# 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 $\lambda^2$
- 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 $\sigma^2$

In [1]:

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

In [2]:

```
# main 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)

    # saving the initial condition
    y[0] = y0

    # 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) )

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

    return [y, u]
```

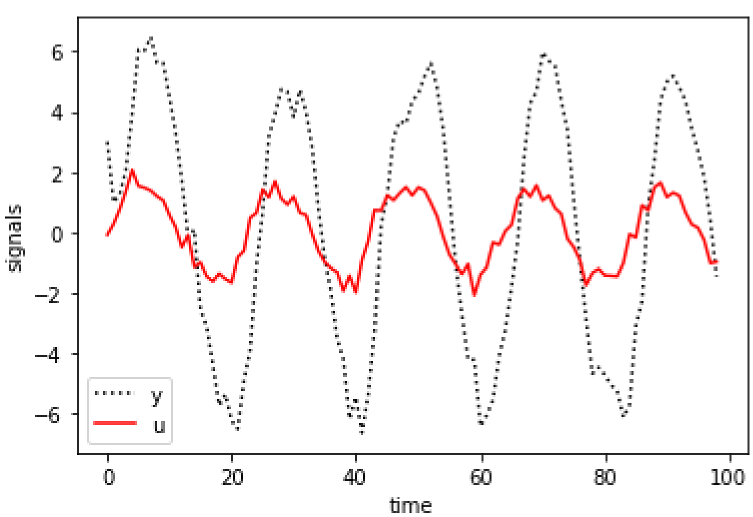
In [3]:

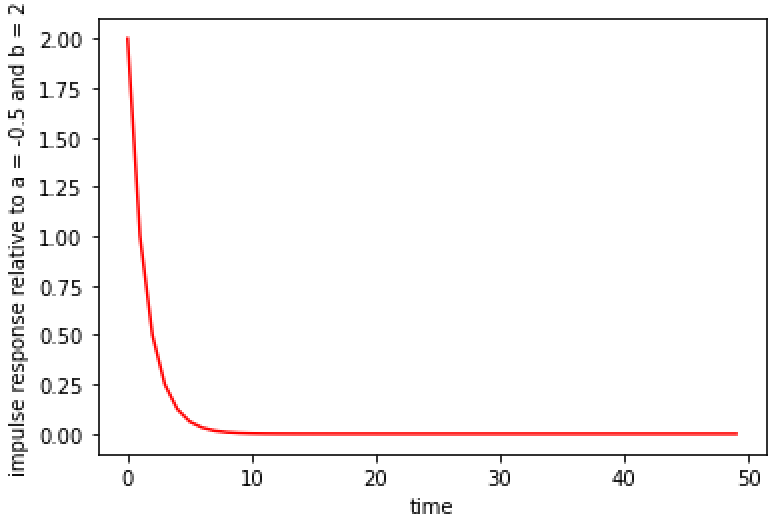
```
# define also a function for doing poles allocation, considering
# that eventually if the reference is absent then the ODE is
#
# y_k + ( a + b K ) y_{k-1} = e_k
#
def compute_gain( a, b, desired_pole_location ):
    K = -(desired_pole_location + a)/b
    return K
```

In [4]:
```python
# plotting of the impulse response
def plot_impulse_response( a, b, figure_number = 1000 ):

    # ancillary quantities
    k = range(0,50)
    y = b * np.power( -a, k )

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

In [5]:
```python
# 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 [6]:
```python
# 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 )
```






# 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 [7]:

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

In [8]:

```
# define the function solving the PEM problem asked in the assignment
def PEM_solver( u, y ):
    N = len(y)
    phi = np.column_stack((-y[:N-1],u[:N-1]))
    # compute the PEM estimate by directly solving the normal equations
    theta_hat = np.linalg.solve(phi.T @ phi,phi.T @ y[1:N]) # be careful with the index

    # explicit the results
    a_hat = theta_hat[0]
    b_hat = theta_hat[1]

    return [a_hat, b_hat]
```

In [9]:

```
# 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: 0.0002026462171137257
a, b = -0.5, 2   --    ahat, bhat = -0.5024752571696591, 2.0140185348399102
```

# 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 [16]:

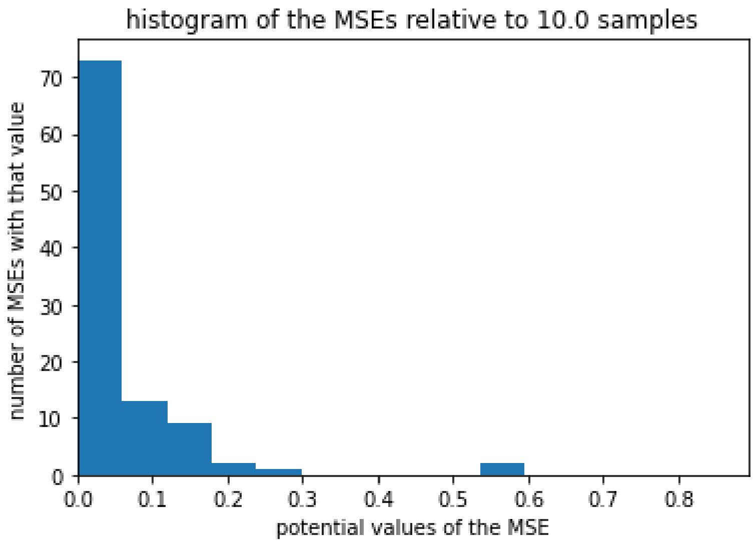
```python
# the best way to show this is to do a Monte Carlo approach:
# - for each N, compute the distribution of the estimates
#   around the true parameters
# - increase N and show that this distribution tends to
#   converge to the true parameters

# 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_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_orders )

    # debug
    print('starting computing order {} of {}'.format(j+1, num_of_N_orders))

    # 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
```

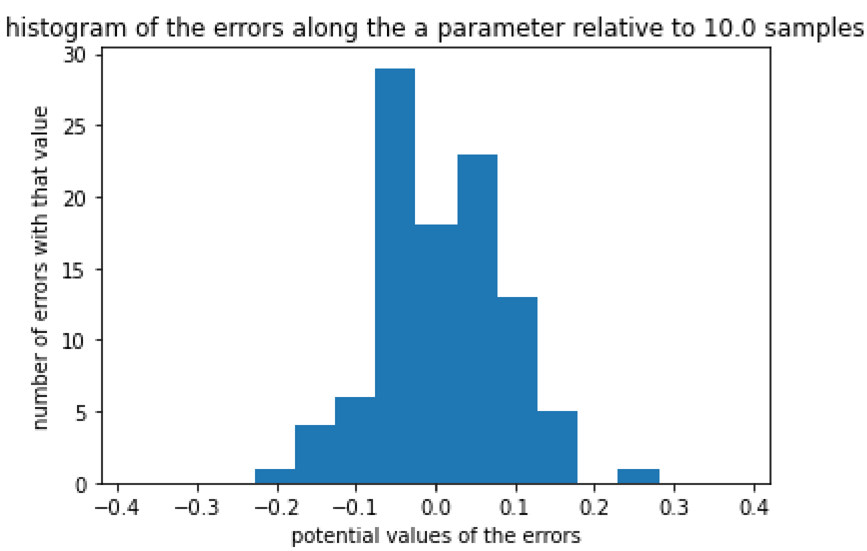
```
starting computing order 1 of 4
starting computing order 2 of 4
starting computing order 3 of 4
starting computing order 4 of 4
```

In [18]:

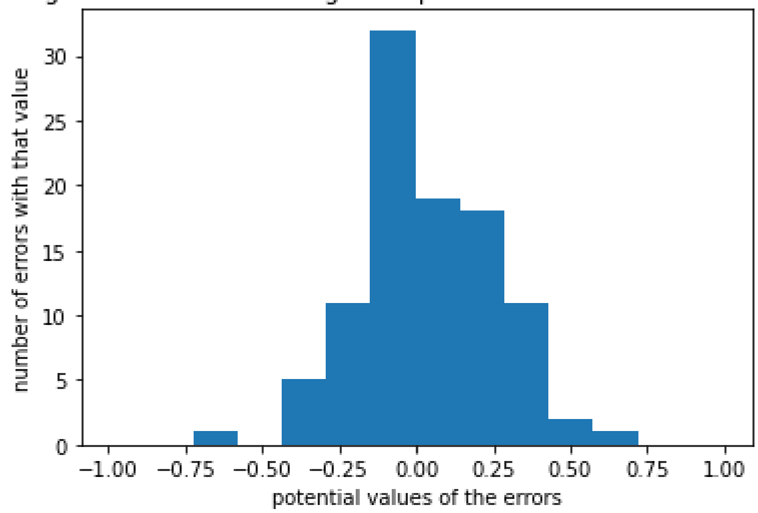
```python
# cycle on the number of samples
for j, N in enumerate( np.logspace( min_order_for_N, max_order_for_N, num_of_N_orders )

    # 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 {} samples'.fo
    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 {} samples'.fo
    plt.xlabel('potential values of the errors')
    plt.ylabel('number of errors with that value')
```

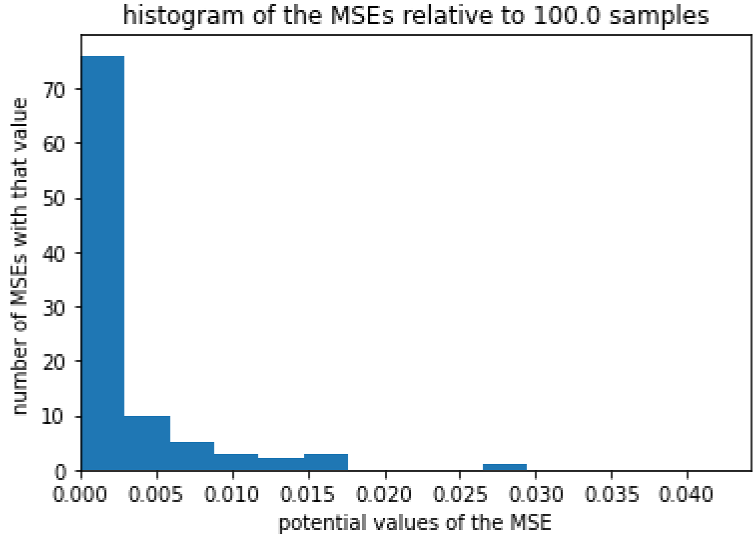


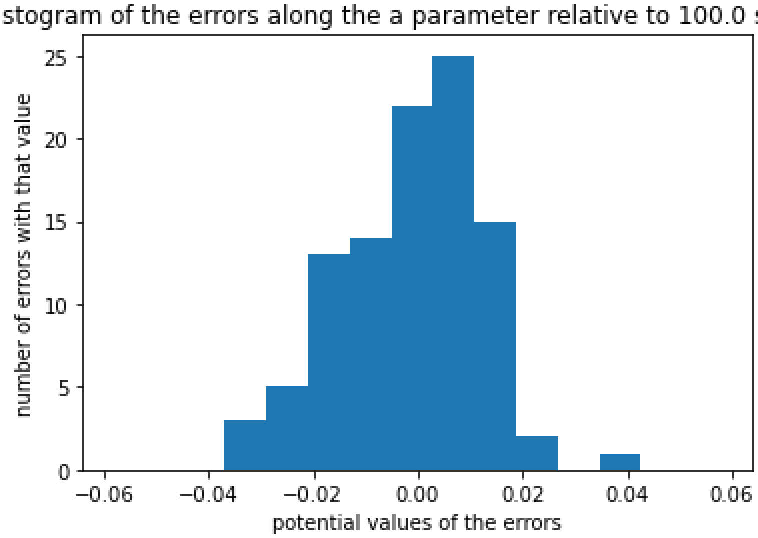
histogram of the errors along the b parameter relative to 10.0 samples

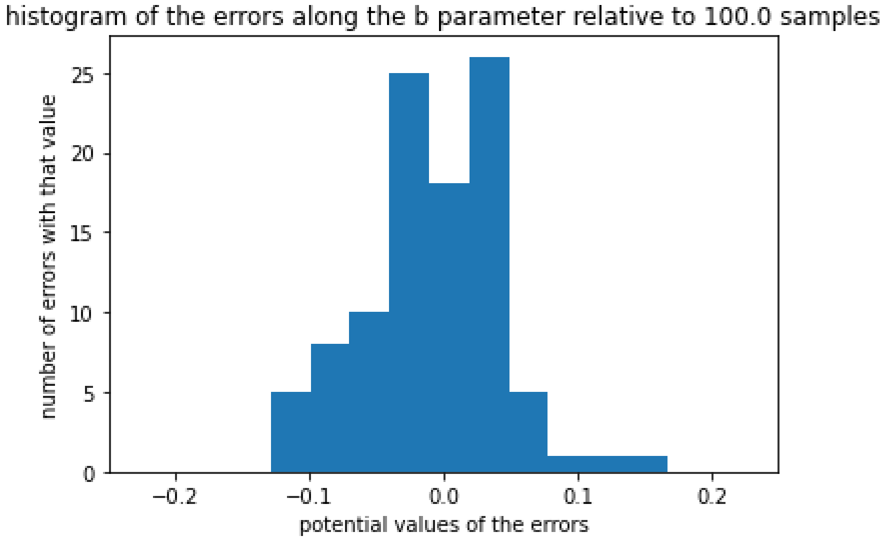


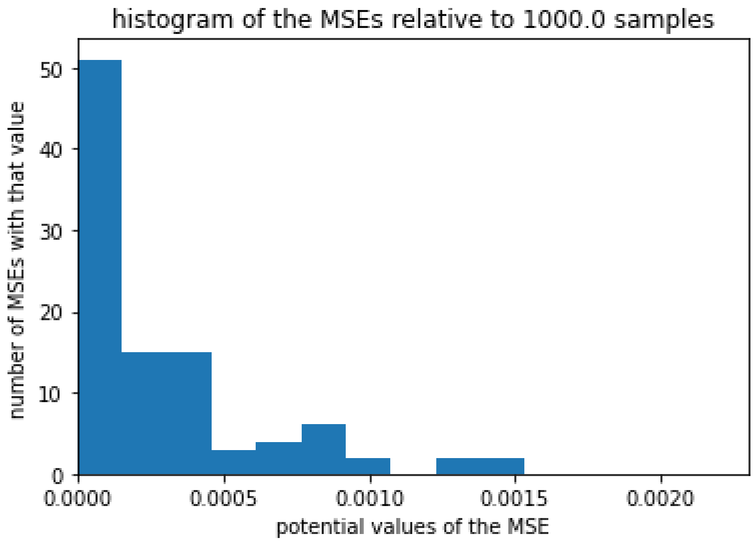

histogram of the MSEs relative to 100.0 samples

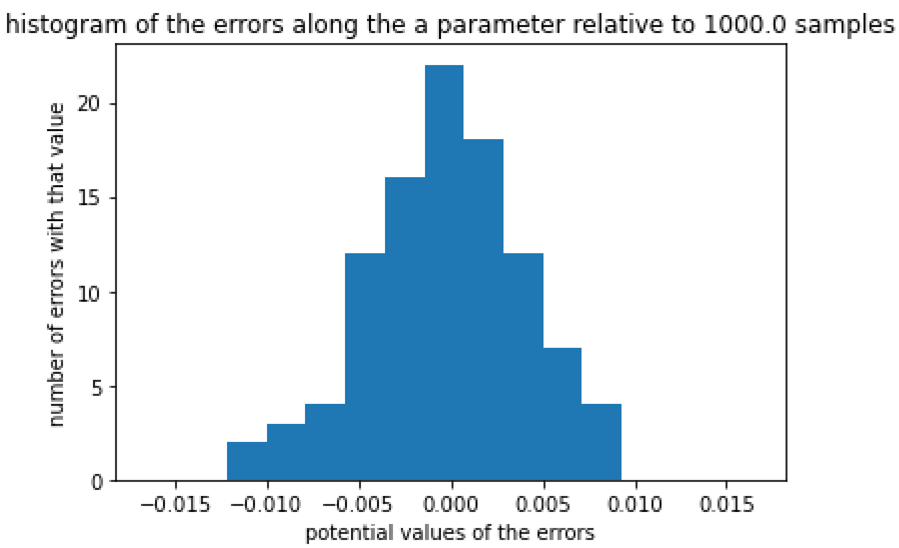


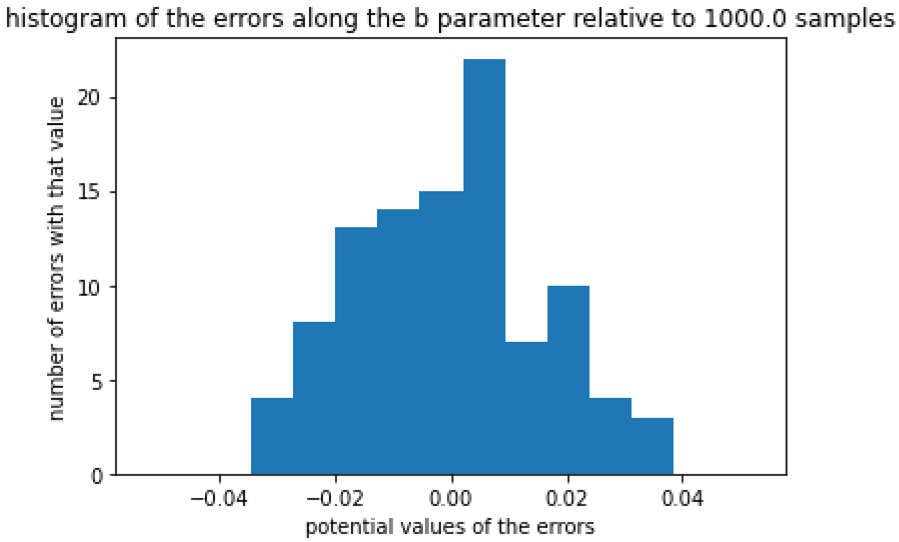

histogram of the errors along the a parameter relative to 100.0 samples

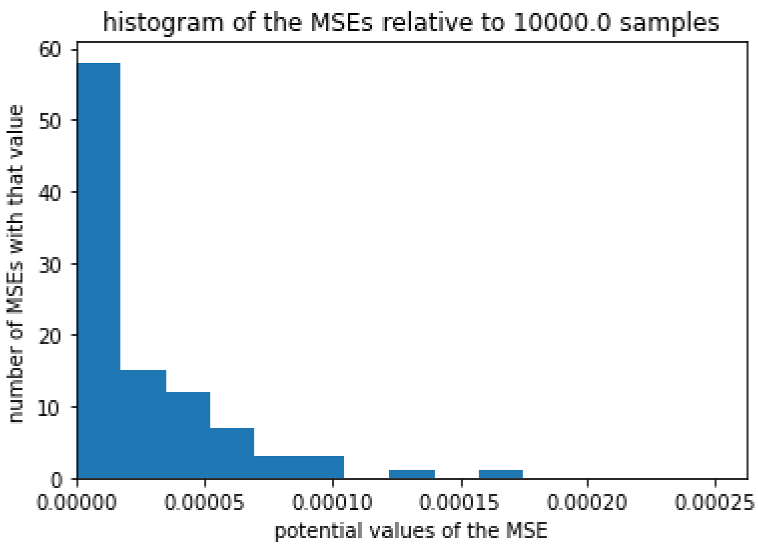


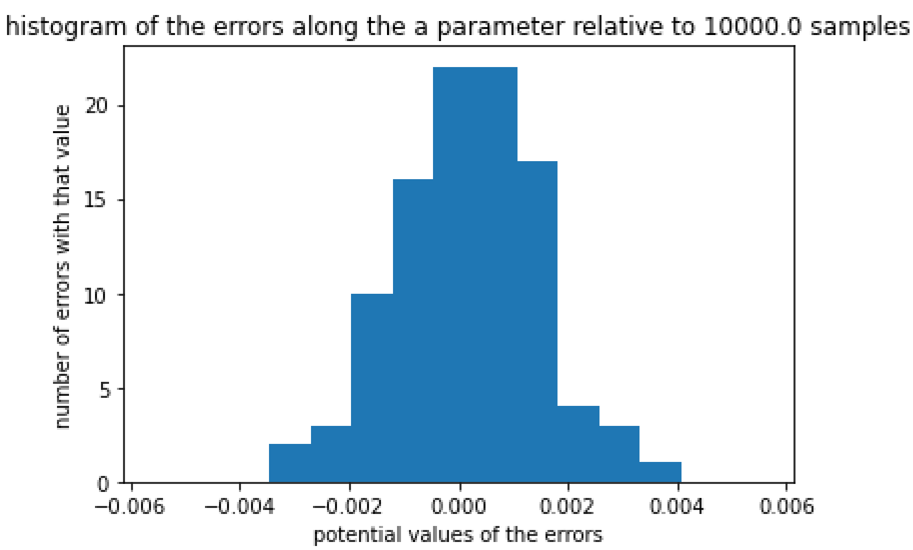

histogram of the errors along the b parameter relative to 100.0 samples
number of errors with that value
potential values of the errors
0
5
10
15
20
25
−0.2
−0.1
0.0
0.1
0.2

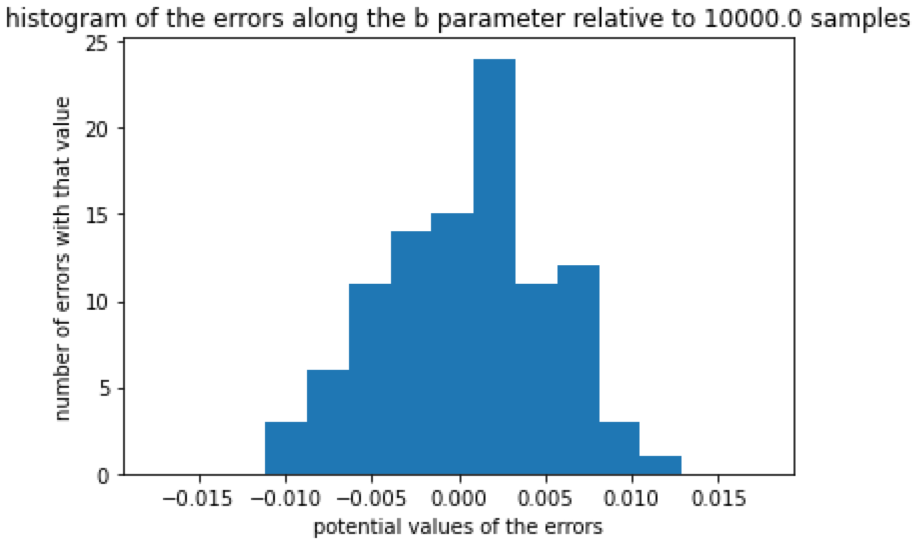

histogram of the MSEs relative to 1000.0 samples
number of MSEs with that value
potential values of the MSE
0
10
20
30
40
50
0.0000
0.0005
0.0010
0.0015
0.0020


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
5
10
15
20
−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
5
10
15
20
−0.04
−0.02
0.00
0.02
0.04


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
60
0.00000
0.00005
0.00010
0.00015
0.00020
0.00025


histogram of the errors along the a parameter relative to 10000.0 samples
number of errors with that value
potential values of the errors
0
5
10
15
20
−0.006
−0.004
−0.002
0.000
0.002
0.004
0.006




## Task 10.3

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

In [19]:

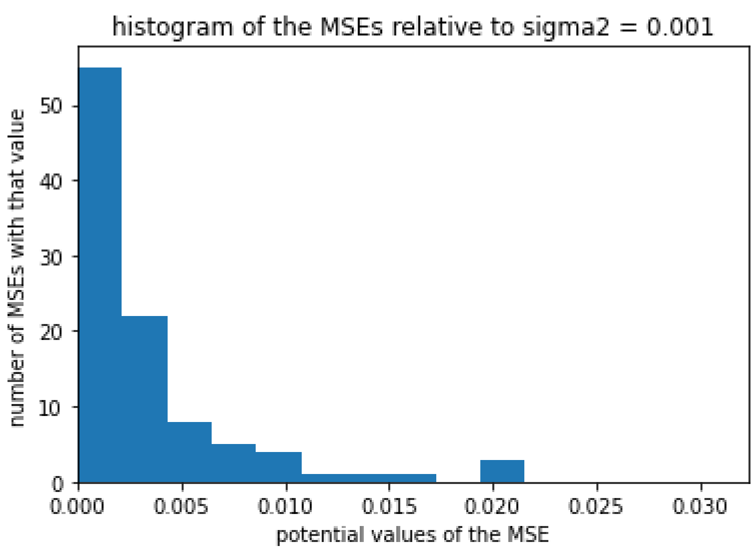
```
# again the best way to show this is to do a Monte Carlo approach:
# - for each sigma2, compute the distribution of the estimates
#   around the true parameters
# - diminish sigma2 and show that this distribution tends to
#   diverge

# defining the MC simulation
N                      = 100
N_MC_runs              = 100
min_order_for_sigma2 = -3
max_order_for_sigma2 = 1
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_sigma2, nu

    # debug
    print('starting computing order {} of {}'.format(j+1, num_of_sigma2_orders))

    # 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)
```

```python
        # 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
```

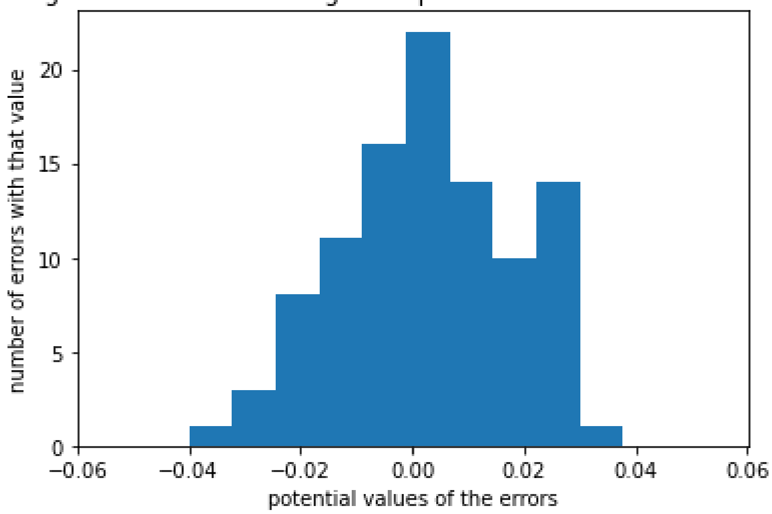
```
starting computing order 1 of 5
starting computing order 2 of 5
starting computing order 3 of 5
starting computing order 4 of 5
starting computing order 5 of 5
```

In [21]:

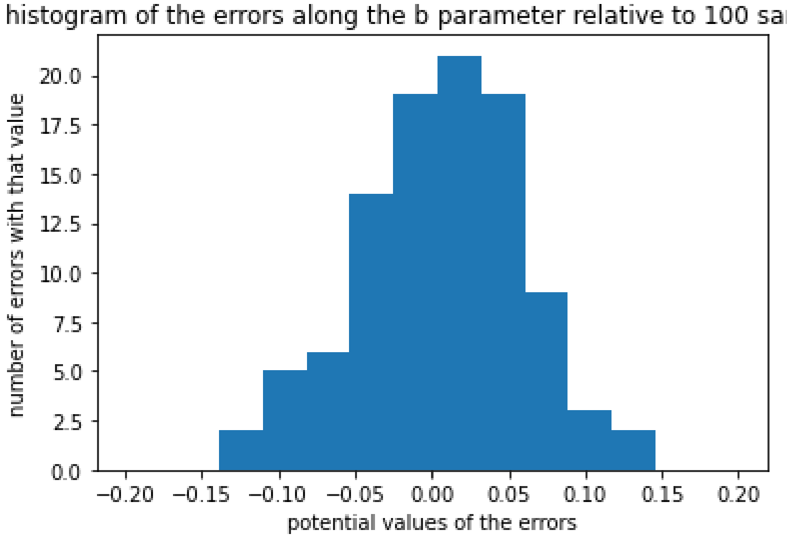
```python
# cycle on the variance of the measurement noise
for j, sigma2 in enumerate( np.logspace( min_order_for_sigma2, max_order_for_sigma2, nu

    # 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 {} samples'.fo
    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 {} samples'.fo
    plt.xlabel('potential values of the errors')
    plt.ylabel('number of errors with that value')
```



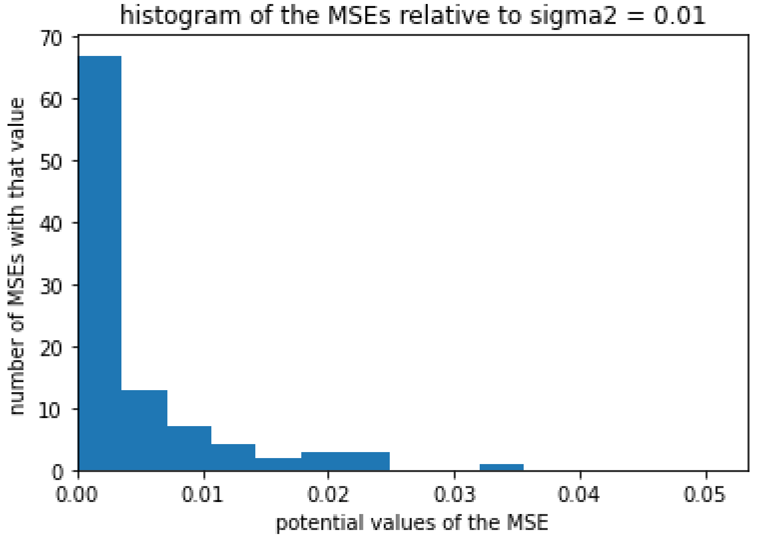
histogram of the errors along the a parameter relative to 100 samples

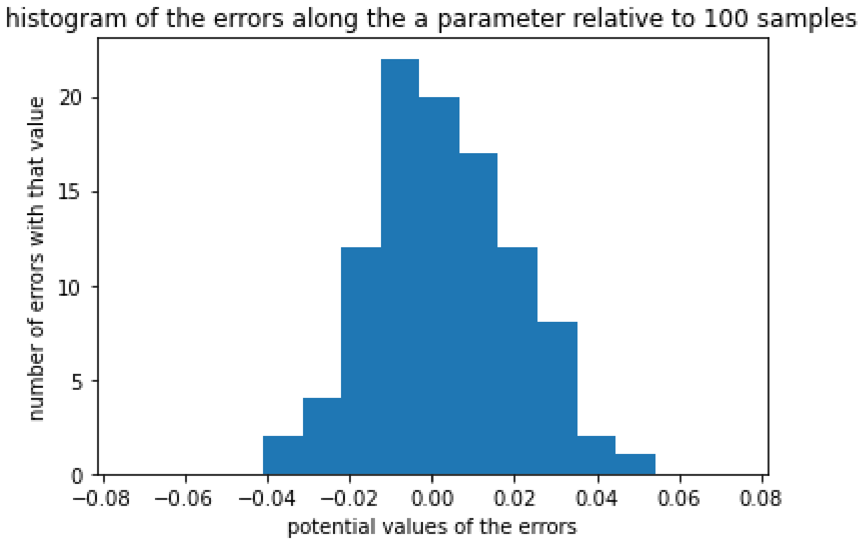


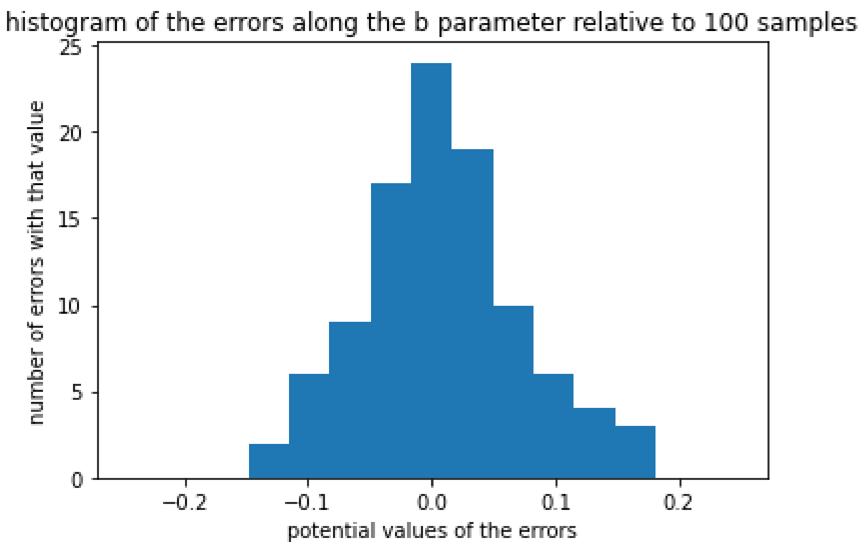

histogram of the errors along the b parameter relative to 100 samples

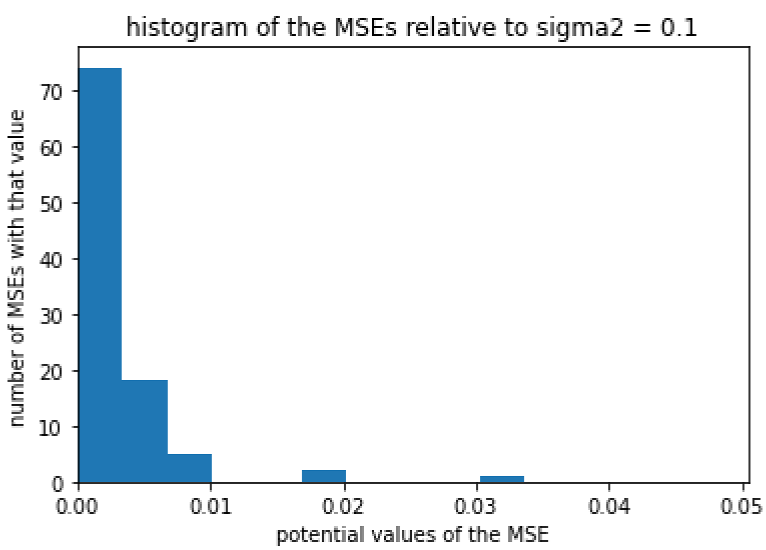


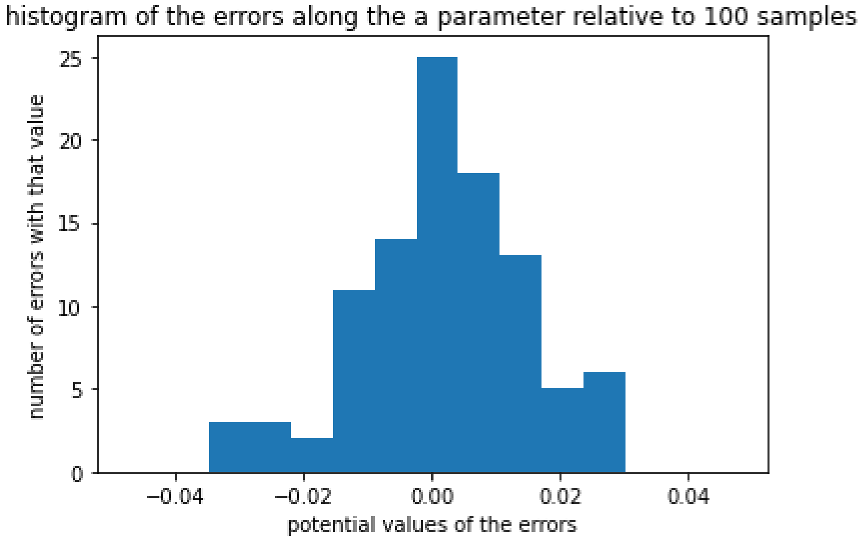

histogram of the MSEs relative to sigma2 = 0.01

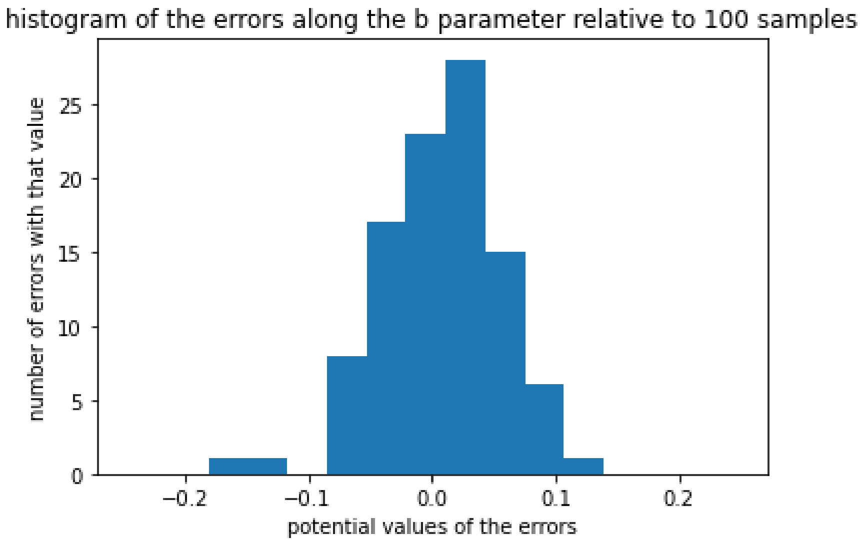


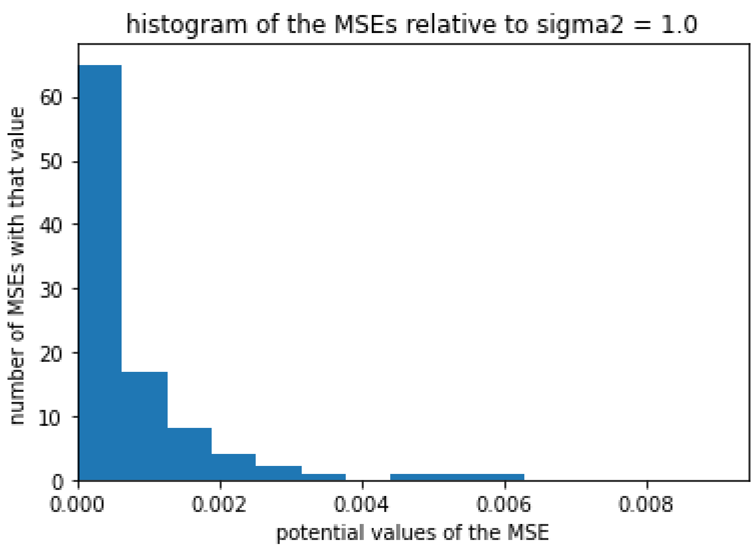

histogram of the errors along the a parameter relative to 100 samples
number of errors with that value
potential values of the errors
0
5
10
15
20
−0.08
−0.06
−0.04
−0.02
0.00
0.02
0.04
0.06
0.08

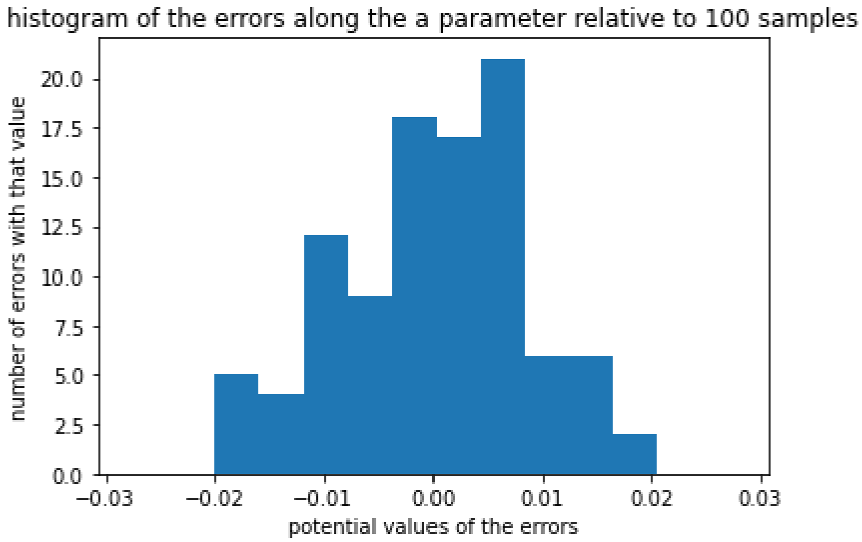

histogram of the errors along the b parameter relative to 100 samples
number of errors with that value
potential values of the errors
0
5
10
15
20
25
−0.2
−0.1
0.0
0.1
0.2

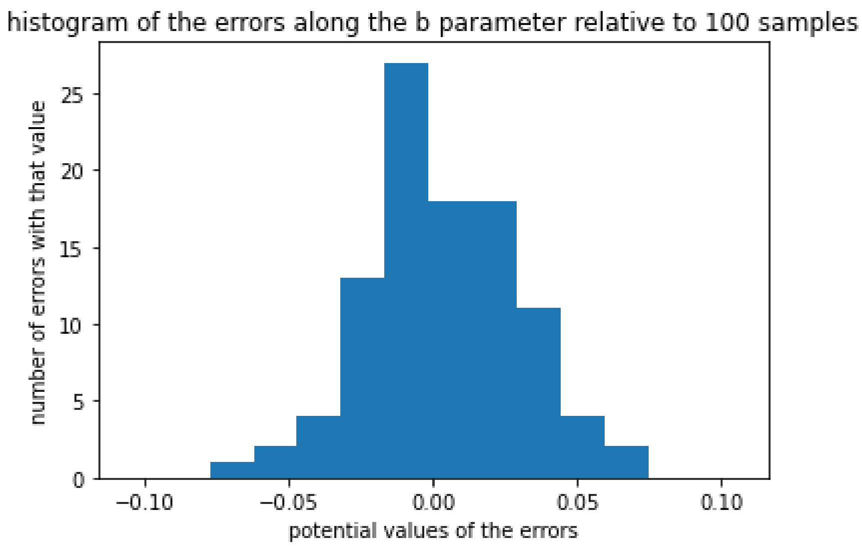

histogram of the MSEs relative to sigma2 = 0.1
number of MSEs with that value
potential values of the MSE
0
10
20
30
40
50
60
70
0.00
0.01
0.02
0.03
0.04
0.05

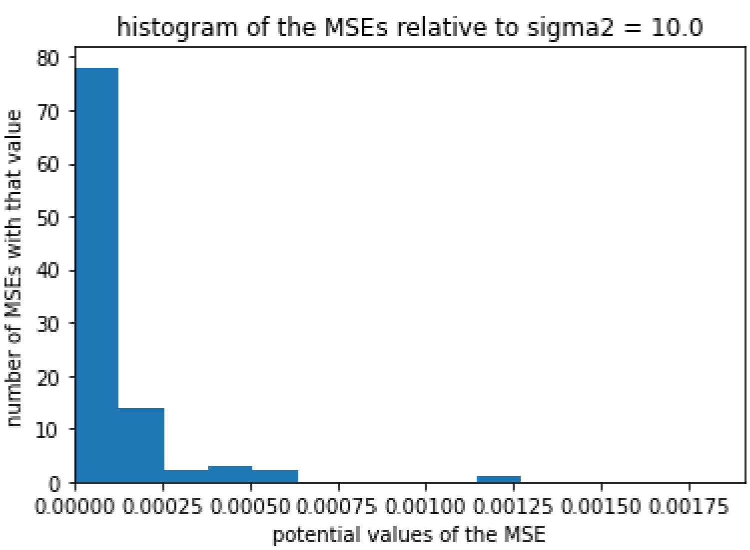

histogram of the errors along the a parameter relative to 100 samples
number of errors with that value
potential values of the errors
0
5
10
15
20
25
−0.04
−0.02
0.00
0.02
0.04

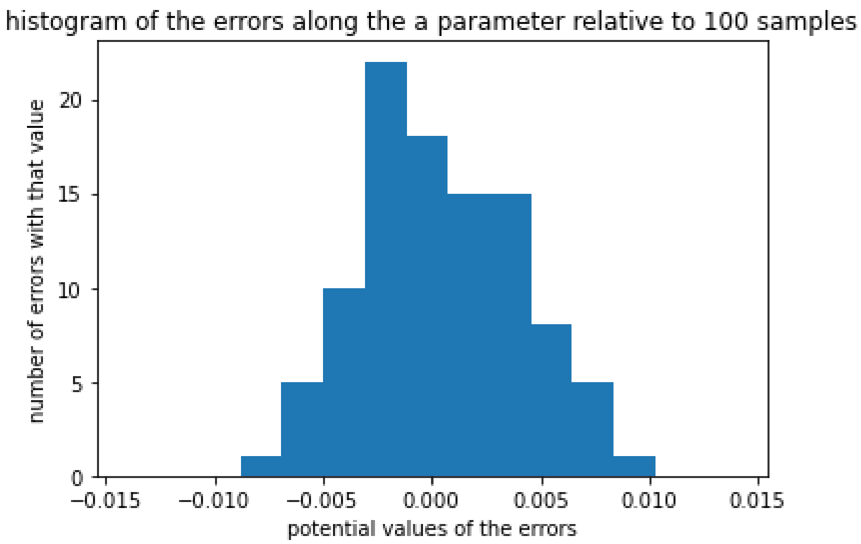

histogram of the errors along the b parameter relative to 100 samples
number of errors with that value
potential values of the errors
0
5
10
15
20
25
−0.2
−0.1
0.0
0.1
0.2

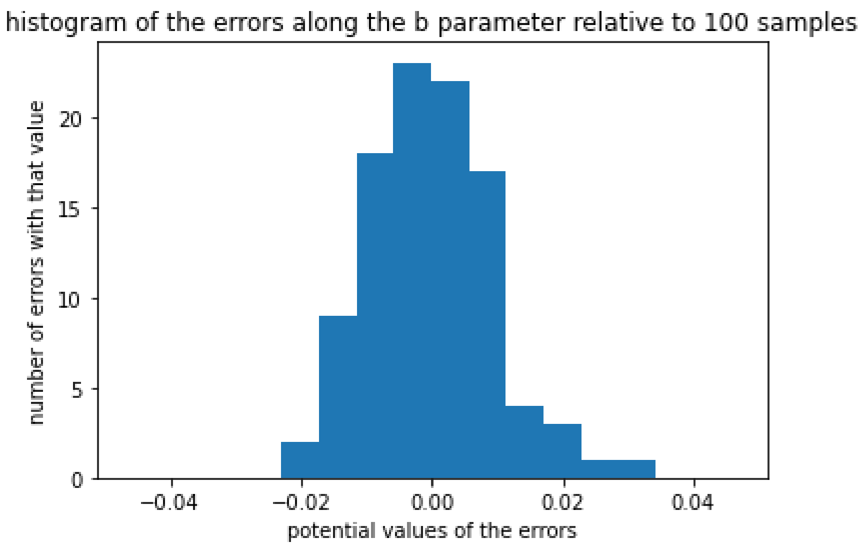

histogram of the MSEs relative to sigma2 = 1.0
number of MSEs with that value
potential values of the MSE
0
10
20
30
40
50
60
0.000
0.002
0.004
0.006
0.008


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
20.0
−0.03
−0.02
−0.01
0.00
0.01
0.02
0.03
potential values of the errors


histogram of the errors along the b parameter relative to 100 samples
number of errors with that value
0
5
10
15
20
25
−0.10
−0.05
0.00
0.05
0.10
potential values of the errors


histogram of the MSEs relative to sigma2 = 10.0
number of MSEs with that value
0
10
20
30
40
50
60
70
80
0.00000
0.00025
0.00050
0.00075
0.00100
0.00125
0.00150
0.00175
potential values of the MSE




# Task 10.4

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

First part of task 10.3: Since the variance of the error signal is constant, the average error will become smaller as the number of samples increase.

It is remarkable that the more deterministic the process becomes, the worse your estimations become. This makes sence, since this means that the reference signal will go towards 0, creating little response and little variation in the signals u and y. Thus, we the system gives us to little information to be able to estimate the parameters exactly. In other words, our signal is not persistantly exciting.

Note to self: $\lambda^2$ is the variance of the error and $\sigma^2$ is the variance of the reference signal.