking inside the exponent, we can also perceive that making Δ big makes the exponent increase, while the dependence on $\sigma_{1,2}$ is a bit more unclear.

Intuitebly, we can understand that, maximising U is equivalent to minimize the variance. To minimize the variance, we want to have small variance in the prior, as well as small variance in the second experiment, for which, we would need to have a measurement close to y^* for this to hapen. To ensure that this is the case, making Δ big helps, but also decreasing our current uncertainty on y, so our measurements are closer to y^*.

4.3 Testing the dependence on Δ

First I made a simulator, that emulates the results of points one and two, and computes the value of the Utility function as a function of the measurement of the second experiment, x_3.

```python
import numpy as np
import matplotlib.pyplot as plt

def linear_(x, theta):
    """
    Linear function: y = theta * x.

    Parameters:
    x (array): Input values.
    theta (float): Slope parameter.

    Returns:
    array: The result of the linear function.
    """
    return theta * x

def linear_measure(x, theta, sigma):
    """
    Simulate measurements with noise added to the linear function.

    Parameters:
    x (array): Input values.
    theta (float): Slope parameter.
    sigma (float): Standard deviation of the noise.

    Returns:
    array: The noisy measurements.
    """
    return linear_(x, theta) + np.random.normal(loc=0, scale=np.sqrt(sigma), size=len(x))

def u1(x0, x1, sigma, tau, xmax):
    """
    Compute the Utility function u1 with parameters x0, x1, sigma, tau, and xmax.

    Parameters:
    x0 (float): Input value.
    x1 (float): Input value.
    sigma (float): A parameter of the model.
    tau (float): A parameter of the model.
    xmax (float): The maximum value for the model.

    Returns:
    float: The result of the model function.
    """
    return ((x1**2 + x0**2) / sigma + xmax / tau)

def sigma1_(x1, x0, sigma):
    """
    Compute sigma_12 from x1, x0, and sigma.

    Parameters:
    x1 (float): Input value.
```

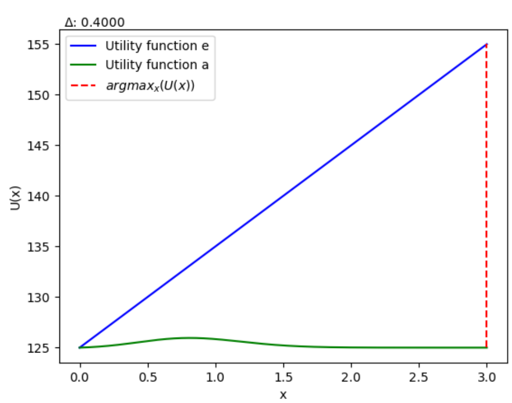
```
    x0 (float): Input value.
    sigma (float): A parameter of the model.

    Returns:
    float: The standard deviation after the first experiment.
    """
    return sigma / (x1**2 + x0**2)

def m1_(x1, x0, y1, y0):
    """
    Compute m_12.

    Parameters:
    x1 (float): Input value.
    x0 (float): Input value.
    y1 (float): Input value.
    y0 (float): Input value.

    Returns:
    float: The mean of the prior after the first experiment.
    """
    return (x1 * y1 + x0 * y0) / (x1**2 + x0**2)

def u2(x0, x1, sigma, tau, xmax, sigmaB, y, m1, sigma1):
    """
    Compute the utility function u2.

    Parameters:
    x0 (float): Input value.
    x1 (float): Input value.
    sigma (float): A parameter of the model.
    tau (float): A parameter of the model.
    xmax (float): The maximum value for the model.
    sigmaB (float): A parameter of the model.
    y (float): A parameter of the model.
    m1 (float): A computed parameter.
    sigma1 (float): A computed parameter.

    Returns:
    float: The utility funciton for model 2.
    """
    sqrt = (1 + (xmax**2) * sigma / sigmaB) ** (-1/2)
    exp = np.exp(-(0.5) * ((m1**2 / sigma1) + (y**2 / sigmaB) - (m1 / sigma1 + xmax *
    y / np.sqrt(sigmaB))**2 / (1 / sigma1 + xmax**2 / sigmaB)))
    return ((x1**2 + x0**2) / sigma + (xmax / tau) * sqrt * exp)
```

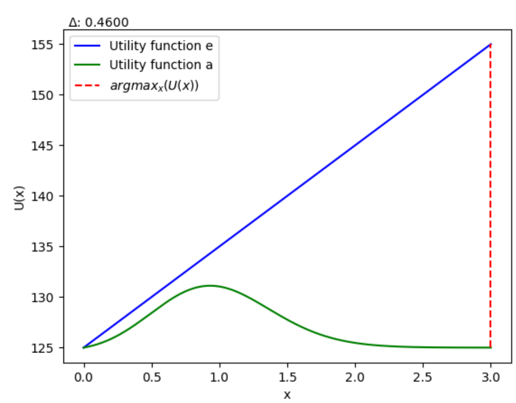
Then, I used the suggested values, and did plots of how U changed as a function of Δ,

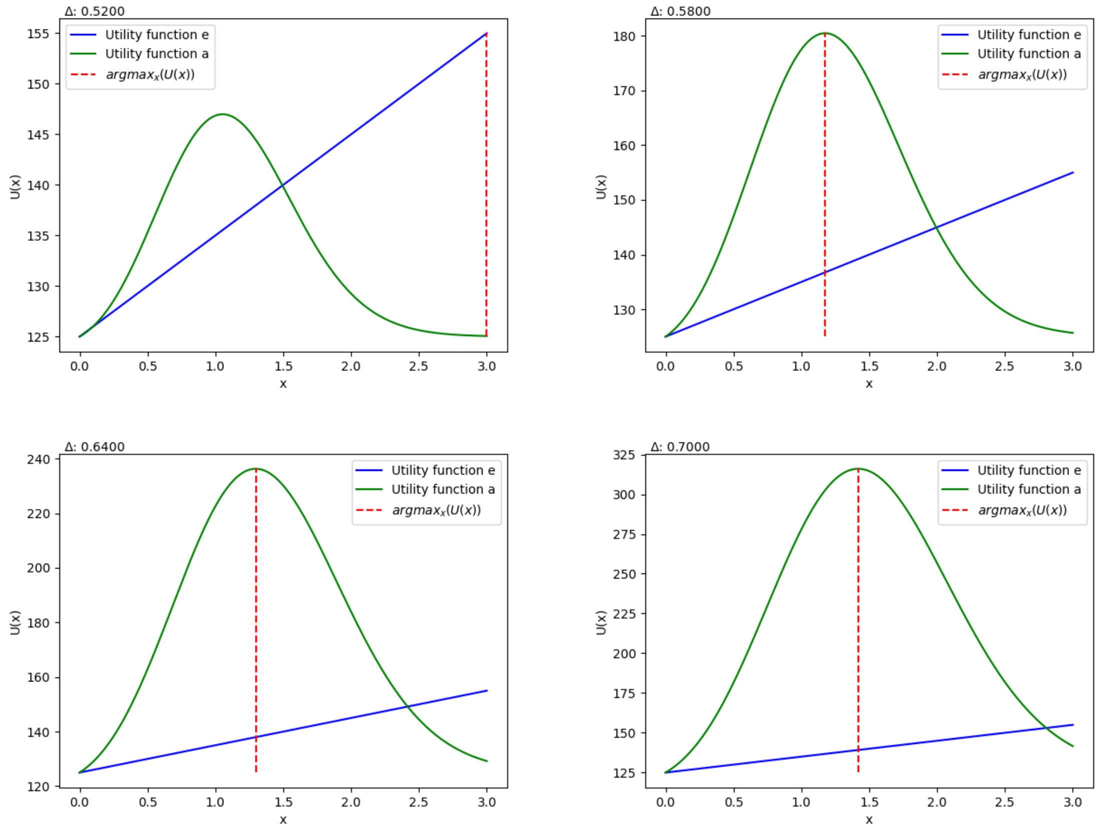
```
    def plot_u(x0_in, x1_in, sigma_in, tau_in, tauS_in, x_in, Delta_in, yS_in, m1_in,
    sigma1_in, theta_in):
    """
    Plot the functions u1 and u2 on the same graph.

    Parameters:
    x0_in (float): Input value.
    x1_in (float): Input value.
    sigma_in (float): A parameter of the model.
    tau_in (float): A parameter of the model.
    tauS_in (float): A parameter of the model.
    x_in (array): Array of x values.
    sigmaB_in (float): A parameter of the model.
    yS_in (float): A parameter of the model.
    m1_in (float): A computed parameter.
    sigma1_in (float): A computed parameter.
    theta_in (float): A parameter of the model.
    """
    ue_ = u1(x0_in, x1_in, sigma_in, tau_in, x_in)
    xmaxe = np.argmax(ue_)  # Find the index where u1 is maximized
    ua_ = u2(x0_in, x1_in, sigma_in, tauS_in, x_in, Delta_in, yS_in, m1_in, sigma1_in)
    xmaxa = np.argmax(ua_)  # Find the index where u2 is maximized

    if np.max(ue_) > np.max(ua_):
```

```python
        xmax = x_in[xmaxe]
    else:
        xmax = x_in[xmaxa]

    # Print the result of a particular formula

    ymin = np.min([np.min(ue_), np.min(ua_)])
    ymax = np.max([np.max(ue_), np.max(ua_)])

    #plt.title('Delta: {}'.format(Delta_in))  # Set plot title with the sigmaB value
    plt.plot(x_in, ue_, label='Utility function e', color='blue')  # Plot u1 (ue)
    plt.plot(x_in, ua_, label='Utility function a', color='green')  # Plot u2 (ua)
    plt.text(.01, 1., '$\Delta$' + ': {:.4f}, Teoretical cross-over: {:.4f}'.format(Delta_in,(theta_in * xmax - yS_in) ** 2 / (2 * np.log(tau_in / tauS_in))), ha='left', va='bottom', transform=plt.gca().transAxes,fontsize=10)  # Add text to the plot
    plt.vlines(xmax, ymin=ymin, ymax=ymax, linestyles='dashed', color='red', label='$argmax_x(U(x))$')  # Add vertical line at xmax
    plt.xlabel('x')
    plt.ylabel('U(x)')
    plt.legend()  # Show legend
    plt.savefig('{:.3f}.png'.format(Delta_in))  # Display the plot

# Define constants for the simulation
x0 = 0.5
x1 = 1
xmax = 3
sigma = 0.1 ** 2
theta = 1
tau = 0.1
tauS = sigma / 5 # tau star
yS = 1.5 #y star

# Generate noisy measurements for y0 and y1
y0, y1 = linear_measure(np.array([x0, x1]), theta, sigma)

# Loop through different values of sigmaB and plot the results
for Delta_sqrt in np.linspace(0.1, 1.5, 6):
    Delta = Delta_sqrt ** 2
    sigma1 = sigma1_(x1, x0, sigma)
    m1 = m1_(x1, x0, y1, y0)
    ue = u1(x0, x1, sigma, tau, xmax)
    ua = u2(x0, x1, sigma, tauS, xmax, Delta, yS, m1, sigma1)
    x = np.linspace(0, 3, 1000)
    plot_u(x0, x1, sigma, tau, tauS, x, Delta, yS, m1, sigma1, theta)  # Call plotting function
```

The result is,







The cross-over is with a value of Δ between 0.52 and 0.58.

5 Solving Point three

Consider now the task of optimizing the future experiment a to distinguish between two models for the data: Model 1 is the linear model above, Model 2 has an additional quadratic term and is given by:

$$y = \theta x + \psi x^2 + \epsilon \tag{16}$$

with ψ an additional parameter with a suitably chosen prior. As a utility function, adopt the volume of parameter space where the absolute value of the logarithm of the Bayes factor between the two models exceeds the threshold value of 2.5. This means that in such a region of parameter space the outcome of model comparison (using both past data and the future data point from a) would be above the 'moderate evidence' threshold (for either model) under the Jeffreys' scale. Determine the value of y^* (for fixed Δ) that maximizes such a utility function and discuss the result by comparing with maximization of the previous utility function.

5.1 Computation of the posterior probabilityes

Now we have two different hypotheses, under which, we would have different posteriors for the parameter θ, and ψ after the second experiment. To get the posterior of θ under model 2, we can make the change of variable $\hat{y} = y - \psi x^2$, and then,

$$p(\theta|\{y_1, y_2, y_3\}, \psi; \sigma, \tau) \propto \exp\left(-\frac{(\theta - m_{3,\tau,\psi})^2}{2\sigma^2_{3,\tau,\psi}}\right), \quad m_{3,\tau,\psi} = \frac{m_{1,2,\psi}\tau^2 + x_3\hat{y}_3\sigma_{1,2,\psi}}{\tau^2 + \sigma^2_{1,2,\psi}x_3^2}, \quad \sigma^2_{3,\tau,\psi} = \frac{\tau^2 + \sigma^2_{1,2,\psi}x_3^2}{\sigma^2_{1,2,\psi}\tau^2}. \tag{17}$$

where $\sigma_{1,2,\psi}$ is the same as $\sigma_{1,2}$, but substituting y with $\hat{y}$.

For ψ, the expression would be the same as long as the change of variables $\hat{x} = x^2$ and $\tilde{y} = y - \theta x$ are performed. The original prior for ψ was also Jeffrey's prior, with the same change of coordinates. If the marginals are required, then integration over the other parameter would be required.

5.2 Computation of the Bayes Factor

The log of the Bayes factor is,

$$\mathrm{logB} = \log p(d|M2) - \log p(d|M1) \tag{18}$$

Where

$$\begin{aligned} \log p(d|M) &= \int p(d|\theta, M)p(\theta)d\theta \\ p(d|\theta, M) &= \frac{1}{(2\pi)^{\frac{3}{2}}\sigma^2\tau} \exp\left(-\frac{1}{2}\sum_{i=1}^{3}\frac{(y_i - \theta x_i)^2}{\sigma_i^2}\right), \qquad \sigma_i = (\sigma, \sigma, \tau), \qquad y_i = \hat{y}_i \quad \text{for M = M2} \end{aligned} \tag{19}$$

To compute these, I need to marginalize over the model parameters. With a bit of patience, since all the computations are Gaussian integrals, the problem can be solved analytically,

$$p(d|M_1) = \frac{1}{(2\pi)\sigma\tau}\frac{1}{\sqrt{\sum_{i=1}^{3}\frac{x_i^2}{\sigma_i^2}}} \exp\left(-\frac{1}{2}\left[\sum_{i=1}^{3}\frac{y_i^2}{\sigma_i^2} - \frac{\left(\sum_{i=1}^{3}\frac{x_i y_i}{\sigma_i^2}\right)^2}{\sum_{i=1}^{3}\frac{x_i^2}{\sigma_i^2}}\right]\right) \tag{20}$$

$$p(d|M_2) = \frac{1}{\sqrt{2\pi}\sigma\tau}\frac{1}{\sqrt{\sum_{i=1}^{3}\frac{x_i^2}{\sigma_i^2}}}\sqrt{\frac{1}{\sum_{i=1}^{3}\frac{x_i^4}{\sigma_i^2}}} \exp\left(-\frac{1}{2}\left[\sum_{i=1}^{3}\frac{y_i^2}{\sigma_i^2} - \frac{\left(\sum_{i=1}^{3}\frac{x_i y_i}{\sigma_i^2}\right)^2}{\sum_{i=1}^{3}\frac{x_i^2}{\sigma_i^2}}\right]\right) \tag{21}$$

$$B = \frac{p(d|M_2)}{p(d|M_1)} = \frac{1}{\sqrt{\sum_{i=1}^{3}\frac{x_i^4}{\sigma_i^2}}} \tag{22}$$

$$logB = -\frac{1}{2}\log\left(\frac{x_1}{\sigma} + \frac{x_2}{\sigma} + \frac{x_3}{\tau}\right) \tag{23}$$

The problem talks about the volume of the parameters. First, I will work on this expression. I want the value of x_3 that maximizes the value of the logB if the data was generated with model 1, in such a way that I can optimally distinguish between the two models.

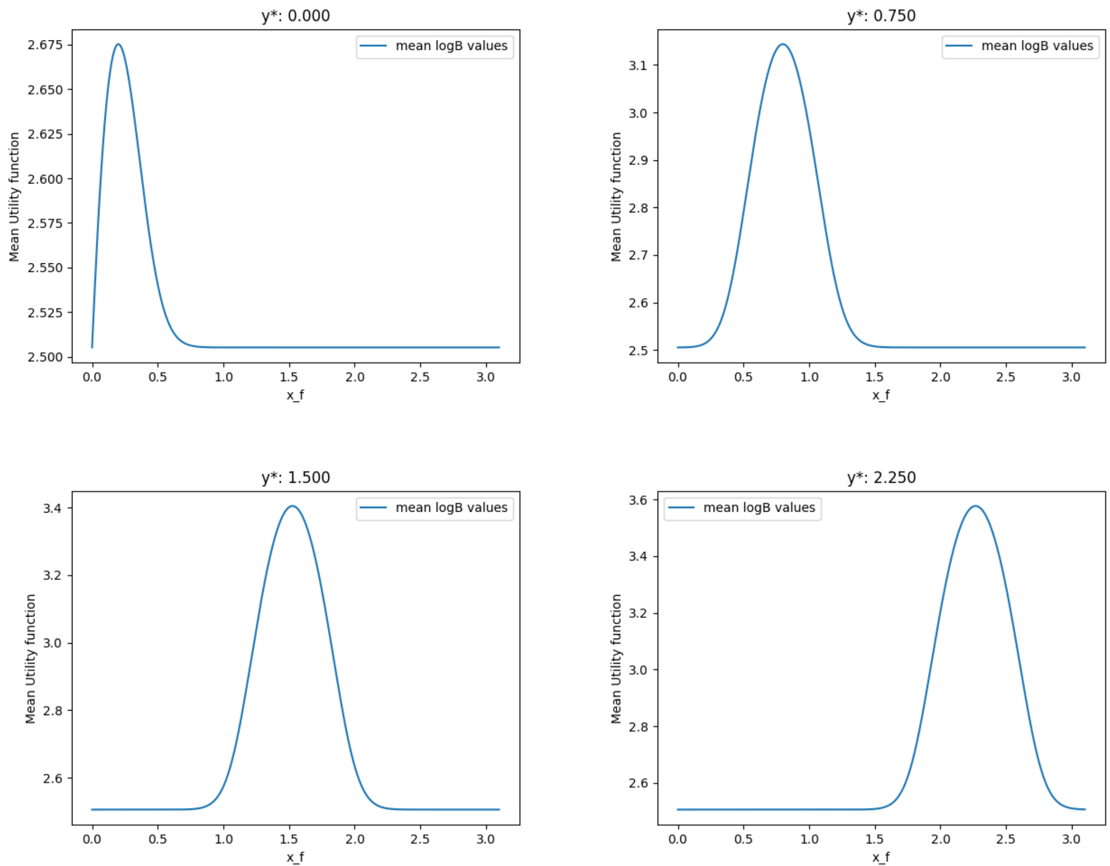
To test the computation, I made again a simulation, and compute many utility functions, but these time, changing y^*,

```python
import numpy as np
import matplotlib.pyplot as plt

def linear_(x, theta):
    """
    Linear function: y = theta * x.

    Parameters:
    x (array or float): Input values.
    theta (float): Slope parameter.

    Returns:
    array or float: The result of the linear function.
    """
    return theta * x

def linear_measure(x, theta, sigma):
    """
    Simulate measurements with noise added to the linear function.

    Parameters:
    x (array or float): Input values.
    theta (float): Slope parameter.
```

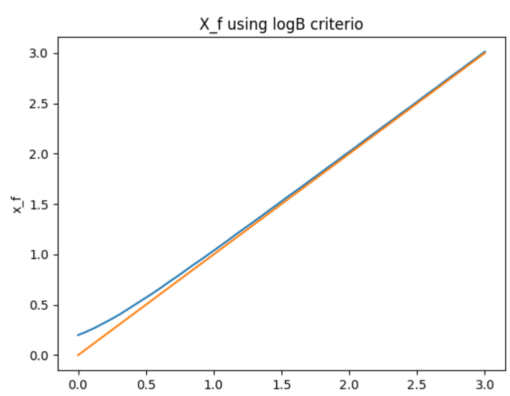
```
    sigma (float): Standard deviation of the noise.

    Returns:
    array or float: The noisy measurements.
    """
    try:
        size = len(x)
    except TypeError:
        size = 1
    return linear_(x, theta) + np.random.normal(loc=0, scale=np.sqrt(sigma), size=size
    )


def tau_a(tauS, yS, y, Delta):
    """
    Compute the adjusted time constant tau_a.

    Parameters:
    tauS (float): Reference time constant.
    yS (float): Reference y value.
    y (array or float): Input y values.
    Delta (float): Scaling factor.

    Returns:
    array or float: The adjusted time constant values.
    """
    return tauS * np.exp((y - yS) ** 2 / (2 * Delta ** 2))

def logB_(x1,x2,x3,sigma,tau):
    return -0.5*np.log((x1/sigma)+ (x2/sigma) + (x3/tau))
# Define constants for the simulation
x0 = 0.5
x1 = 1
xmax = 3
sigma = 0.1 ** 2
theta = 1
tauS = sigma / 5  # tau star
Delta = 0.2

x2 = np.linspace(0, 3.1, 1000)

yS_values = np.linspace(linear_(0, theta), linear_(3, theta), 5)
for yS in yS_values:
    tau = tau_a(tauS, yS, linear_(x2, theta), Delta)
    logB = np.zeros(len(x2))
    for _ in range(100):

        logB +=  logB_(x0,x1,x2,sigma,tau)
    logB /= 100

    plt.plot(x2, np.abs(logB),label='mean logB values')
    plt.xlabel('x_f')
    plt.ylabel('Mean Utility function')

    plt.title('y*: {:.3f}'.format(yS))
    plt.legend()
    plt.savefig(f"logB_y_{yS:.3f}.png")
    plt.close()

# Compute and plot optimal experiment results
max_x = []
ySsss = np.linspace(linear_(0, theta), linear_(3, theta), 50)
for yS in ySsss:
    tau = tau_a(tauS, yS, linear_(x2, theta), Delta)
    logB = np.zeros(len(x2))
    for _ in range(100):
        logB += logB_(x0,x1,x2,sigma,tau)
    logB /= 100
    max_x.append(x2[np.argmax(np.abs  (logB))])

plt.plot(ySsss, max_x, label='Optimal Experiment')
```

```
plt.xlabel('y*')
plt.ylabel('x_f')
plt.title('X_f using logB criterio')
plt.plot(ySsss, ySsss / theta)
plt.savefig('y_x_f.png')
```

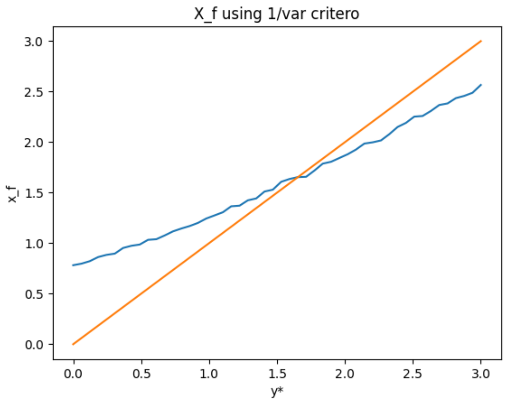
The output images for the Utility functions using different $y*$ are,



The interesting part is that, the maximum of the utility function is align with the value with $y*$, in other words, the optimal experimental measurement x_f is such that y_f is equal to y^*,



We can compare these result with the one obtained by using the 1/var criteria, and we obtain,



What we notice is that both criteria are behaving similarly with respect to $y*$.

5.3 Computation of the Volume on the parameters

To compute the x_f that maximizes the volume over the parameters, that makes the absolute value of the log of the bayes factor greater than 2.5, I estimate a proxi of the volume as the number of samples that have a value of the log bayes greater than 2.5 with respect of a total number of samples, using Gibbs sampling.

I know that,

$$\begin{aligned}\log B &= \log p(d|M2) - \log p(d|M2) \\ &= \log \int_{\Sigma_2} p(d|M2,\theta_2)p(\theta|M2)d\theta_2 - \log \int_{\Sigma_1} p(d|M1,\theta_1)p(\theta|M1)d\theta_1 \\ &\approx \log \frac{1}{N}\sum_{i=1}^{N} p(d|M2,\theta_{2,i}) - \log \frac{1}{N}\sum_{i=1}^{N} p(d|M1,\theta_{1,i}), \quad \theta_{2,i} \sim p(\theta|M2), \quad \theta_{1,i} \sim p(\theta|M1)\end{aligned} \tag{24}$$

Then, if I choose N = 1, would be equivalent to approximate logB integrated on a volume close to the samples. Then, I compute this for a number of times, and use the fraction of computations biguer than 2.5 as a proxi of the volume.

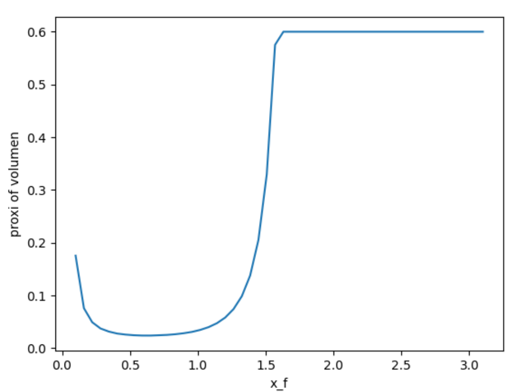
```
import numpy as np
from scipy.stats import norm
import matplotlib.pyplot as plt
from numba import njit

@njit
def linear_(x, theta):
    """
    Linear function: y = theta * x.

    Parameters:
    x (array or float): Input values.
    theta (float): Slope parameter.

    Returns:
    array or float: The result of the linear function.
    """
    return theta * x

def linear_measure(x, theta, sigma):
    """
    Simulate measurements with noise added to the linear function.

    Parameters:
    x (array or float): Input values.
    theta (float): Slope parameter.
    sigma (float): Standard deviation of the noise.

```

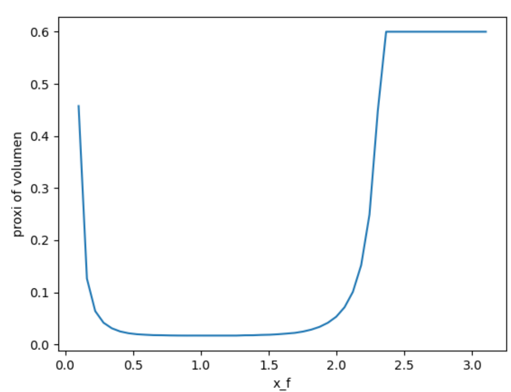
```python
    Returns:
    array or float: The noisy measurements.
    """
    try:
        size = len(x)
    except TypeError:
        size = 1
    return linear_(x, theta) + np.random.normal(loc=0, scale=np.sqrt(sigma), size=size)

@njit
def sigma12(sigma,x1,x2):
    return sigma**2 / (x1**2 + x2**2)

@njit
def mu12(x1,x2,y1,y2):
    return (x1 * y1 + x2 * y2) / (x1**2 + x2**2)

@njit
def mu3(x1,x2,x3,y1,y2,y3,sigma,tau):
    num = mu12(x1,x2,y1,y2)*tau**2 + x3*y3*np.sqrt(sigma12(sigma,x1,x2))
    den = tau**2 + sigma12(sigma,x1,x2)*x3**2
    m_3_tau = num / den
    return m_3_tau

@njit
def sigma3(x1,x2,x3,sigma,tau):
    num = tau**2 + sigma12(sigma,x1,x2)*x3**2
    den = sigma12(sigma,x1,x2)*tau**2
    return num/den

@njit
def theta_probability_model1(x1, x2, x3, y1, y2, y3, sigma,tau,theta):
    mu = mu3(x1,x2,x3,y1,y2,y3,sigma,tau)
    sigma_ = sigma3(x1,x2,x3,sigma,tau)
    return norm.pdf(theta, loc=mu, scale=np.sqrt(sigma_))

@njit
def theta_probability_model2(x1, x2, x3, y1, y2, y3, sigma,tau,psi,theta):
    y1_hat, y2_hat, y3_hat = y1 - psi * x1**2, y2 - psi * x2**2, y3 - psi * x3**2
    mu = mu3(x1,x2,x3,y1_hat,y2_hat,y3_hat,sigma,tau)
    sigma_ = sigma3(x1,x2,x3,sigma,tau)
    return norm.pdf(theta, loc=mu, scale=np.sqrt(sigma_))

@njit
def probability_model1_data_and_theta(x1, x2, x3, y1, y2, y3, sigma,tau,theta):
    #den = (2*np.pi)**(3/2) * (sigma**2) * tau
    exponent = -0.5*((y1-theta*x1)**2/sigma**2 + (y2-theta*x2)**2/sigma**2 + (y3-theta*x3)**2/tau**2 )
    return exponent

@njit
def probability_model2_data_and_theta_psi(x1, x2, x3, y1, y2, y3, sigma,tau,theta,psi):
    #den = (2*np.pi)**(3/2) * (sigma**2) * tau
    exponent = -0.5*((y1-theta*x1 - psi*x1**2)**2/sigma**2 + (y2-theta*x2 - psi*x2**2)**2/sigma**2 + (y3-theta*x3 - psi*x3**2)**2/tau**2 )
    return exponent

@njit
def tau_a(tauS, yS, y, Delta):
    """
    Compute the adjusted time constant tau_a.

    Parameters:
    tauS (float): Reference time constant.
    yS (float): Reference y value.
    y (array or float): Input y values.
    Delta (float): Scaling factor.

    Returns:
```

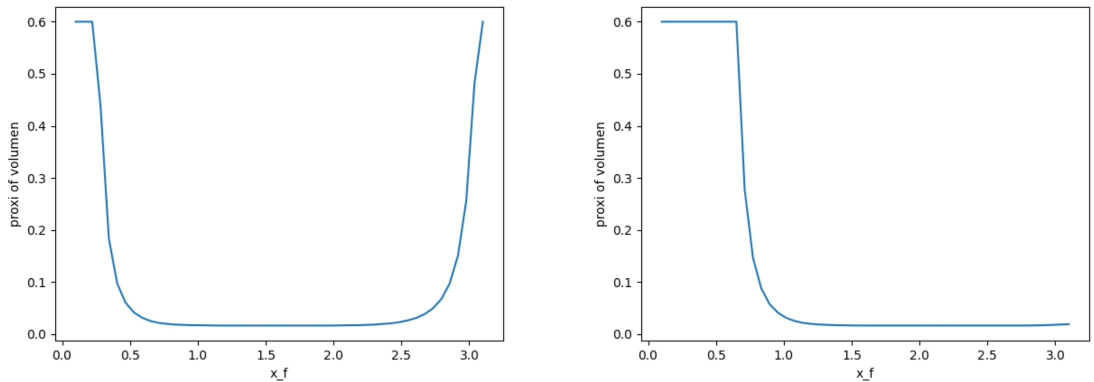
```
    array or float: The adjusted time constant values.
    """
    return tauS * np.exp((y - yS) ** 2 / (2 * Delta ** 2))

def theta_sampling_model1(x1, x2, x3, y1, y2, y3, sigma, tau,num_samples=1000,burn_in=50):

    mu = mu3(x1,x2,x3,y1,y2,y3,sigma,tau)
    sigma_ = sigma3(x1,x2,x3,sigma,tau)

    # Compute probability density of theta
    return np.random.normal(mu, np.sqrt(sigma_),num_samples)[burn_in:]

def gibbs_sampling_model2(x1, x2, x3, y1, y2, y3, sigma, tau,ys,num_samples=1000,burn_in=50):

    samples_theta = np.zeros(num_samples)
    samples_psi = np.zeros(num_samples)

    # Initialize psi randomly
    psi = np.random.normal(ys, 1)

    for i in range(num_samples):
        # Compute the transformed y-values
        y1_hat, y2_hat, y3_hat = y1 - psi * x1**2, y2 - psi * x2**2, y3 - psi * x3**2

        mu = mu3(x1,x2,x3,y1_hat,y2_hat,y3_hat,sigma,tau)
        sigma_ = sigma3(x1,x2,x3,sigma,tau)

        theta = np.random.normal(mu, np.sqrt(sigma_))

        y1_hat, y2_hat, y3_hat = y1 - theta * x1, y2 - theta * x2, y3 - theta * x3
        x1_hat, x2_hat, x3_hat = x1**2,x2**2,x3**2

        mu = mu3(x1_hat,x2_hat,x3_hat,y1_hat,y2_hat,y3_hat,sigma,tau)
        sigma_ = sigma3(x1_hat,x2_hat,x3_hat,sigma,tau)

        psi = np.random.normal(mu, np.sqrt(sigma_))

        samples_theta[i] = theta
        samples_psi[i] = psi

    # Remove burn-in period
    return samples_theta[burn_in:], samples_psi[burn_in:]

@njit
def p_greater_25(logB_):
    return len(logB_[np.abs(logB_) > 2.5])/len(logB_)

def compute_proxi_volume(x1, x2, x3, y1, y2,sigma,tauS, yS,Delta):
    volumes = []
    y3 = linear_(x3,theta)
    taus = tau_a(tauS, yS, y3, Delta)

    for i in range(len(x3)):
        thetas_model1 = theta_sampling_model1(x1, x2, x3[i], y1, y2, y3[i], sigma, taus[i])
        thetas_model2,psis_model2 = gibbs_sampling_model2(x1, x2, x3[i], y1, y2, y3[i], sigma, taus[i],yS)
        p_M1 = probability_model1_data_and_theta(x1, x2, x3[i], y1, y2, y3[i], sigma, taus[i],thetas_model1)
        p_M2 = probability_model2_data_and_theta_psi(x1, x2, x3[i], y1, y2, y3[i], sigma,taus[i],psis_model2,thetas_model2)
        logBs = p_M2 - p_M1
        volumes.append(p_greater_25(logBs))
    return np.array(volumes)

x0 = 0.5
x1 = 1
xmax = 3
sigma = 0.1 ** 2
```

```python
theta = 1
tauS = sigma / 5  # tau star
Delta = 0.5
x2 = np.linspace(0.1, 3.1, 50)


yS_values = np.linspace(linear_(0, theta), linear_(3, theta), 10)
volumes = []

from tqdm import tqdm

for j in tqdm(range(len(yS_values))):
    proxi_volumes = np.zeros(len(x2))
    for i in range(30):
        proxi_volumes += compute_proxi_volume(x0, x1, x2, linear_measure(x0, theta, sigma), linear_measure(x1, theta, sigma),sigma,tauS, yS_values[j],Delta)
    volumes.append(proxi_volumes/len(proxi_volumes))

    plt.xlabel('x_f')
    plt.ylabel('proxi of volumen')
    plt.plot(x2,proxi_volumes/len(proxi_volumes))
    plt.savefig('vol_{:.3f}.png'.format(yS_values[j]))
    plt.close()

volumes = np.array(volumes)
x_fs = []
for v in volumes:
    x_fs.append( [ x2[np.argmax(v)] , x2[-np.argmax(v[::-1])] ])

x_fs = np.array(x_fs)
plt.plot(yS_values,x_fs[:,0],'o',label='first maximum value using Volume criterio',
    color='black')
plt.plot(yS_values,x_fs[:,1],'o',label = 'second maximum value using Volume criterio',
    color='gray')
plt.xlabel('y*')
plt.ylabel('x_f')

plt.plot(yS_values, yS_values/theta,'--',label = 'most precise measurement x*',color='
    red')
plt.show()
```

The shape of the volume as a function of y^* is shown in the next images,







References

[1] Kathryn Chaloner and Isabella Verdinelli. Bayesian experimental design: A review. *Statistical science*, pages 273–304, 1995.