Task 10.1

Implement the system

$$ y_k + a y_{k-1} = b u_{k-1} + e_k $$
where:

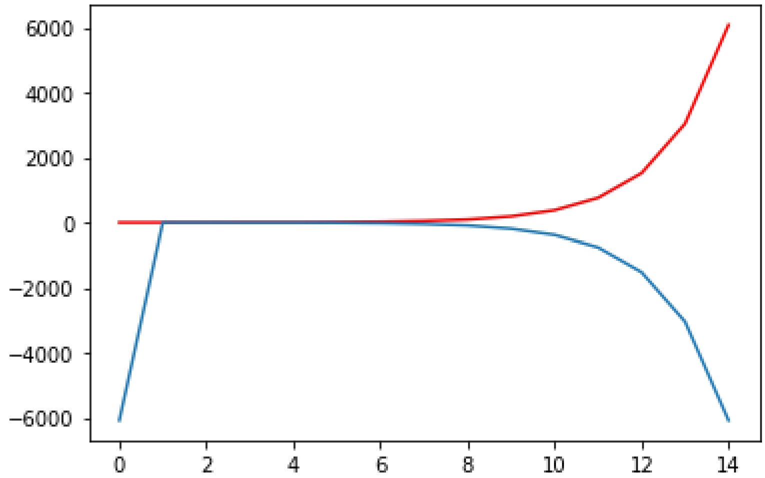
- e_k is white Gaussian zero-mean noise with variance λ^2
- the input is computed through a state-feedback law $u_k = - K y_k + r_k$ with r_k a reference signal
- K is so that the closed loop system in the absence of the reference signal is asymptotically stable, and the mode of the system is non-oscillatory
- r_k, for the sake of this assignment, is another white Gaussian zero-mean noise with variance σ^2

In [44]:

```
# importing the right packages
import numpy as np
import matplotlib.pyplot as plt
import scipy.optimize as optimize
```

In [45]:

```
# Function to simulate the system
def simulate(a, b, K, lambda2, sigma2, y0, N, reference_frequency=0):
    # Storage allocation
    y = np.zeros(N)
    u = np.zeros(N)

    # system noises
    e = np.random.normal(0, np.sqrt(lambda2),  N)
    r = np.random.normal(0, np.sqrt(sigma2), N) + \
        np.sin(reference_frequency * np.arange(N))

    # Saving the initial condition
    y[0] = y0
    u[0] = - K * y0 + r[0]

    # Cycle on the steps
    for t in range(1, N):
        y[t] = b * u[t-1] + e[t] - a * y[t-1]
        u[t] = u[0] = - K * y[t] + r[t]

    return [y, u]

sim = simulate(-1, -1, 1, 10, 0, 0, 15)
plt.plot(sim[0], 'r')
plt.plot(sim[1])
plt.show()
```



In [46]:

```
# define also a function for doing poles allocation, considering
# that eventually if the reference is absent then the ODE is
#
# y_k + (a + b K) y_{k-1} = e_k
# pole at H(q) = 1/A(q), A(q) = (a + bK)q^-1, => 1 + a + bK = 1 - desired = 0
def compute_gain(a, b, desired_pole_location):
    K = - (desired_pole_location + a) / b
    return K
```

In [47]:

```
# plotting of the impulse response
def plot_impulse_response(a, b, plot_args, figure_number=1000):
    # ancillary quantities
    k = range(0, 50)
    y = b * np.power(-a, k)

    # plotting the various things
    plt.figure(figure_number, **plot_args)
    plt.plot(y, 'r-', label='u')
    plt.xlabel('time')
    plt.title('impulse response relative to a = {} and b = {}'.format(a, b))
```

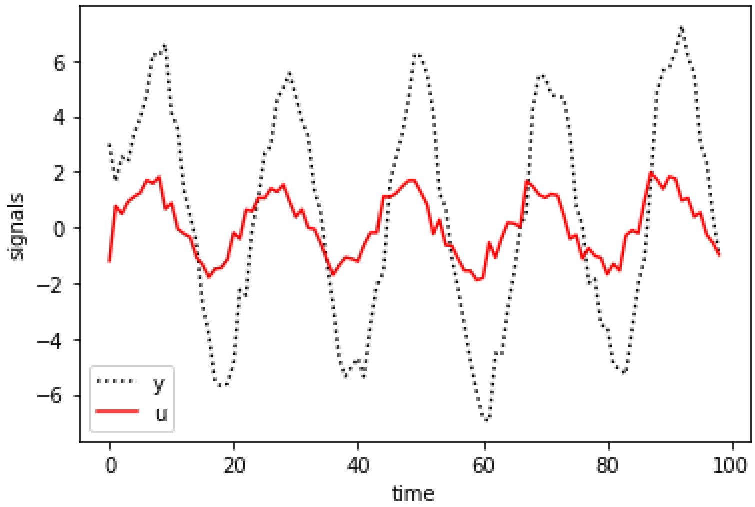
In [39]:

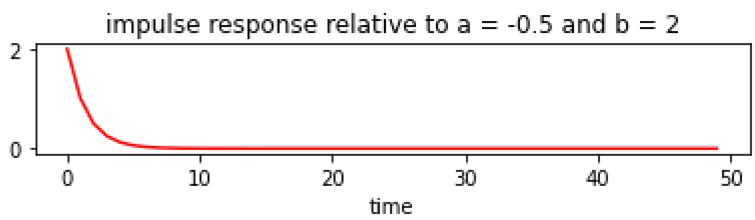
```
# define the system parameters
a = -0.5
b = 2
K = compute_gain(a, b, 0.7)

# noises
lambda2 = 0.1 # on e
sigma2 = 0.1 # on r

# initial condition
y0 = 3

# number of steps
N = 100
```

In [26]:

```
# DEBUG - check that things work as expected

# run the system
y, u = simulate(a, b, K, lambda2, sigma2, y0, N, 0.3)

# plotting the various things
plt.figure()
plt.plot(y[:-1], 'k:', label='y')
```

```
plt.plot(u[:-1], 'r-', label='u')
plt.xlabel('time')
plt.ylabel('signals')
plt.legend();

plot_impulse_response(a, b, {'figsize': (6.4, 1)})
```





Task 10.2

Implement a PEM-based approach to the estimation of the system, assuming to know the correct model structure but not knowing about the existence of the feedback loop given by K.

In [27]:

```
# important: the system is an ARX one, and e_k is Gaussian so PEM = ???
# And given this, how can we simplify things?
```

In [41]:

```
# define the function solving the PEM problem asked in the assignment
def PEM_solver(u, y):
    yu_prev = np.zeros((len(y) - 1, 2))
    yu_prev[:, 0] = y[:-1]
    yu_prev[:, 1] = - u[:-1]
    rhs = - np.array(y[1:])
    # Compute the PEM estimate, should have used normal eqs.
    a_hat, b_hat = np.linalg.lstsq(yu_prev, rhs, rcond=None)[0]
    return a_hat, b_hat
```

In [36]:

```
# compute the solution
a_hat, b_hat = PEM_solver(u, y)

# assess the performance
MSE = np.linalg.norm([a - a_hat, b - b_hat])**2

# print debug info
```

```
print('MSE: {}'.format(MSE))
print('a, b = {}, {}    --    ahat, bhat = {}, {}'.format(a, b, a_hat, b_hat))
```

```
MSE: 0.0003191037492320515
a, b = -0.5, 2   --    ahat, bhat = -0.510246025584857, 1.9853670676572839
```

Task 10.3

Show from a numerical perspective that for $\lambda^2 = 0.1$ (i.e., a constant variance on the process noise) the estimates are consistent.

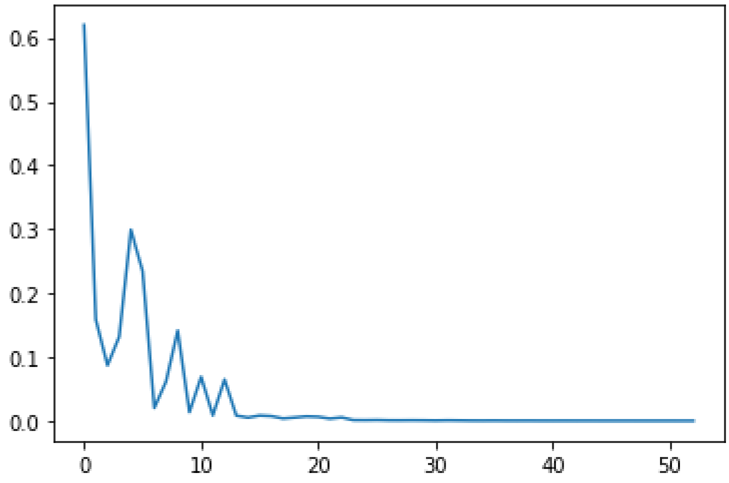
In [65]:

```
# the best way to show this is to do a Monte Carlo approach:
# - for each N, compute the distribution of the estimates
#   around the true parameters
# - increase N and show that this distribution tends to
#   converge to the true parameters

# defining the MC simulation
N_MC_runs = 100
min_order_for_N = 1
max_order_for_N = 4
num_of_N_orders = max_order_for_N - min_order_for_N + 1

# noises and initial condition
lambda2 = 0.1 # on e
sigma2 = 0.1 # on r
y0 = 0

# storage allocation
MSEs = np.zeros((num_of_N_orders, N_MC_runs))
#theta_hats = np.zeros((num_of_sigma2_orders, N_MC_runs, 2))
theta_hats = np.zeros((num_of_N_orders, N_MC_runs, 2))

# cycle on the number of samples
for j, N in enumerate(np.logspace(min_order_for_N, max_order_for_N, num_of_N_
    # MC cycles
    for m in range(N_MC_runs):
        # simulate the system
        y, u = simulate(a, b, K, lambda2, sigma2, y0, int(N), 0.3)
        # compute the solution
        a_hat, b_hat = PEM_solver(u, y)

        # assess the performance
        MSEs[j, m] = np.linalg.norm([a - a_hat, b - b_hat])**2
        # save the results
        theta_hats[j, m, 0] = a_hat
        theta_hats[j, m, 1] = b_hat
```

In [63]:

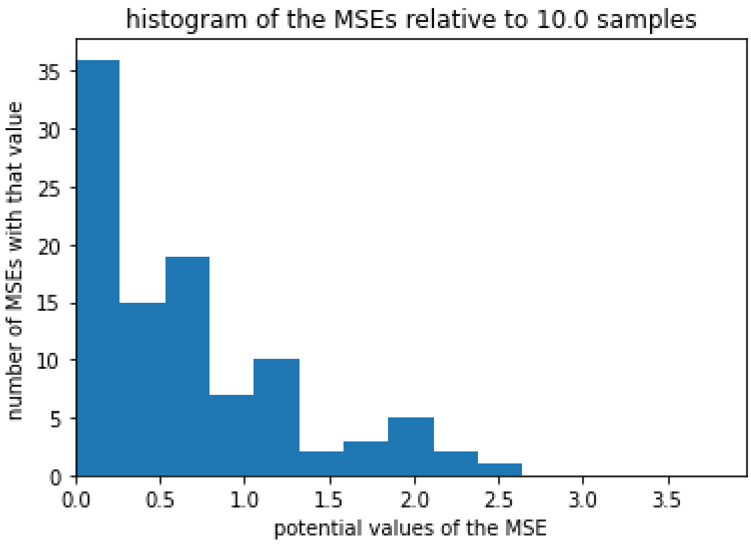
```
plt.plot([np.mean(MSEs[j, :]) for j in range(MSEs.shape[0])])
```

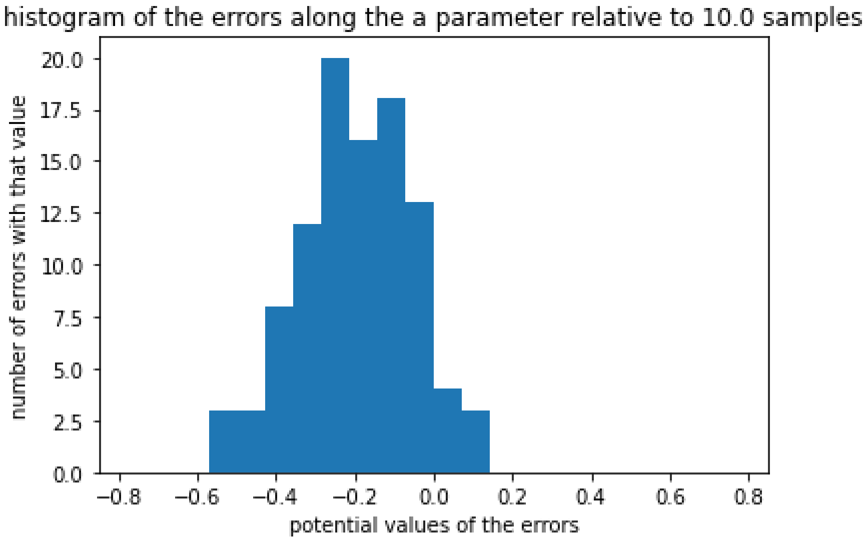
Out[63]: [<matplotlib.lines.Line2D at 0x7f2727df0190>]

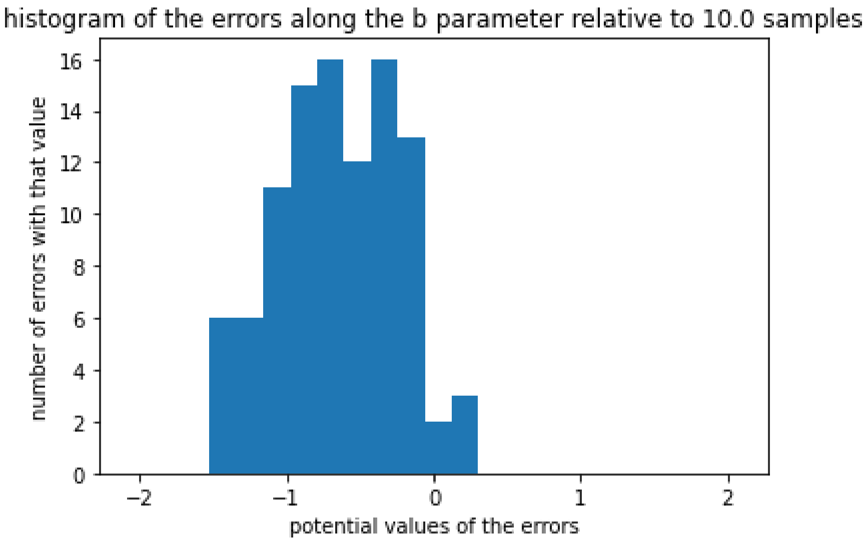


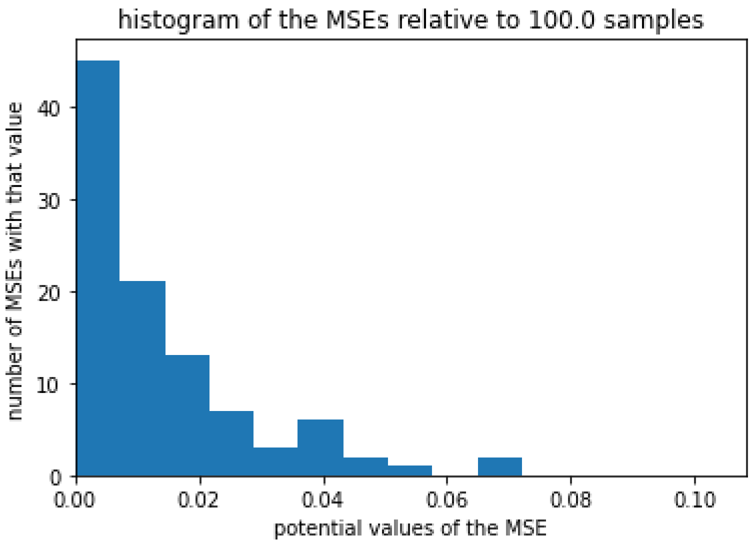
In [66]:

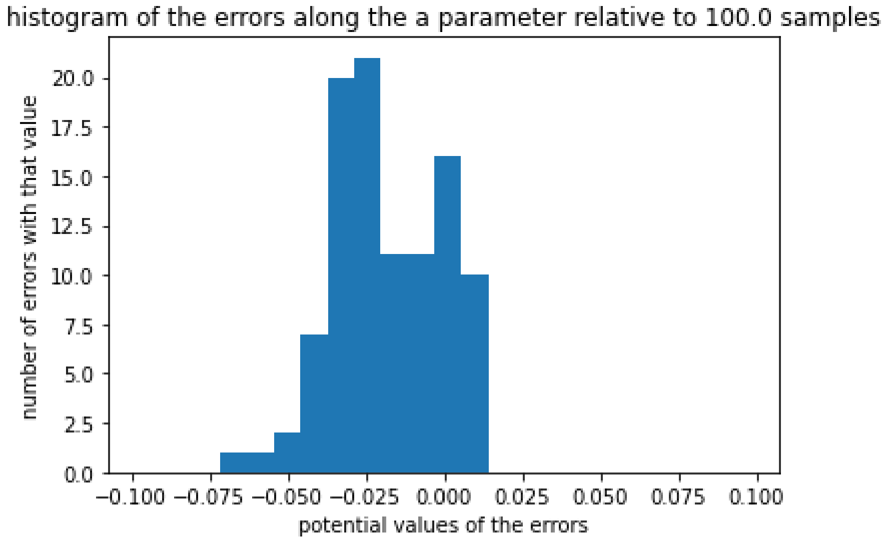
```
# cycle on the number of samples
for j, N in enumerate(np.logspace( min_order_for_N, max_order_for_N, num_of_N
    # plot the histogram of the MSEs relative to this number of samples
    plt.figure(j)
    plt.hist(MSEs[j,:])
    plt.xlim(0, 1.5 * np.max(MSEs[j,:]))
    plt.title('histogram of the MSEs relative to {} samples'.format(N))
    plt.xlabel('potential values of the MSE')
    plt.ylabel('number of MSEs with that value')

    # plot the histogram of the errors along the a parameter
    plt.figure(j + 100)
    x_lim = np.max(np.abs(theta_hats[j,:,0] - a))
    plt.hist(theta_hats[j,:,0] - a)
    plt.xlim(-1.5 * x_lim, 1.5 * x_lim)
    plt.title('histogram of the errors along the a parameter relative to {} s
    plt.xlabel('potential values of the errors')
    plt.ylabel('number of errors with that value')

    # plot the histogram of the errors along the a parameter
    plt.figure(j + 200)
    x_lim = np.max(np.abs(theta_hats[j,:,1] - b))
    plt.hist(theta_hats[j,:,1] - b)
    plt.xlim(-1.5 * x_lim, 1.5 * x_lim)
    plt.title('histogram of the errors along the b parameter relative to {} s
    plt.xlabel('potential values of the errors')
    plt.ylabel('number of errors with that value')
```

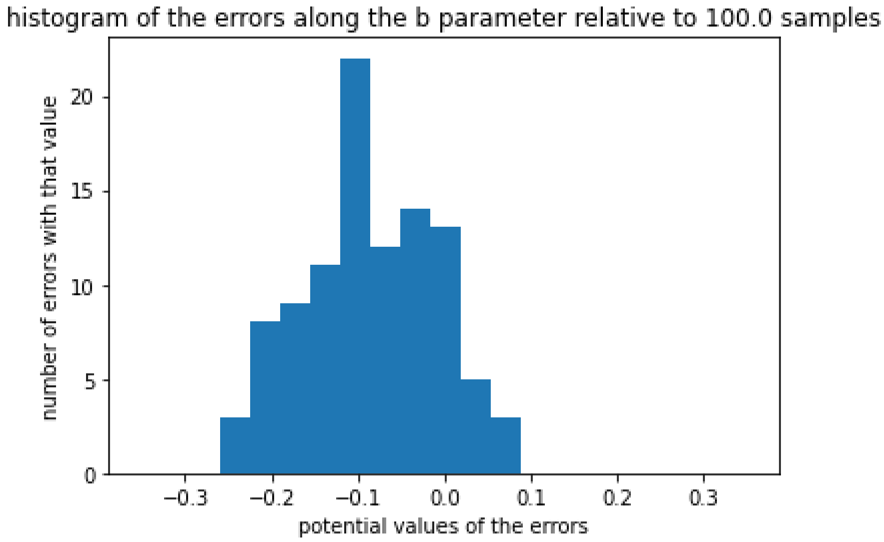


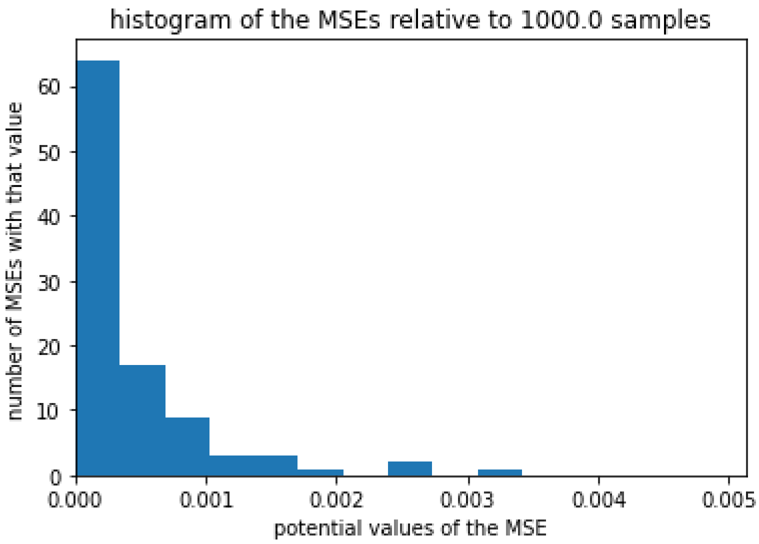

histogram of the errors along the a parameter relative to 10.0 samples
number of errors with that value
potential values of the errors
20.0
17.5
15.0
12.5
10.0
7.5
5.0
2.5
0.0
−0.8
−0.6
−0.4
−0.2
0.0
0.2
0.4
0.6
0.8

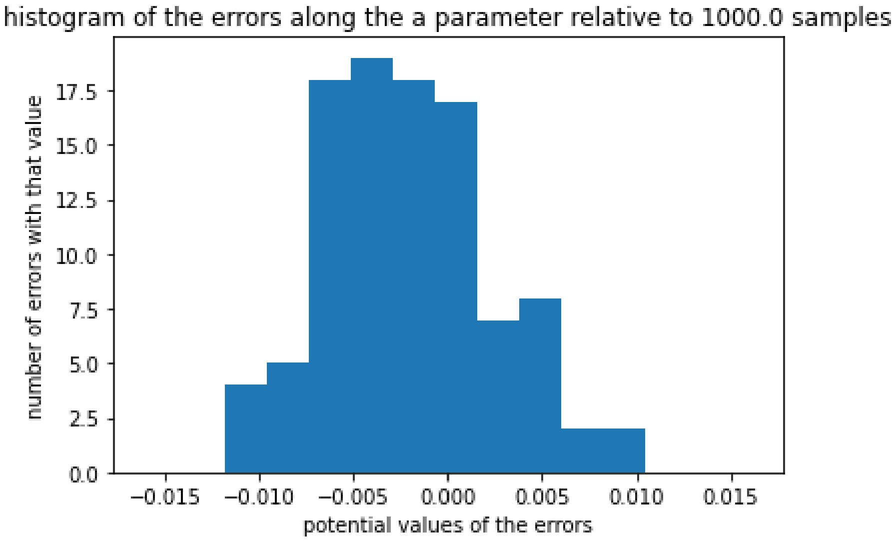

histogram of the errors along the b parameter relative to 10.0 samples
number of errors with that value
potential values of the errors
16
14
12
10
8
6
4
2
0
−2
−1
0
1
2

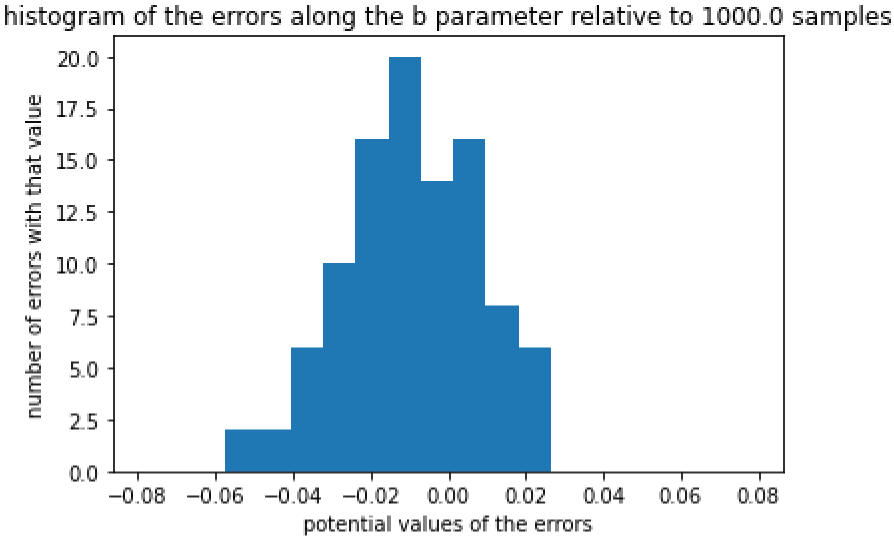

histogram of the MSEs relative to 100.0 samples
number of MSEs with that value
potential values of the MSE
40
30
20
10
0
0.00
0.02
0.04
0.06
0.08
0.10

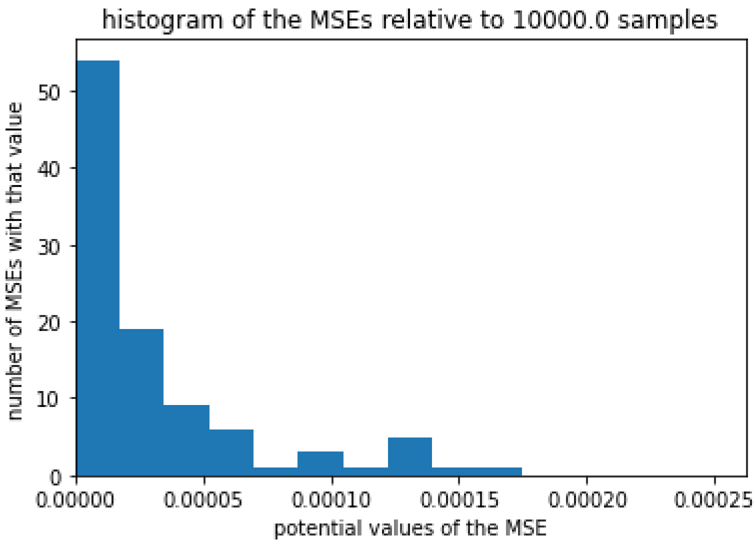

histogram of the errors along the a parameter relative to 100.0 samples
number of errors with that value
potential values of the errors
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
−0.100 −0.075 −0.050 −0.025 0.000 0.025 0.050 0.075 0.100

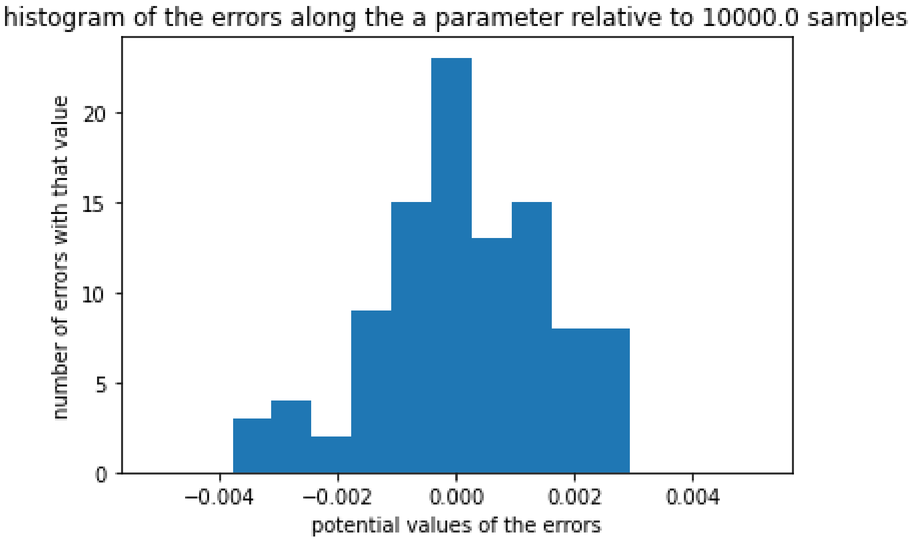

histogram of the errors along the b parameter relative to 100.0 samples
number of errors with that value
potential values of the errors
0
5
10
15
20
−0.3 −0.2 −0.1 0.0 0.1 0.2 0.3

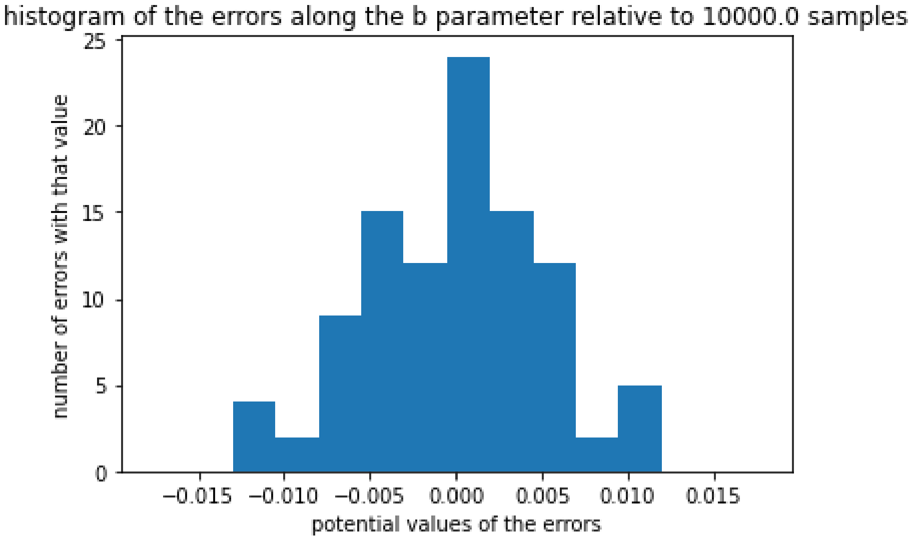

histogram of the MSEs relative to 1000.0 samples
number of MSEs with that value
potential values of the MSE
0
10
20
30
40
50
60
0.000 0.001 0.002 0.003 0.004 0.005


histogram of the errors along the a parameter relative to 1000.0 samples
number of errors with that value
potential values of the errors
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
−0.015
−0.010
−0.005
0.000
0.005
0.010
0.015


histogram of the errors along the b parameter relative to 1000.0 samples
number of errors with that value
potential values of the errors
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
−0.08
−0.06
−0.04
−0.02
0.00
0.02
0.04
0.06
0.08


histogram of the MSEs relative to 10000.0 samples
number of MSEs with that value
potential values of the MSE
0
10
20
30
40
50
0.00000
0.00005
0.00010
0.00015
0.00020
0.00025





Task 10.3

Show that the variances of the estimates though will tend to infinity as $\sigma^2 \rightarrow 0$, i.e., the reference becomes a deterministic known signal.

In [84]:

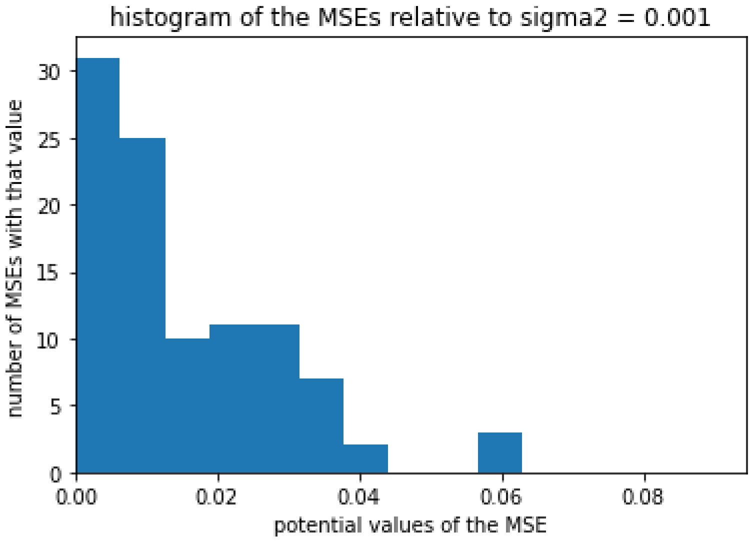
```
# again the best way to show this is to do a Monte Carlo approach:
# - for each sigma2, compute the distribution of the estimates
#   around the true parameters
# - diminish sigma2 and show that this distribution tends to
#   diverge

# defining the MC simulation
N = 100
N_MC_runs = 100
min_order_for_sigma2 = -10
max_order_for_sigma2 = 10
num_of_sigma2_orders = max_order_for_sigma2 - min_order_for_sigma2 + 1

# noises and initial condition
lambda2 = 0.1 # on e
y0 = 0

# storage allocation
MSEs = np.zeros((num_of_sigma2_orders, N_MC_runs))
theta_hats = np.zeros((num_of_sigma2_orders, N_MC_runs, 2))
```

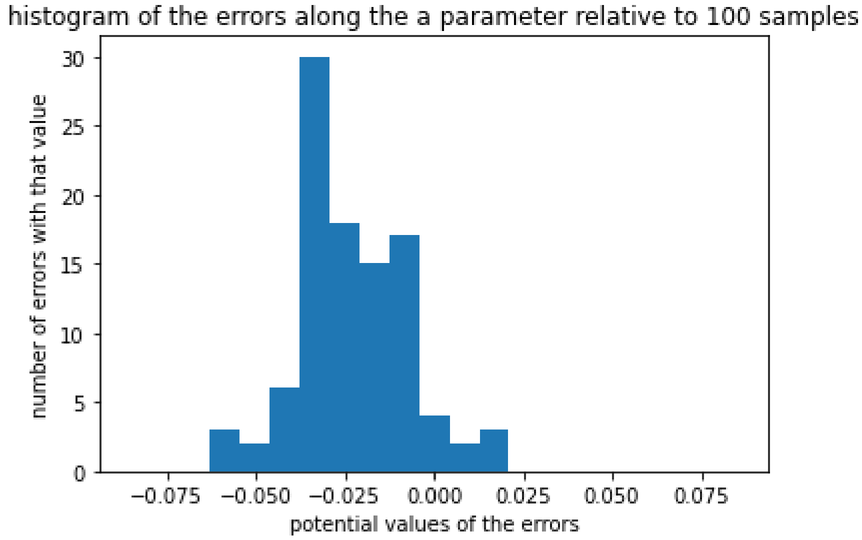
```
# cycle on the variance of the measurement noise
for j, sigma2 in enumerate(np.logspace(min_order_for_sigma2, max_order_for_si
    # MC cycles
    for m in range(N_MC_runs):
        # simulate the system
        y, u = simulate(a, b, K, lambda2, sigma2, y0, N, 0.3)
        # compute the solution
        a_hat, b_hat = PEM_solver(u, y)
        # assess the performance
        MSEs[j, m] = np.linalg.norm([a - a_hat, b - b_hat])**2
        # save the results
        theta_hats[j, m, 0] = a_hat
        theta_hats[j, m, 1] = b_hat
    print('sigma^2: {}, mean MSE: {}'.format(sigma2, np.mean(MSEs[j, :])))
```

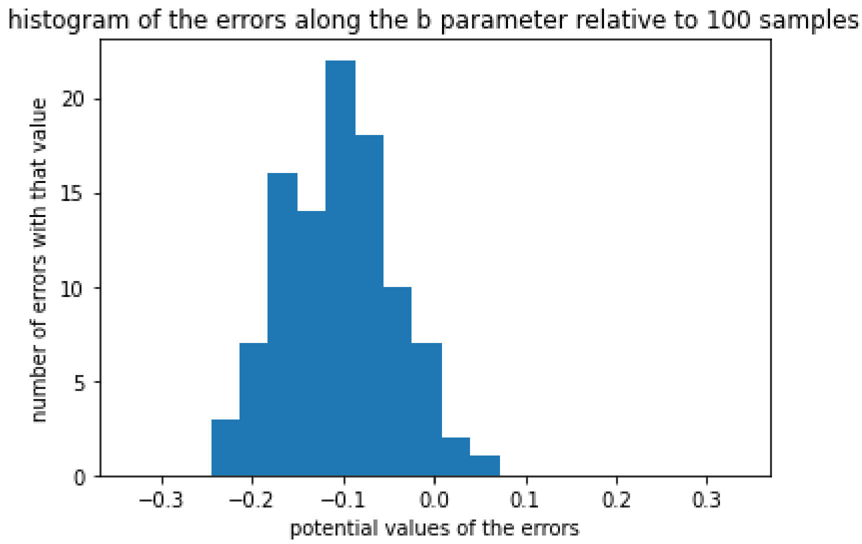
```
sigma^2: 1e-10, mean MSE: 0.015564968610855765
sigma^2: 1e-09, mean MSE: 0.014281251170752862
sigma^2: 1e-08, mean MSE: 0.01465343419593655
sigma^2: 1e-07, mean MSE: 0.015459301646150789
sigma^2: 1e-06, mean MSE: 0.015481844746281453
sigma^2: 1e-05, mean MSE: 0.012057170858130779
sigma^2: 0.0001, mean MSE: 0.016965243954967l9
sigma^2: 0.001, mean MSE: 0.016762080444981925
sigma^2: 0.01, mean MSE: 0.01515164840405222
sigma^2: 0.1, mean MSE: 0.013155274666114925
sigma^2: 1.0, mean MSE: 0.003349299554562071
sigma^2: 10.0, mean MSE: 0.0015481262567580531
sigma^2: 100.0, mean MSE: 0.0022939907841216804
sigma^2: 1000.0, mean MSE: 0.0020069557037689094
sigma^2: 10000.0, mean MSE: 0.0011657358052547208
sigma^2: 100000.0, mean MSE: 0.001472325270924003
sigma^2: 1000000.0, mean MSE: 0.0019061305105055484
sigma^2: 10000000.0, mean MSE: 0.001248338832911906
sigma^2: 100000000.0, mean MSE: 0.0014276196260910518
sigma^2: 1000000000.0, mean MSE: 0.0020755373746351572
sigma^2: 10000000000.0, mean MSE: 0.00342740898455812
```

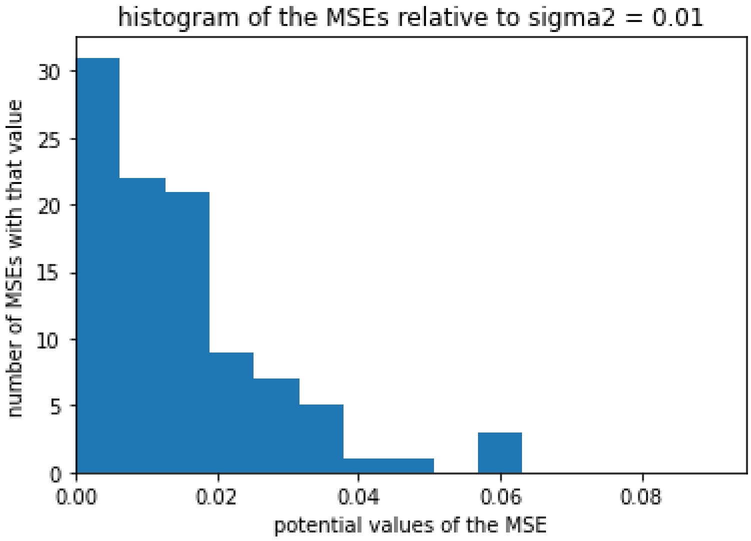
In [83]:

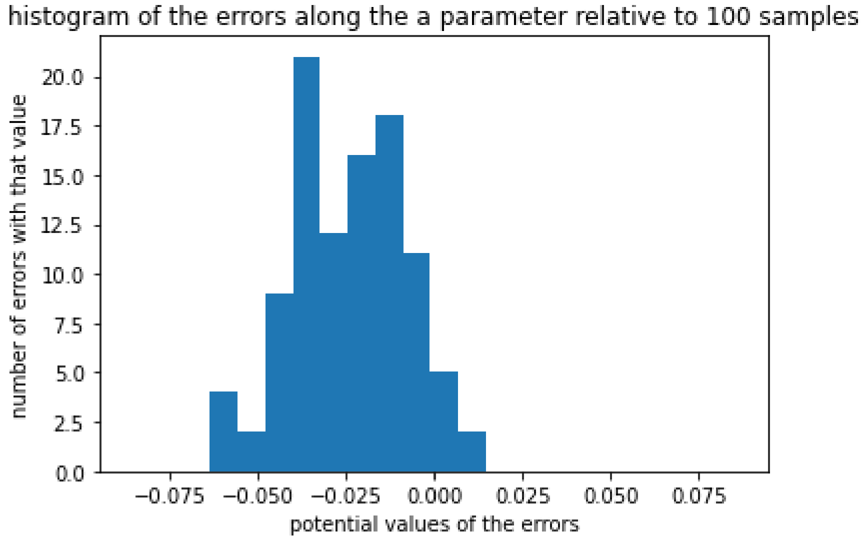
```
# cycle on the variance of the measurement noise
for j, sigma2 in enumerate(np.logspace(min_order_for_sigma2, max_order_for_si
    # plot the histogram of the MSEs relative to this number of samples
    plt.figure(j)
    plt.hist(MSEs[j,:])
    plt.xlim(0, 1.5 * np.max(MSEs[j,:]))
    plt.title('histogram of the MSEs relative to sigma2 = {}'.format(sigma2))
    plt.xlabel('potential values of the MSE')
    plt.ylabel('number of MSEs with that value')

    # plot the histogram of the errors along the a parameter
    plt.figure(j + 100)
    x_lim = np.max(np.abs(theta_hats[j,:,0] - a))
    plt.hist(theta_hats[j,:,0] - a)
    plt.xlim(-1.5 * x_lim, 1.5 * x_lim)
    plt.title('histogram of the errors along the a parameter relative to {} s
    plt.xlabel('potential values of the errors')
    plt.ylabel('number of errors with that value')

    # plot the histogram of the errors along the b parameter
    plt.figure(j + 200)
    x_lim = np.max(np.abs(theta_hats[j,:,1] - b))
    plt.hist(theta_hats[j,:,1] - b)
    plt.xlim(-1.5 * x_lim, 1.5 * x_lim)
    plt.title('histogram of the errors along the b parameter relative to {} s
    plt.xlabel('potential values of the errors')
    plt.ylabel('number of errors with that value')
```

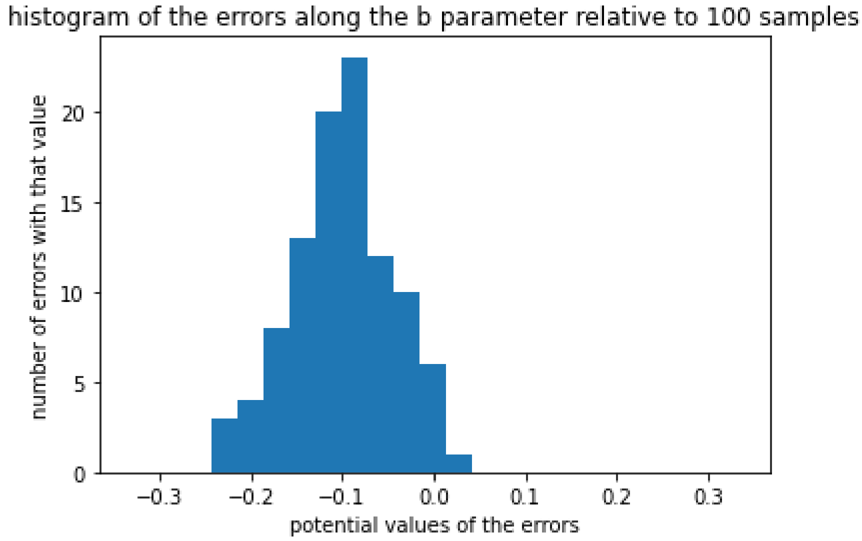

histogram of the MSEs relative to sigma2 = 0.001
number of MSEs with that value
potential values of the MSE
0
5
10
15
20
25
30
0.00
0.02
0.04
0.06
0.08

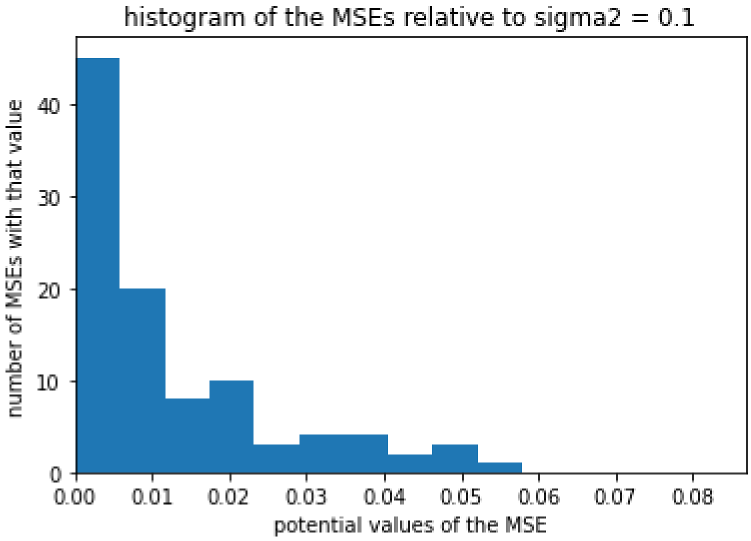

histogram of the errors along the a parameter relative to 100 samples
number of errors with that value
potential values of the errors
0
5
10
15
20
25
30
−0.075
−0.050
−0.025
0.000
0.025
0.050
0.075

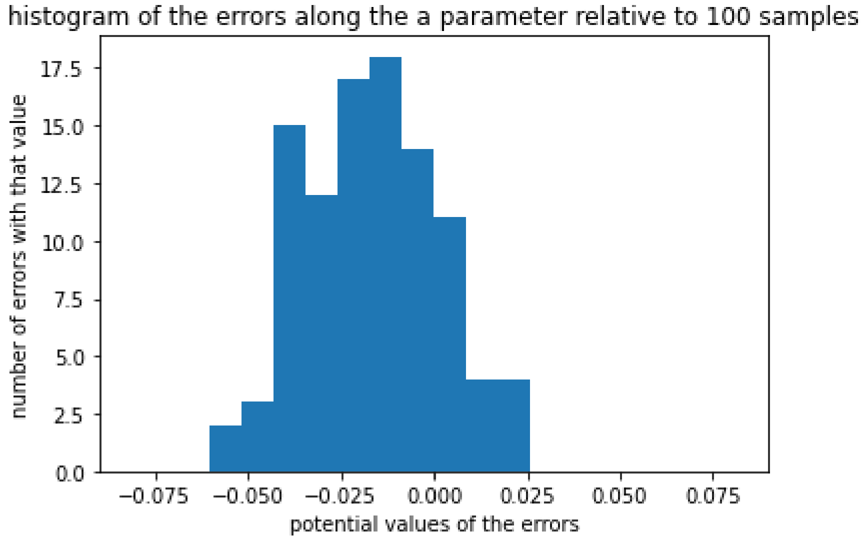

histogram of the errors along the b parameter relative to 100 samples
number of errors with that value
potential values of the errors
0
5
10
15
20
−0.3
−0.2
−0.1
0.0
0.1
0.2
0.3

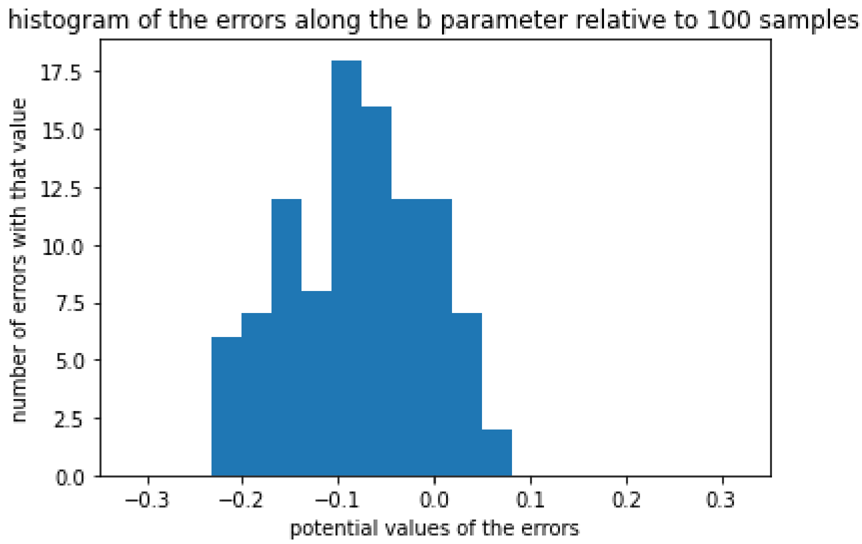

histogram of the MSEs relative to sigma2 = 0.01
number of MSEs with that value
potential values of the MSE

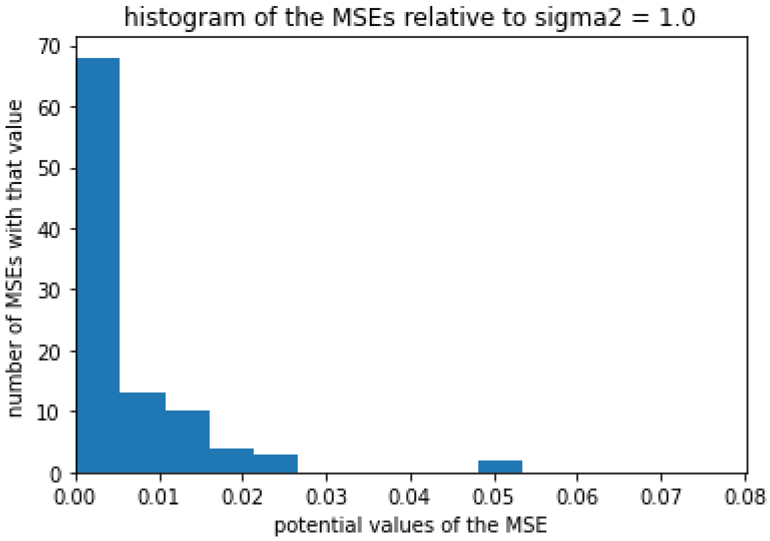

histogram of the errors along the a parameter relative to 100 samples
number of errors with that value
potential values of the errors

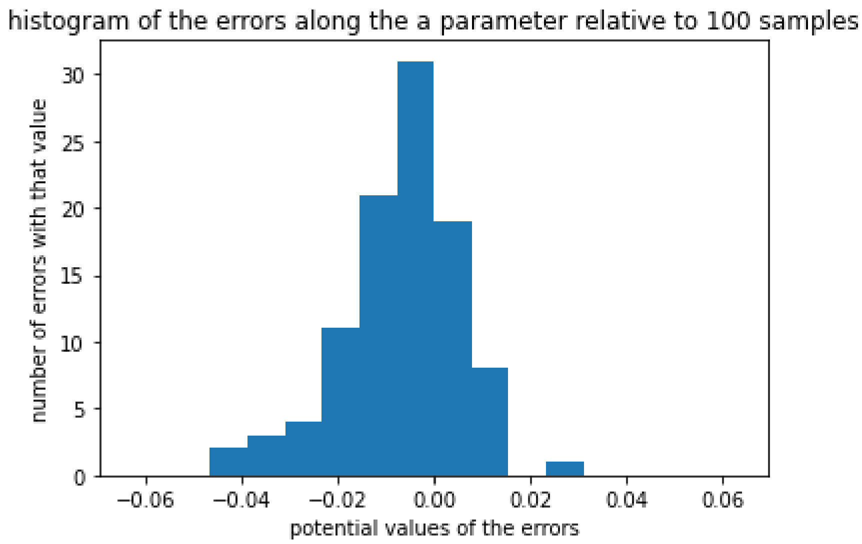

histogram of the errors along the b parameter relative to 100 samples
number of errors with that value
potential values of the errors

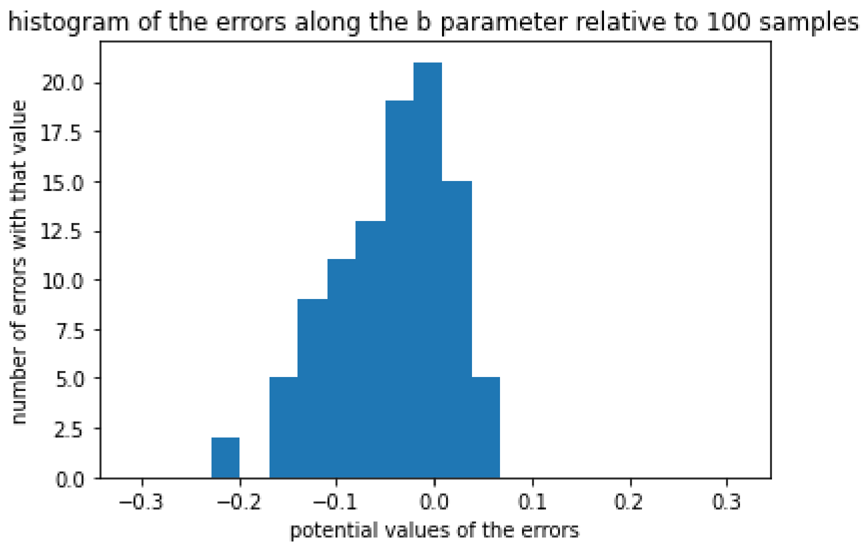

histogram of the MSEs relative to sigma2 = 0.1
number of MSEs with that value
0
10
20
30
40
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08
potential values of the MSE

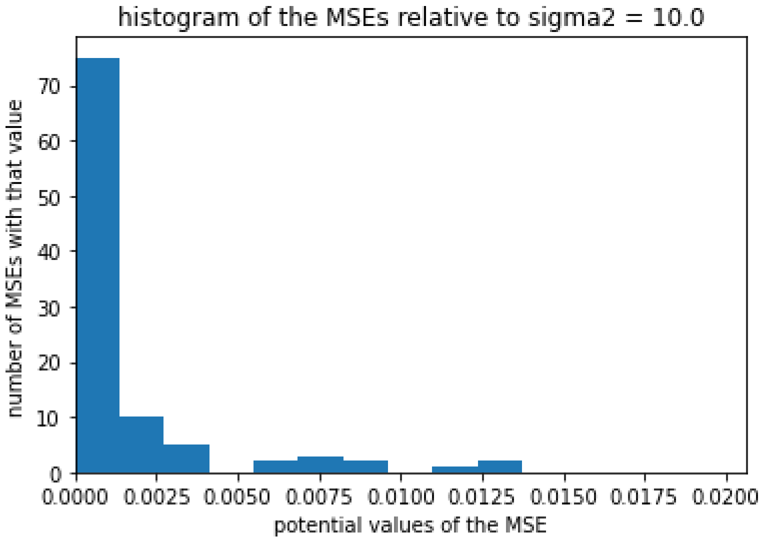

histogram of the errors along the a parameter relative to 100 samples
number of errors with that value
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
−0.075
−0.050
−0.025
0.000
0.025
0.050
0.075
potential values of the errors

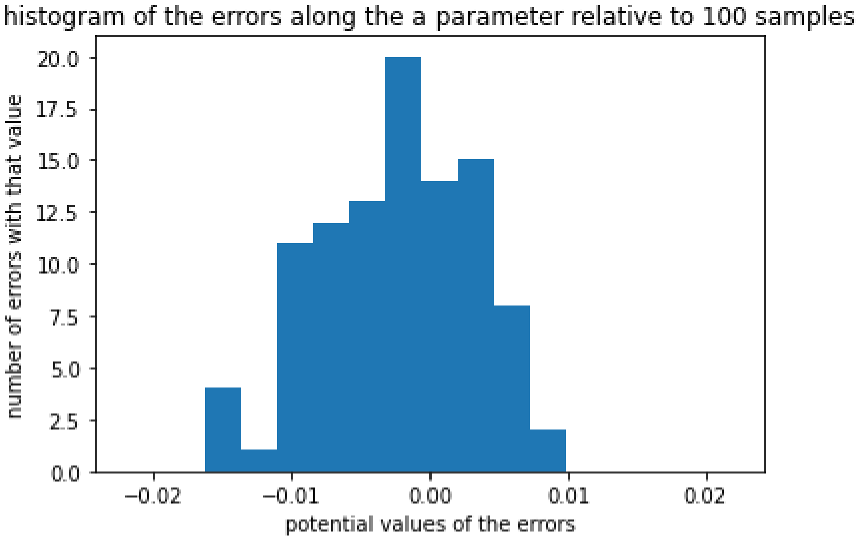

histogram of the errors along the b parameter relative to 100 samples
number of errors with that value
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
−0.3
−0.2
−0.1
0.0
0.1
0.2
0.3
potential values of the errors

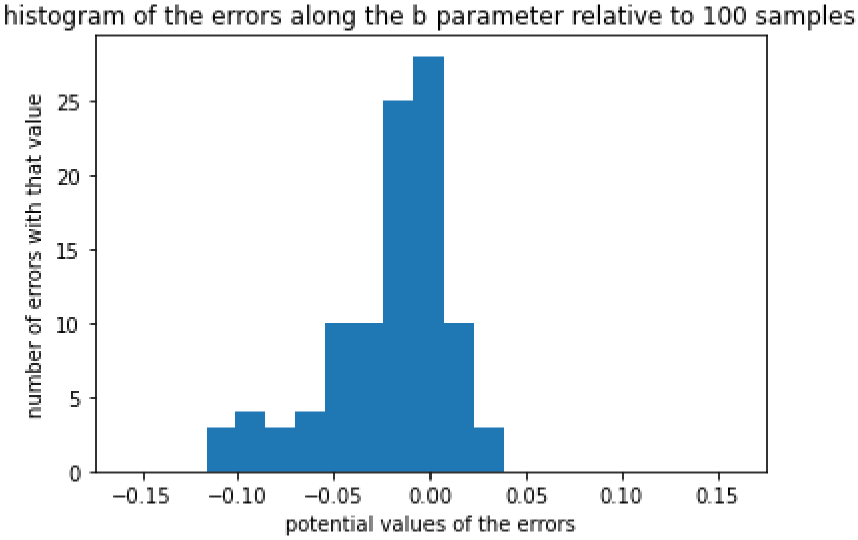

histogram of the MSEs relative to sigma2 = 1.0
number of MSEs with that value
potential values of the MSE
0
10
20
30
40
50
60
70
0.00
0.01
0.02
0.03
0.04
0.05
0.06
0.07
0.08


histogram of the errors along the a parameter relative to 100 samples
number of errors with that value
potential values of the errors
0
5
10
15
20
25
30
−0.06
−0.04
−0.02
0.00
0.02
0.04
0.06


histogram of the errors along the b parameter relative to 100 samples
number of errors with that value
potential values of the errors
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
20.0
−0.3
−0.2
−0.1
0.0
0.1
0.2
0.3







Task 10.4

Comment what you think is a remarkable fact relative to the simulations above.

The variance of the estimates does not approach infinity when $\sigma^2 \rightarrow 0$, but might have the potential to converge to infinity at sufficiently small σ^2. Notice the minimum and maximum values MSE recorded:

```
sigma^2: 10000000000.0, mean MSE: 0.00342740898455812
sigma^2: 1e-10, mean MSE: 0.015564968610855765
```

These MSEs are not close to 0 or ∞, but shows that large values for σ^2 gives lower MSE than small values for σ^2

In [ ]:

```

```