Национальный исследовательский университет «МЭИ»

Институт Радиотехники и электроники им. В.А. Котельникова

Кафедра Радиотехнических систем

Домашнее задание

по курсу «Методы оптимального приёма сигналов в аппаратуре потребителей СРНС»

по теме «Оптимальная линейная фильтрация»

Выполнила: Малафеева Д.Д.

Группа: ЭР-12м-19

Москва

2020

Часть 1

Анализ и моделирование системы ЧАП

Начальные данные:

- Ширина спектра флуктуаций ускорения $\alpha = 1c^{-1}$;
- Флуктуационная характеристика частотного дискриминатора $\tilde{N}_0(q_{c/n0}) = \dfrac{2}{q_{c/n0}T^2}\left(1 + \dfrac{1}{2q_{c/n0}T}\right), T = 10мс$;
- Отношение мощности сигнала к спектральной плотности шума на входе приемника $q_{c/n0} = 10^{0.1(14К\ 50\ ДбГц)}\left[Гц\right]$;
- Несущая частота: $\omega_0 = 2\pi \cdot (1602Мгц)$;

Задание

1. Найти аналитически и построить на графиках зависимости среднеквадратической ошибки фильтрации частоты и оптимальной полосы ЧАП от отношения с/ш при $\sigma_a = 10м/c^2$:

Решение

1) Найдем СПМ формирующего шума:

$$S_\xi = 2{\sigma_a}^2\alpha\left(\frac{\omega_0}{c}\right)^2 \tag{1.1}$$

2) Далее можно рассчитать в заданном диапазоне $q_{c/n0}$ флуктуационную характеристику частотного дискриминатора:

$$\tilde{N}_0(q_{c/n0}) = \frac{2}{q_{c/n0}T^2}\left(1 + \frac{1}{2q_{c/n0}T}\right), \tag{1.2}$$

3) Среднеквадратическую ошибку фильтрации частоты можно определить как корень из дисперсии D_{11}:

$$\sigma_\Omega = \sqrt{D_{11}} = \sqrt{\frac{\alpha\tilde{N}_0}{2}\left(\sqrt{1 + 2\sqrt{S_\xi \big/ \left(\alpha^2\tilde{N}_0\right)}} - 1\right)} \tag{1.3}$$

Проведем соответствующее моделирование на языке Python:

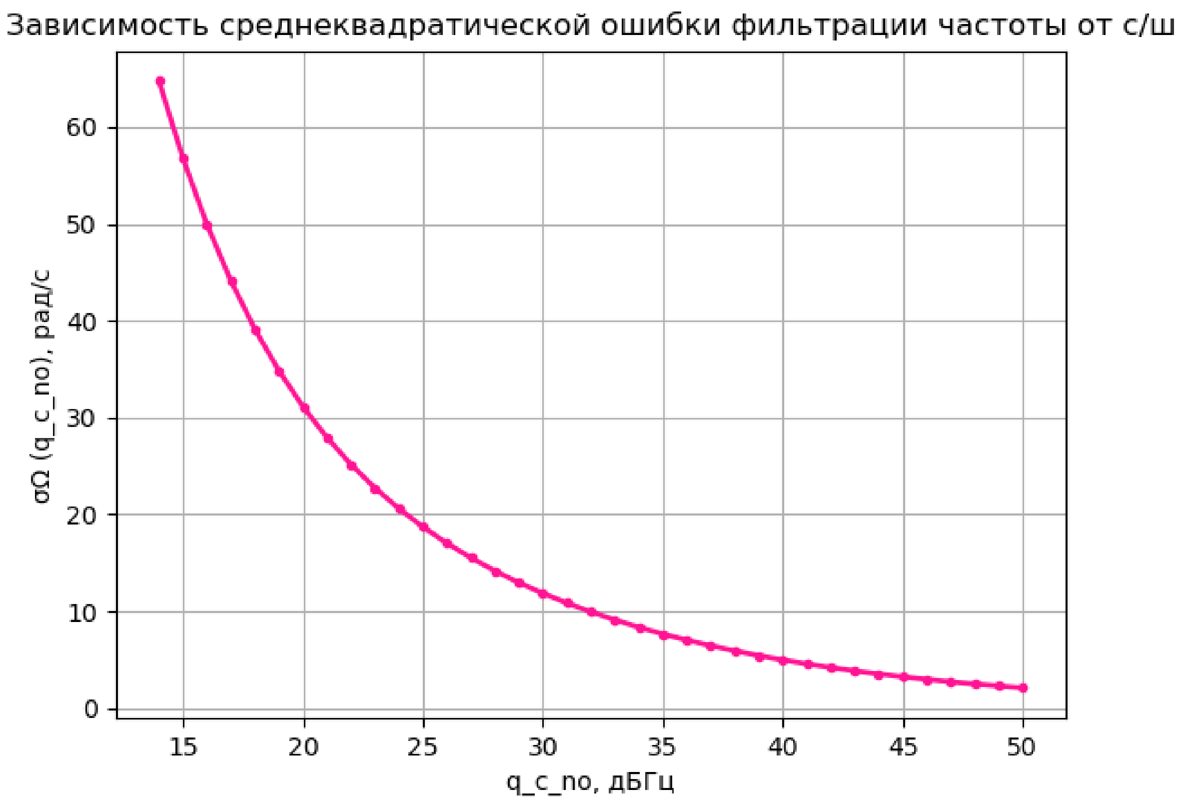

Рис.1.1 Зависимость среднеквадратической ошибки фильтрации частоты от с/ш

Как и предполагалось, с увеличением отношения с/ш в пределах до 50 дБГц среднеквадратическая ошибка фильтрации частоты уменьшается.

4) Далее рассчитаем коэффициент передачи фильтра для расчета полосы ЧАП:

$$K1 = \alpha\left(\sqrt{1+2\sqrt{S_\xi/\left(\alpha^2 N_0\right)}}-1\right), K2 = K1^2/2,$$
$$K_\phi(p) = \frac{1}{p}\left(K_1+\frac{K_2}{p+\alpha}\right), \tag{1.4}$$

5) Полоса ЧАП:

$$\Delta F_{ЧАП} = \frac{1}{2\pi\left|K_{y\Omega}(0)\right|^2}\int_0^\infty \left|K_{y\Omega}(j\omega)\right|^2 d\omega,$$
$$K_{y\Omega}(p) = \frac{K_\phi(p)}{1+K_\phi(p)}, \tag{1.5}$$

С учетом того, что $\left|K_{y\Omega}(0)\right| = 1$ и $p = j\omega$ итоговая формула для полосы:

$$\Delta F_{\text{ЧАП}} = \frac{1}{2\pi}\int_0^\infty \left|K_{y\Omega}(j\omega)\right|^2 d\omega.$$

Результат моделирования:

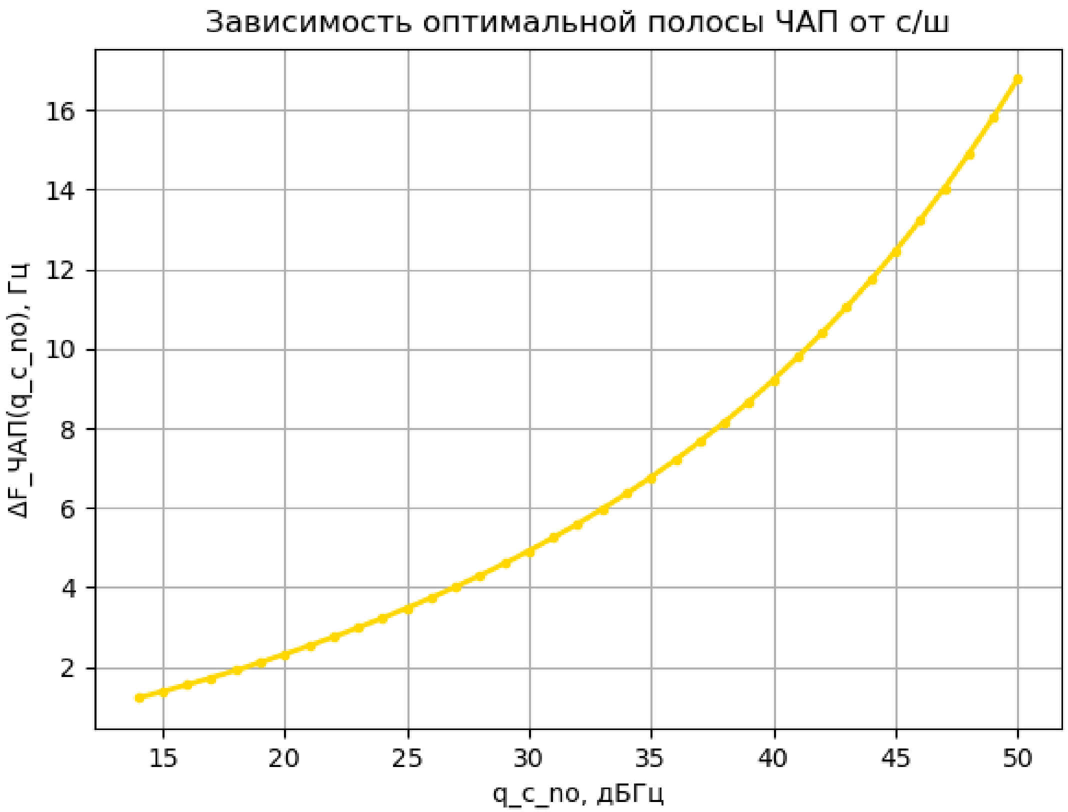

Рис.1.2 Зависимость оптимальной полосы ЧАП от с/ш

Оптимальная полоса ЧАП растёт с увеличением отношения с/ш.

2. Повторяем аналогичное моделирование при фиксированном отношении с/ш и изменяющемся среднеквадратическом ускорении - $q_{c/n0} = 10^{0.1(30\,ДбГц)}\left[Гц\right],\ \sigma_a = 1K\ 30\,м/c^2$:

Результаты моделирования:

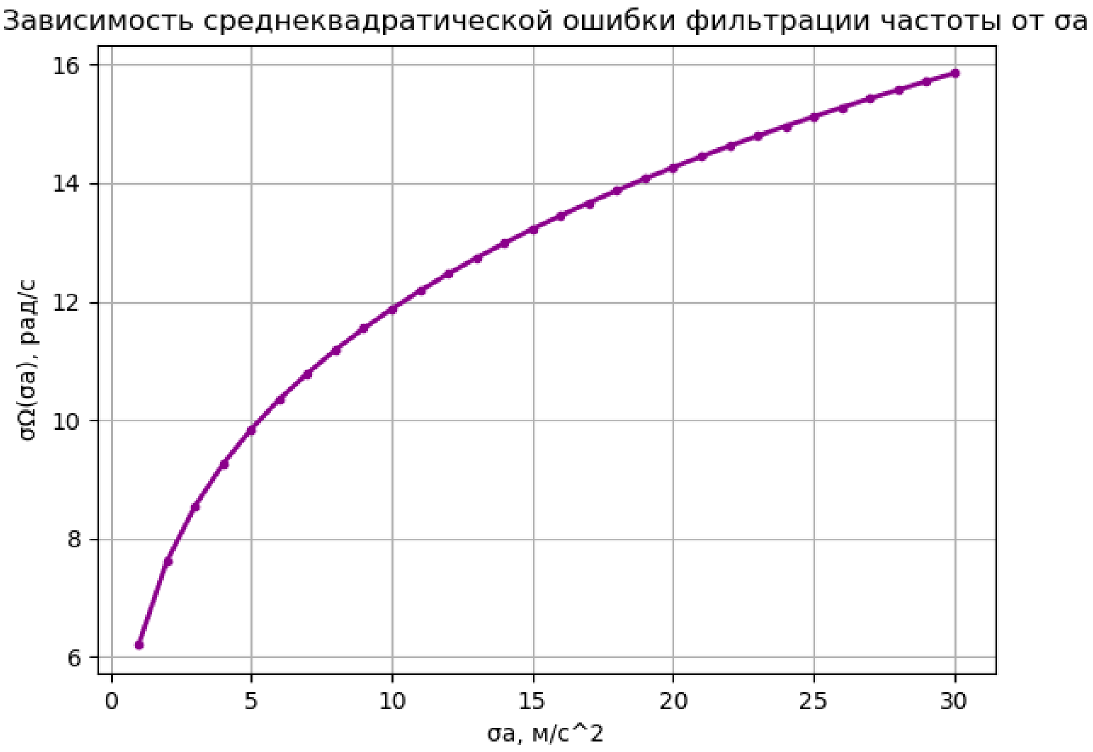

Рис.1.3 Зависимость среднеквадратической ошибки фильтрации частоты от среднеквадратического ускорения

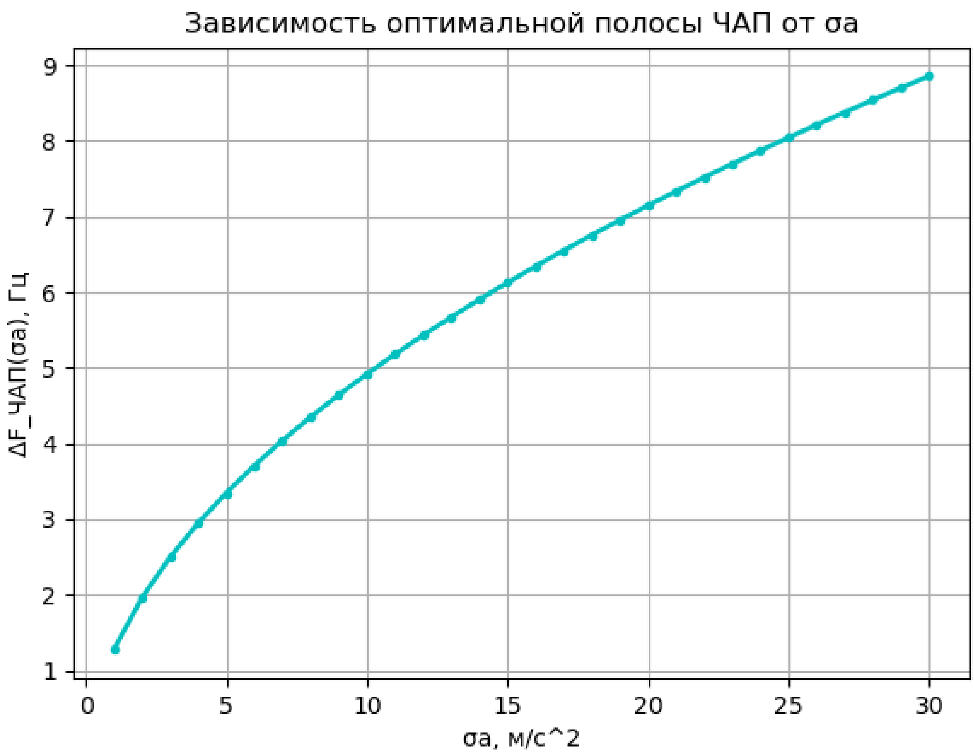

Рис.1.4 Зависимость оптимальной полосы ЧАП от среднеквадратического ускорения

С увеличением σ_a растут оба параметра.

Листинг программы

```
import math

import matplotlib.pyplot as plt

from scipy.integrate import quad

import numpy as np



# константы

c         = 3e8

w0        = 2 * math.pi * 1602e6

alpha     = 1

T         = 10e-3


q_c_n0_list           = []

N0_list               = []

sigma_11_list         = []

sigma_11_sigma_a_list = []

Ky_Omega_p_list       = []

delta_F_list          = []

delta_F_list_sigma_a  = []


"""------------------------------Меняется с/ш-------------------------------"""

sigma_a   = 10

S_ksi     = 2 * (sigma_a**2) * alpha * ((w0/c)**2)


for k in range(14, 51, 1):

    q_c_n0 = (10**(0.1*k))

    q_c_n0_list.append(q_c_n0)


    N0 = (2/(q_c_n0 * (T**2))) * (1 + (1/(2 * q_c_n0 * T)))

    N0_list.append(N0)


    D11_pt2 = (math.sqrt(1 + 2 * math.sqrt((S_ksi)/((alpha**2) * N0 ))) - 1)
```

```
    D11 = ((alpha * N0)/ 2) * (D11_pt2)


    sigma_11 = math.sqrt(D11)

    sigma_11_list.append(sigma_11)


    # попытка расчета полосы

    K1 = alpha * (math.sqrt(1 + 2 * math.sqrt((S_ksi)/((alpha**2) * N0))) - 1)


    K2 = (K1**2)/2


    def integrand(w):

        p        = (0 + 1j) * w

        K_f      = 1/p * (K1 + K2/(p + alpha))

        K_y_Omega = K_f/(1 + K_f)

        return ((abs(K_y_Omega))**2)

    # считаем интеграл - quad(integrand - интегрируемая функция, 0 - нижний предел, np.inf - верхний
беспредел)

    delta_F_pt2 = quad(integrand, 0, np.inf)[0]


    delta_F = (1/(2*math.pi)) * delta_F_pt2

    delta_F_list.append(delta_F)


plt.figure(1)

plt.plot(range(14, 51, 1), sigma_11_list, '.-', color = 'deeppink', linewidth = 2)

plt.xlabel('q_c_no, дБГц')

plt.ylabel('σΩ (q_c_no), рад/с')

plt.title('Зависимость среднеквадратической ошибки фильтрации частоты от с/ш')

plt.grid()

plt.show()


plt.figure(2)

plt.plot(range(14, 51, 1), delta_F_list, '.-', color = 'gold', linewidth = 2)
```

```
plt.xlabel('q_c_no, дБГц')

plt.ylabel('ΔF_ЧАП(q_c_no), Гц')

plt.title('Зависимость оптимальной полосы ЧАП от с/ш')

plt.grid()

plt.show()


"""---------------------------Меняется СКО ускорения------------------------"""

q_c_no_sigma_a = 10**(0.1*30)

N0_sigma_a     = (2 / (q_c_no_sigma_a * (T**2))) * (1 + (1/(2 * q_c_no_sigma_a * T)))


for sigma_a in range(1, 31, 1):

    S_ksi_sigma_a = 2 * (sigma_a**2) * alpha * ((w0/c)**2)


    D11_sigma_a_pt2 = (math.sqrt(1 + 2 * math.sqrt((S_ksi_sigma_a)/((alpha**2)*N0_sigma_a))) - 1)

    D11_sigma_a = ((alpha * N0_sigma_a)/2) * D11_sigma_a_pt2


    sigma_11_sigma_a  = math.sqrt(D11_sigma_a)

    sigma_11_sigma_a_list.append(sigma_11_sigma_a)


    # попытка расчета полосы

    K1_sigma_a = alpha * (math.sqrt(1 + 2 * math.sqrt((S_ksi_sigma_a)/((alpha**2) * N0_sigma_a))) - 1)


    K2_sigma_a = (K1_sigma_a**2)/2


    def integrand_sigma_a(w):

        p        = (0 + 1j) * w

        K_f      = 1/p * (K1_sigma_a + K2_sigma_a/(p + alpha))

        K_y_Omega = K_f/(1 + K_f)

        return ((abs(K_y_Omega))**2)

    # считаем интеграл - quad(integrand - интегрируемая функция, 0 - нижний предел, np.inf - верхний беспредел)

    delta_F_pt2_sigma_a = quad(integrand_sigma_a, 0, np.inf)[0]
```

```
    delta_F_sigma_a = 1/(2*math.pi) * delta_F_pt2_sigma_a
    delta_F_list_sigma_a.append(delta_F_sigma_a)

plt.figure(3)
plt.plot(range(1, 31, 1), sigma_11_sigma_a_list, '.-', color = 'darkmagenta', linewidth = 2)
plt.xlabel('σa, м/с^2')
plt.ylabel('σΩ(σa), рад/с')
plt.title('Зависимость среднеквадратической ошибки фильтрации частоты от σa')
plt.grid()
plt.show()

plt.figure(4)
plt.plot(range(1, 31, 1), delta_F_list_sigma_a, '.-', color = 'c', linewidth = 2)
plt.xlabel('σa, м/с^2')
plt.ylabel('ΔF_ЧАП(σa), Гц')
plt.title('Зависимость оптимальной полосы ЧАП от σa')
plt.grid()
plt.show()
```

Часть 2

Задача:

Смоделировать входное воздействие и оптимальную систему ЧАП в дискретном времени.

Начальные данные:

$$\Omega_k = \Omega_{k-1} + \nu_{k-1}T, T = 10\text{мс},$$

$$\nu_k = \nu_{k-1} \cdot (1 - \alpha T) + \alpha T \cdot \xi_{k-1}, \qquad \sigma_\xi^2 = \frac{S_\xi}{2T},$$

$$\tilde{y}_k = \Omega_k + \tilde{n}_k,\, \tilde{n}_k - \text{ ДБГШ с дисперсией } \sigma_n^2 = \frac{\tilde{N}_0\left(q_{c/no}\right)}{2T}$$

Решение:

Будем использовать алгоритм оптимальной линейной фильтрации для векторных наблюдений и процессов (фильтр Калмана).

Инициализация фильтра

1) Запишем вектор состояния:

$$\mathbf{x} = \begin{vmatrix} \Omega \\ \nu \end{vmatrix},$$

2) Динамика вектора состояния определяется выражением:

$$\mathbf{x}_k = \mathbf{F}_{k-1}\mathbf{x}_{k-1} + \mathbf{G}_{k-1}\boldsymbol{\xi}_{k-1}, \text{ где}$$

$$\mathbf{F}_{k-1} = \begin{bmatrix} 1 & T \\ 0 & 1 - \alpha T \end{bmatrix},$$

$$\mathbf{G}_{k-1} = \begin{bmatrix} 0 \\ \alpha T \end{bmatrix},$$

$\boldsymbol{\xi}_{k-1}$ − векторный ДБГШ с матрицей дисперсий $\mathbf{D}_{\xi} = \left[\sigma_{\xi}^{2}\right]$.

3) Наблюдения фильтра:

$$\boldsymbol{y}_k = \mathbf{H}_k \mathbf{x}_k + \boldsymbol{n}_k,$$

наблюдениями фильтра является только наблюдения частоты Доплера, то есть в нашем случае наблюдения не являются векторными. Матрица $\mathbf{H}_k$ определяется как:

$$\mathbf{H}_k = [1 \quad 0],$$

а $\boldsymbol{n}_k -$ шум наблюдений с матрицей дисперсий $\mathbf{D}_{\mathrm{n}} = [\sigma_n^2]$.

Фильтрация

- Экстраполяция:
  1) Экстраполяция вектора состояния:

$$\tilde{\mathbf{x}}_k = \mathbf{F}_{k-1} \hat{\mathbf{x}}_{k-1},$$

  2) Экстраполяция матрицы дисперсии ошибок:

$$\widetilde{\mathbf{D}}_{\mathrm{x},k} = \mathbf{F}_{k-1} \mathbf{D}_{k-1} \mathbf{F}_{k-1}^{T} + \mathbf{G}_{k-1} \mathbf{D}_{\xi} \mathbf{G}_{k-1}^{T}.$$

- Оценивание:
  1) Коэффициент фильтра:

$$\mathbf{K}_k = \widetilde{\mathbf{D}}_{\mathrm{x},k} \mathbf{H}_k^{T} \left(\mathbf{H}_k \widetilde{\mathbf{D}}_{\mathrm{x},k} \mathbf{H}_k^{T} + \mathbf{D}_{\mathrm{n}}\right)^{-1},$$

  в нашем случае размерность коэффициента $\mathbf{K}_k - 2 \times 1$.

  2) Скорректированная матрица дисперсии ошибок оценивания вектора состояния:

$$\widehat{\mathbf{D}}_{\mathrm{x},k} = (\mathbf{I} - \mathbf{K}_k \mathbf{H}_{k-1}) \widetilde{\mathbf{D}}_{\mathrm{x},k}, \text{ где}$$

  $\mathbf{I} -$ единичная матрица размерностью 2×2.

3) Скорректированная оценка вектора состояния:

$$\hat{\mathbf{x}}_k = \tilde{\mathbf{x}}_k + \mathbf{K}_k(\boldsymbol{y}_k - \mathbf{H}_k\tilde{\mathbf{x}}_k).$$

Моделирование

Зададим недостающие для моделирования начальные параметры:

1)

$$\mathbf{x}_0 = \begin{vmatrix} 100 \\ 100 \end{vmatrix},$$

2)

$$\hat{\mathbf{x}}_0 = \begin{vmatrix} 0 \\ 0 \end{vmatrix},$$

3)

$$q_{c/no} = 10^{0.1\cdot 30}\ \text{Гц},$$

4)

$$\sigma_a = 10\,\frac{\text{м}}{\text{с}^2},$$

5)

$$D_0 = \begin{bmatrix} \left(34\,\frac{\text{рад}}{\text{с}}\right)^2 & 0 \\ 0 & \left(340\,\frac{\text{рад}}{\text{с}}\right)^2 \end{bmatrix}.$$

Результаты

1.

Рис.1 Зависимость истинной доплеровской частоты от времени

Построим совместный график истинной доплеровской частоты с её оценкой:

Рис. 2 Зависимость истинной доплеровской частоты и ее оценки от времени

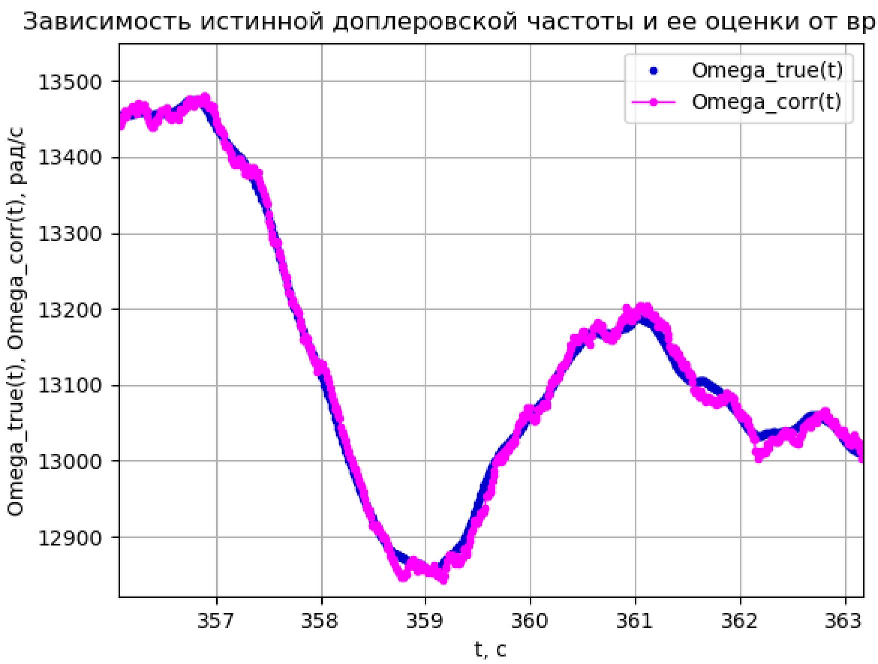

Рис. 3 Зависимость истинной доплеровской частоты и ее оценки от времени (увеличен масштаб)

Видно, что оценка частоты почти повторяет истинную частоту Доплера.

2.

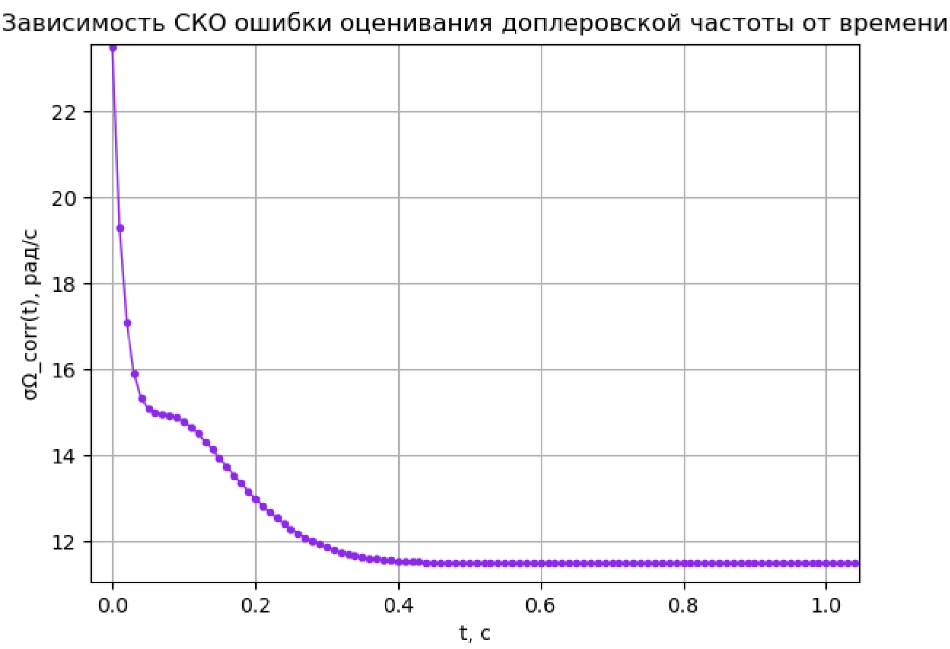

Рис.4 Зависимость среднеквадратической ошибки фильтрации частоты от времени

3.

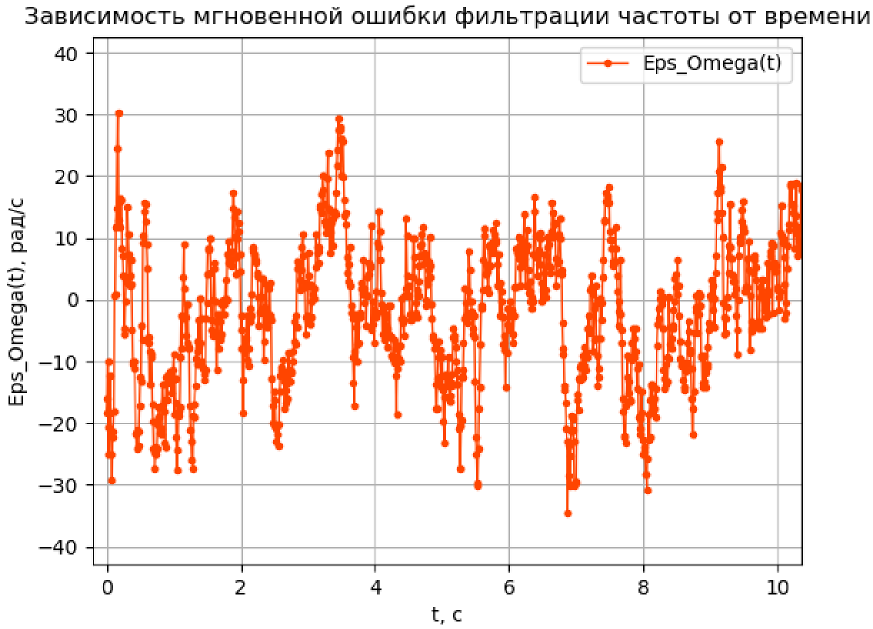

Рис.5 Зависимость мгновенной ошибки фильтрации частоты от времени

4. Построим вместе зависимости дисперсии ошибки фильтрации частоты от времени и мгновенной ошибки фильтрации частоты от времени, рассмотрим участок с установившимся режимом:

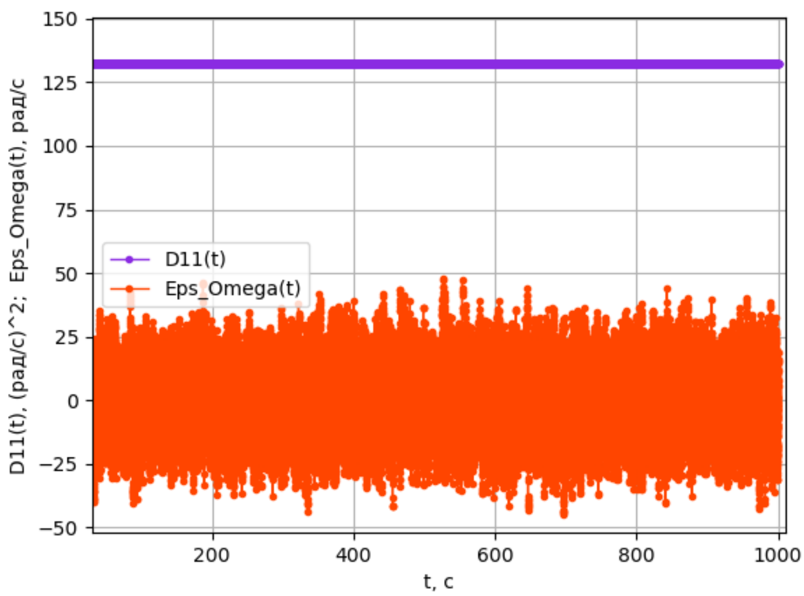

Рис. 6 Зависимости дисперсии ошибки фильтрации частоты и мгновенной ошибки фильтрации частоты от времени

Из графика мгновенной ошибки фильтрации частоты определим ее СКО: согласно правилу трёх сигм выберем $3\sigma_{\varepsilon\Omega} \cong 35$ рад/с, следовательно, $\sigma_{\varepsilon\Omega} \cong 11.(6)$ рад/с, тогда $D_{\varepsilon\Omega} = \sigma_{\varepsilon\Omega}^2 \cong 136.(1)$ (рад/с)^2. Это значение примерно совпадает со значением дисперсии ошибки фильтрации частоты $D_\Omega \cong 132.253$ (рад/с)^2.

Листинг программы

```
import numpy as np

import math

from numpy.linalg import inv

import matplotlib.pyplot as plt


"""-----------------------------Параметры моделирования---------------------"""

T      = 10e-3

M      = 100000

c      = 3* 1e8

w0     = 2 * math.pi * 1602e6

t_list = []


"""-----------------------Моделируем шум------------------------------------"""

# формирующий шум скорости

alpha     = 1

sigma_a   = 10

S_ksi     = 2 * (sigma_a**2) * alpha * ((w0/c)**2)

sigma_ksi = math.sqrt((S_ksi)/(2*T))

ksi_list  = np.random.normal(loc = 0.0, scale = sigma_ksi * 1, size = M)


ksi_k     = np.array([[ksi_list[0]]])


# шум наблюдения

q_c_n0    = 10 ** (0.1 * 30)

N0        = ((2) / (q_c_n0 * (T**2))) * (1 + ((1)/(2 * q_c_n0 * T)))

sigma_n   = math.sqrt((N0)/(2 * T))

n_list    = np.random.normal(loc = 0.0, scale = sigma_n * 1, size = M)


"""---------------------------Инициализация--------------------------------"""

# начальные значения истинных параметров

Omega_true      = 100

v_true          = 100
```

```
Omega_true_list = []
v_true_list     = []

# начальные значения скорректированных оценок
Omega_corr          = 0
v_corr              = 0
Omega_corr_list     = []
v_corr_list         = []
sigma_Omega_corr_list = []
Eps_Omega_list      = []
D11_list            = []

# матрицы фильтра
X_true   = np.array([[Omega_true],\
                     [v_true]])

X_corr   = np.array([[Omega_corr],\
                     [v_corr]])                    # 2x1

I       = np.eye(2)                                # 2x2

H       = np.array([[1, 0]])                       # 1x2

D_x_corr = np.array([[34**2,      0],\
                     [    0, 340**2]])             # 2x2

G       = np.array([[0],\
                    [alpha * T]])                  # 2x1

D_ksi    = np.array([[sigma_ksi**2]])              # 1x1

D_n      = np.array([[sigma_n**2]])
```

```
F       = np.array([[1,             T],\
                    [0, (1 - alpha * T)]])                  # 2x2

for k in range(0, M, 1):

    t = k * T
    t_list.append(t)

    """---------------------Входное воздействие---------------------------"""
    X_true      = F.dot(X_true) + G.dot(ksi_k)
    Omega_true_list.append(X_true[0])
    v_true_list.append(X_true[1])

    ksi_k       = np.array([[ksi_list[k]]])

    y           = H.dot(X_true) + n_list[k]

    """-----------------------------Фильтрация----------------------------"""
    """---------------------------Экстраполяция---------------------------"""
    X_extr      = F.dot(X_corr)                             # 2x1

    D_x_extr_pt1 = (F.dot(D_x_corr)).dot(F.transpose())          # 2x2

    D_x_extr_pt2 = (G.dot(D_ksi)).dot(G.transpose())             # 2x2

    D_x_extr    = D_x_extr_pt1 + D_x_extr_pt2                    # 2x2

    """-----------------------------Коррекция-----------------------------"""
    K           = (D_x_extr.dot(H.transpose())).dot(inv(((H.dot(D_x_extr)).dot(H.transpose())) + D_n))  # 2x1

    D_x_corr    = (I - K.dot(H)).dot(D_x_extr)                   # 2x2

    X_corr      = X_extr + K.dot(y - H.dot(X_extr))              # 2x1
```

```
    # мгновенная ошибка фильтрации
    Eps_Omega    = X_corr[0] - X_true[0]
    Eps_Omega_list.append(Eps_Omega)

    Omega_corr   = X_corr[0]
    Omega_corr_list.append(Omega_corr)

    v_corr       = X_corr[1]
    v_corr_list.append(v_corr)

    sigma_Omega_corr = math.sqrt(D_x_corr[0,0])
    sigma_Omega_corr_list.append(sigma_Omega_corr)

    D11 = D_x_corr[0,0]
    D11_list.append(D11)

plt.figure(1)
plt.plot(t_list[0::], Omega_true_list[0::], '.', color = 'mediumblue', linewidth = 1)
plt.plot(t_list[0::], Omega_corr_list[0::], '.-', color = 'magenta', linewidth = 1)
plt.xlabel('t, c')
plt.ylabel('Omega_true(t), Omega_corr(t), рад/с')
plt.legend(['Omega_true(t)', 'Omega_corr(t)'])
plt.title('Зависимость истинной доплеровской частоты и ее оценки от времени')
plt.grid()
plt.show()

plt.figure(2)
plt.plot(t_list[0::], sigma_Omega_corr_list[0::], '.-', color = 'blueviolet', linewidth = 1)
plt.xlabel('t, c')
plt.ylabel('σΩ_corr(t), рад/с')
plt.title('Зависимость СКО ошибки оценивания доплеровской частоты от времени')
plt.grid()
```

```
plt.show()


plt.figure(3)
plt.plot(t_list[0::], Eps_Omega_list[0::], '.-', color = 'orangered', linewidth = 1)
plt.xlabel('t, c')
plt.ylabel('Eps_Omega(t), рад/c')
plt.legend(['Eps_Omega(t)'])
plt.title('Зависимость мгновенной ошибки фильтрации частоты от времени')
plt.grid()
plt.show()


plt.figure(4)
plt.plot(t_list[0::], D11_list[0::], '.-', color = 'blueviolet', linewidth = 1)
plt.plot(t_list[0::], Eps_Omega_list[0::], '.-', color = 'orangered', linewidth = 1)
plt.xlabel('t, c')
plt.ylabel('D11(t), (рад/c)^2;  Eps_Omega(t), рад/c')
plt.legend(['D11(t)','Eps_Omega(t)'])
plt.grid()
plt.show()


plt.figure(5)
plt.plot(t_list[0::], Omega_true_list[0::], '.', color = 'mediumblue', linewidth = 1)
plt.xlabel('t, c')
plt.ylabel('Omega_true(t), рад/c')
plt.title('Зависимость истинной доплеровской частоты от времени')
plt.grid()
plt.show()
```