САНКТ-ПЕТЕРБУРГСКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ

Механика и математическое моделирование

Гидроаэромеханика

Каплин Борис

Измерение стационарных и нестационарных аэродинамических характеристик тела малого удлинения

Магистерская диссертация

Научный руководитель:

Доктор физико-математических наук

Рябинин А. Н.

Рецензент:

Кандидат технических наук

Ежов О.Н.

Санкт-Петербург

2018

SAINT-PETERSBURG STATE UNIVERSITY

Mechanics and Mathematical Modelling

Hydroaeromechanics

Kaplin Boris

Measurement of steady and unsteady aerodynamic characteristics of the body of small aspect ratio

Master's Thesis

Scientific supervisor:

Doctor of Physical and Mathematical Sciences

Ryabinin A.N.

Reviewer:

Candidate of Technical Sciences

Yezhov O. N.

Saint-Petersburg

2018

СОДЕРЖАНИЕ

РЕФЕРАТ

Каплин Борис. Выпускная квалификационная работа «Измерение стационарных и нестационарных аэродинамических характеристик тела малого удлинения». Санкт-Петербург: СПбГУ, Математико-механический факультет, направление механики и математического моделирования, кафедра гидроаэромеханики, 2018, 34 с.

В работе рассматривается поведение тел малого удлинения в потоке воздуха. Рассматриваются поступательные и вращательные колебания такого тела. Вычисляются аэродинамические коэффициенты. Строится модель зависимости коэффициента затуханий вращательных колебаний от амплитуды колебаний. Рассматривается проблема поступательного галопирования прямой призмы в потоке воздуха. В 3 главе подробна описана программа по обработке данных экспериментов, написанная на языке программирования Python.

ABSTRACT

Kaplin Boris. Graduation qualification work «Measurement of steady and unsteady aerodynamic characteristics of the body of small aspect ratio». St. Petersburg: St. Petersburg State University, Faculty of Mathematics and Mechanics, Direction of Mechanics and Mathematical Modeling, Department of Hydroaeromechanics, 2018, 34 p.

The behavior of small elongation bodies in the air flow is considered in this paper. The translational and rotational vibrations of such a body are considered. The aerodynamic coefficients are calculated. A model is constructed for the dependence of the damping coefficient of rotational vibrations on the amplitude of the oscillations. The problem of translational galloping of a direct prism in a stream of air is considered. In Chapter 3, the program for processing experimental data, written in the Python programming language, is described in detail.

ВВЕДЕНИЕ

Проблема галопирования плохообтекаемых тел начала развиваться с середины XX века. Первые работы в этой области представили Паркинсон и Брукс [4], Паркинсон и Смит [5], Новак [6]. В этих работах рассматривалось галопирование прямой призмы с квадратным и прямоугольным поперечным сечением. Далее, много работ было посвящено исследованию галопирования призм с круглым и другими поперечными сечениями.

Причиной активизации исследований в данной сфере стало разрушение некоторых построек под действием ветра. На сегодняшний день, моделирование галопирования позволяет обеспечивать безопасность, например, при построении гондол подвесных канатных дорог. Они могут раскачиваться под действием ветра в процессе эксплуатации, что может привести к непоправимым последствиям.

В данной работе исследовалось поступательное галопирование и вращательные колебания прямой призмы с квадратным сечением.

1. ИССЛЕДОВАНИЕ ПОВЕДЕНИЯ КРЫЛА В ПОТОКЕ ВОЗДУХА

1.1 Исследование динамической устойчивости крыла

При движении крыла в воздухе оно может колебаться из-за действия на него аэродинамических сил. При длительном колебании, тело может отклониться от траектории своего движения. Для того, чтобы охарактеризовать динамическую устойчивость крыла в потоке, обычно пользуются вращательными производными момента [3]. В данной работе не использовалось квазистационарное приближение [1, 2], так как аэродинамические силы, действующие на крыло, зависят не только от мгновенных углов атаки и скольжения. Математическая модель включает в себя параметры, которые определяются из эксперимента.

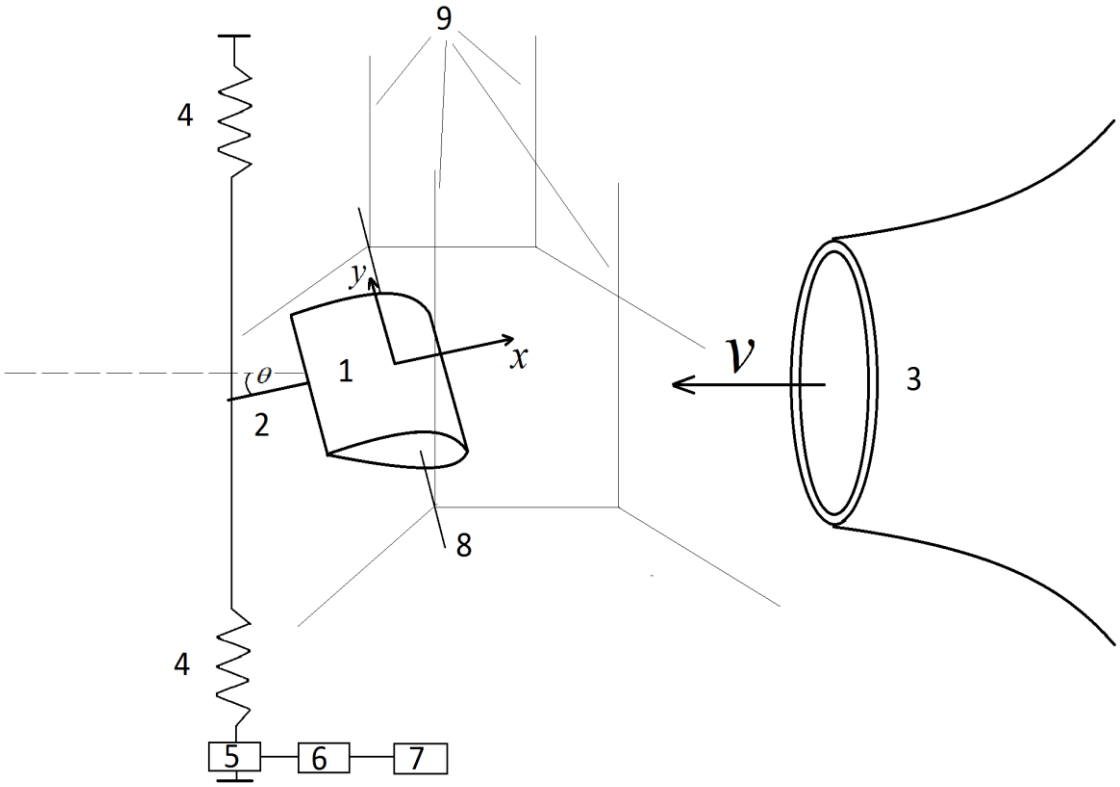

Рисунок 1.1.1. Экспериментальная установка

Эксперименты проводились в дозвуковой аэродинамической трубе. Скорость потока регулировалась от 0 до 15 м/с. Макет крыла закреплялся на проволочной подвеске. К кромке крыла прикреплялась хвостовая державка, к которой были прикреплены две пружины. Верхняя пружина крепилась к механизму изменения угла атаки, а нижняя – к неподвижному тензопреобразователю. Сигнал поступал на PC-осцилограф, связанный с компьютером. Показания записывались в текстовый файл с частотой 100 Гц. Для каждого эксперимента было сделано 1700 измерений. При градуировке на державку навешивался груз известной массы и определялось отклонение державки от положения равновесия.

Движение данной модели описывается уравнением

$$I_z\ddot{\theta} + r\dot{\theta} + k\theta = \left(m_z^{\omega} + m_z^{\dot{\alpha}}\right)\frac{qSb^2}{v}\dot{\theta} + m_z^{\alpha}qSL\theta, \qquad (1.1.1)$$

где I_z – компонента момента инерции, которая соответствует одной из главных осей инерции тела, θ – угол тангажа, k – приведенная жесткость упругой системы, $r\dot{\theta}$ – вязкое сопротивление подвески, $q = \frac{\rho v^2}{2}$ – динамический напор, ρ – плотность воздуха, $S = Lb$ – характерная площадь, v – скорость воздушного потока, m_z^{ω}, $m_z^{\dot{\alpha}}$, m_z^{α} – вращательные производные коэффициента момента тангажа по угловой скорости тангажа, скорости изменения угла атаки и углу атаки соответственно.

Принимая, что момент упругих сил подвески много больше компоненты момента аэродинамических сил, получаем

$$\ddot{\theta} + \omega^2\theta = \mu\dot{\theta}, \qquad \mu = \left(m_z^{\omega} + m_z^{\dot{\alpha}}\right)\frac{qSb^2}{I_z v} - \frac{r}{I_z}, \qquad \omega^2 = \frac{k}{I_z} \qquad (1.1.2)$$

Будем искать решение (1.1.2) в виде

$$\theta = \theta_0 \exp(\lambda t), \qquad \lambda = \eta + ip \tag{1.1.3}$$

Подставив (1.1.3) в уравнение (1.1.2) получим

$$\mu = 2\eta, \qquad \omega^2 = \eta^2 + p^2 \approx p^2 \tag{1.1.4}$$

После несложных преобразований коэффициент затухания можно представить в виде

$$\eta = A + Bv, \quad A = -\frac{r}{2I_z}, \quad B = \left(m_z^{\omega} + m_z^{\dot{\alpha}}\right)\frac{\rho S b^2}{4I_z} \tag{1.1.5}$$

Коэффициенты модели A и B находились методом наименьших квадратов.

Далее представлены графики зависимости коэффициента затухания от скорости набегающего потока (1.1.5).

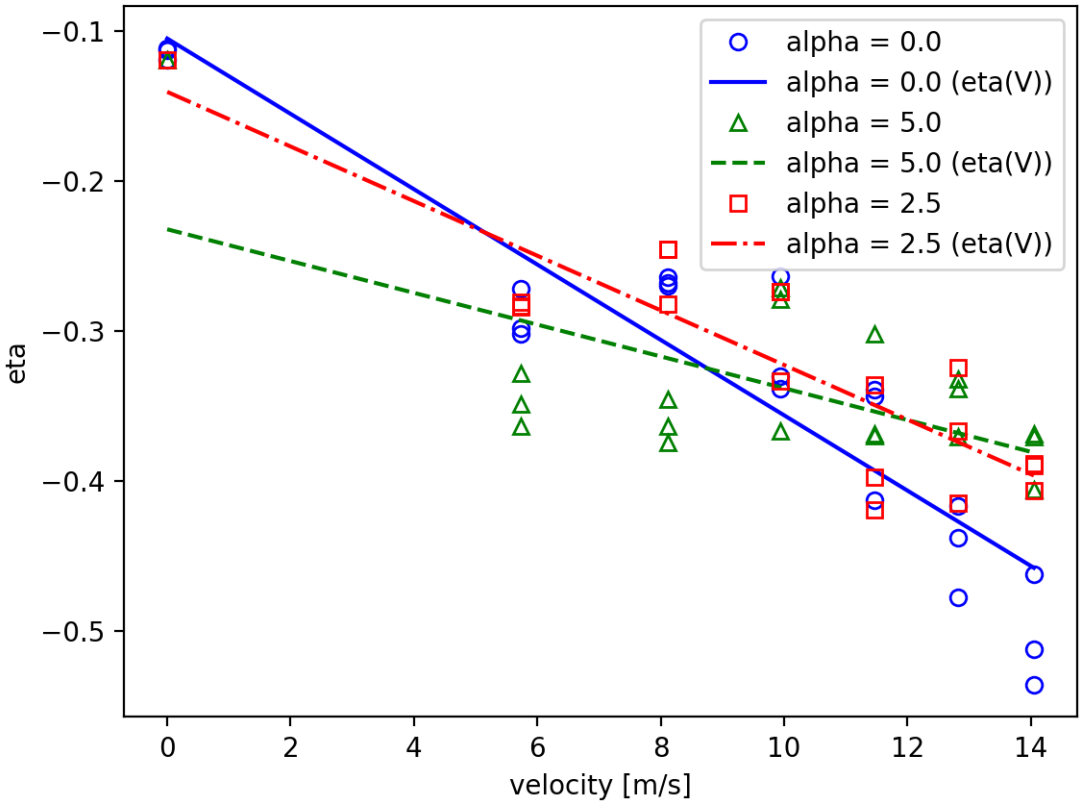

Рисунок 1.1.2. Зависимость коэффициента затухания от скорости

Таким образом, искомые вращательные производные находятся из третьего уравнения в (1.1.5)

$$m_z^{\omega} + m_z^{\dot{\alpha}} = \frac{4BI_z}{\rho Sb^2} = \frac{4Bk}{\rho Sb^2\omega^2} \qquad (1.1.6)$$

Результаты расчета периода колебаний и вращательных производных представлены в табл.1.

α, град	Период T, с	B, 1/м	$m_z^{\omega} + m_z^{\dot{\alpha}}$
0	0.243	-0.025102	-0.523
2,5	0.243	-0.018197	-0.379
5	0.243	-0.010591	-0.220

Таблица 1

Период колебаний мало изменяется при различных скоростях набегающего потока и углах атаки. На графиках видно, что при угле атаки 5 градусов точки не лежат на одной прямой. Следовательно, принятая математическая модель плохо описывает результаты экспериментов. Для остальных значений угла атаки точки ложатся на прямые с небольшим разбросом.

1.2 Аэродинамические коэффициенты крыла

Аэродинамические коэффициенты определялись из эксперимента. Эксперименты проводились в дозвуковой аэродинамической трубе при скорости набегающего потока равной 26 м/с.

Далее приведены графики аэродинамических коэффициентов для данной модели. Коэффициенты построены в зависимости от угла атаки, который изменялся от -6 до 26 градусов с шагом 2 градуса.

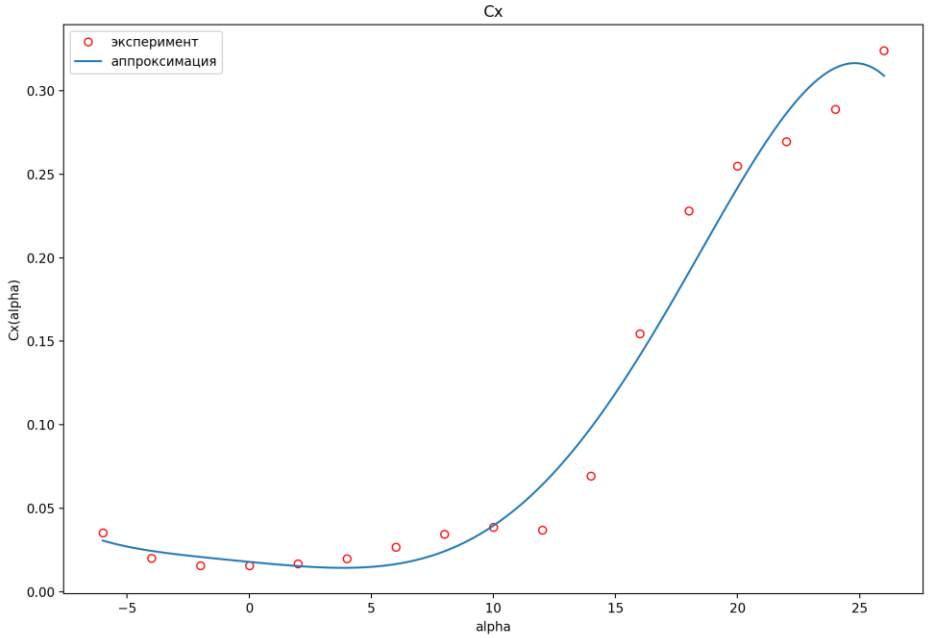

Рисунок 1.2.1. Коэффициент силы лобового сопротивления

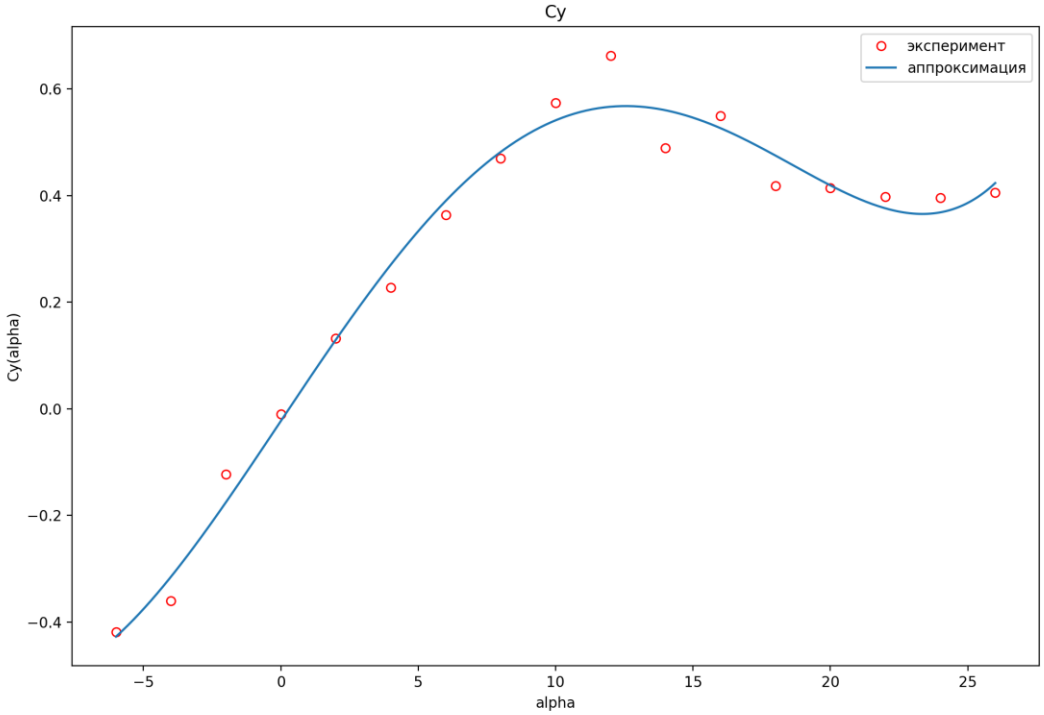

Рисунок 1.2.2. Коэффициент подъемной силы

По графикам видно, что оптимальный угол атаки, который обеспечивает максимальную подъемную силу и минимальное лобовое сопротивление, составляет около 9-10 градусов.

Далее будет рассмотрено поведение плохообтекаемого тела в потоке воздуха.

2. ИССЛЕДОВАНИЕ ПОВЕДЕНИЯ ПЛОХООБТЕКАЕМОГО ТЕЛА МАЛОГО УДЛИНЕНИЯ В ПОТОКЕ ВОЗДУХА

2.1 Исследование динамической устойчивости прямой призмы

При обдувании плохообтекаемого тела потоком воздуха оно также может колебаться из-за действия на него аэродинамических сил. Для того, чтобы охарактеризовать динамическую устойчивость такого тела в потоке, как и в предыдущем случае, обычно пользуются вращательными производными момента [3]. Если аэродинамические силы, действующие на тело, зависят только от мгновенных углов атаки, то достаточно пользоваться квазистационарным приближением [1, 2]. Однако, из-за того, что аэродинамические силы, действующие на плохообтекаемые тела, зависят еще и от производных мгновенных углов атаки, квазистационарным приближением пользоваться нельзя.

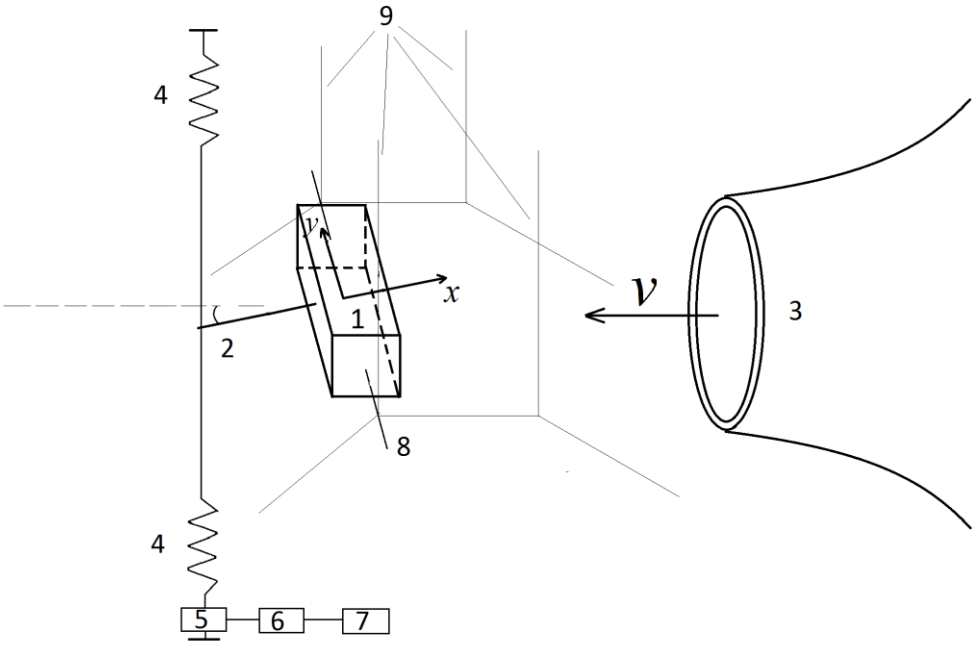

Рисунок 2.1.1. Экспериментальная установка

Эксперименты проводились в дозвуковой аэродинамической трубе. Скорость потока регулировалась от 0 до 23.7 м/с. Макет плохообтекаемого тела закреплялся на проволочной подвеске. Сзади прикреплялась державка, к которой были прикреплены две пружины. Верхняя пружина крепилась к механизму изменения угла атаки, а нижняя – к неподвижному тензопреобразователю. Сигнал поступал на PC-осциллограф, который связан с компьютером. Для каждого показания скорости проводилось 2 считывания: в режиме осциллографа и в режиме самописца. Показания записывались в текстовый файл с частотой 1250 Гц и 100 Гц соответственно. При градуировке на державку навешивался груз известной массы и определялось отклонение державки от положения равновесия.

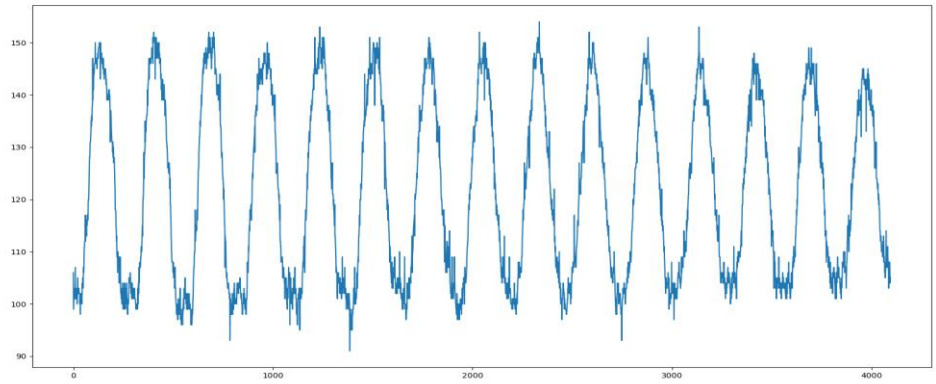

Рисунок 2.1.2. Пример записи сигнала в режиме осциллографа

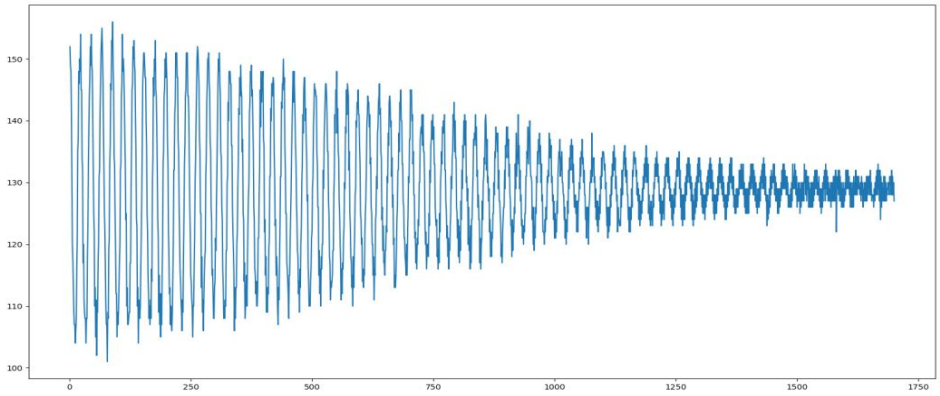

Рисунок 2.1.3. Пример записи сигнала в режиме самописца

На рис. 2.1.2 представлен пример записи колебаний призмы в режиме осциллографа. На рис. 2.1.3 – пример записи колебаний в режиме самописца.

На графиках можно увидеть, что амплитуда колебаний в режиме осциллографа практически не меняется за время измерения, поэтому зависимость между коэффициентом затухания и амплитудой определялась только исходя из данных самописца.

При обработке сигналов, предполагалось, что измеренный угол поворота призмы от времени является суммой гармонических функций:

$$\beta_i = B\cos(\omega t_i) + C\sin(\omega t_i) + E + \xi_i \tag{2.1.1}$$

Пусть n – это количество считываний за один период. Тогда умножая на синус и на косинус равенство (2.1.1), получим следующее

$$\begin{aligned}
\frac{1}{n}\sum_{i=1}^{n}\beta_i\cos(\omega t_i) &= B\frac{1}{n}\sum_{i=1}^{n}\cos^2(\omega t_i) + C\frac{1}{n}\sum_{i=1}^{n}\cos(\omega t_i)\sin(\omega t_i) + \\
&E\frac{1}{n}\sum_{i=1}^{n}\cos(\omega t_i) + \frac{1}{n}\sum_{i=1}^{n}\xi_i\cos(\omega t_i), \\
\frac{1}{n}\sum_{i=1}^{n}\beta_i\sin(\omega t_i) &= B\frac{1}{n}\sum_{i=1}^{n}\cos(\omega t_i)\sin(\omega t_i) + C\frac{1}{n}\sum_{i=1}^{n}\sin^2(\omega t_i) + \\
&E\frac{1}{n}\sum_{i=1}^{n}\sin(\omega t_i) + \frac{1}{n}\sum_{i=1}^{n}\xi_i\sin(\omega t_i).
\end{aligned} \tag{2.1.2}$$

Предполагая число n достаточно большим, суммы в этих уравнениях, которые содержат только тригонометрические функции, могут быть вычислены.

$$\begin{aligned}
&\frac{1}{n}\sum_{i=1}^{n}\cos^2(\omega t_i) \approx 0.5, \qquad \frac{1}{n}\sum_{i=1}^{n}\sin^2(\omega t_i) \approx 0.5, \\
&\frac{1}{n}\sum_{i=1}^{n}\cos(\omega t_i)\sin(\omega t_i) \approx 0, \qquad \frac{1}{n}\sum_{i=1}^{n}\cos(\omega t_i) \approx 0, \\
&\frac{1}{n}\sum_{i=1}^{n}\sin(\omega t_i) \approx 0, \qquad \frac{1}{n}\sum_{i=1}^{n}\xi_i\cos(\omega t_i) \approx 0.
\end{aligned} \tag{2.1.3}$$

Таким образом, из уравнений можно найти коэффициенты и может быть посчитана амплитуда колебаний.

$$A = \sqrt{B^2 + C^2} \tag{2.1.4}$$

Движение данной модели описывается уравнением

$$I\ddot{\beta} = L_a + L_s, \tag{2.1.5}$$

где I – момент инерции, L_a – момент аэродинамических сил, L_s – момент упругих сил и сил трения. Математическая модель, которая используется в данной работе, подробно описана в [7]

Компоненты моментов сил могут быть записаны в виде

$$L_a = \frac{\rho v^2}{2} LS\left(m + \frac{L\dot{\beta}}{v} m^{\dot{\beta}} \right), \qquad L_s = -kl^2\beta - k_1\dot{\beta}. \tag{2.1.6}$$

В формуле (2.1.6) ρ – плотность воздуха, v – скорость набегающего потока, L – длина призмы, S – характерная площадь, m, $m^{\dot{\beta}}$ – коэффициент момента аэродинамической силы и коэффициент аэродинамической производной, k – жесткость пружины, l – расстояние от оси вращения до конца державки, k_1 – коэффициент вязкого трения.

Коэффициент момента пропорционален β, а вращательная производная считается по такой формуле

$$m = -C_\beta\beta, \qquad m^{\dot{\beta}} = C_{\dot{\beta}}\left(1 - \delta\beta^2\right) \tag{2.1.7}$$

Выражения (2.1.7) содержат безразмерные величины $C_\beta, C_{\dot{\beta}}, \delta$. После подстановки (2.1.6, 2.1.7) в (2.1.5) получаем уравнение

$$\ddot{\beta} + \omega^2\beta + \Omega^2\beta = \mu\frac{v}{L}\left(1 - \delta\beta^2\right)\dot{\beta} - \mu k_2\dot{\beta}, \tag{2.1.8}$$

где

$$\omega^2 = \frac{kl^2}{L}, \qquad \Omega^2 = \frac{\rho v^2}{2I} LSC_\beta, \qquad \mu = \frac{\rho L^3 S}{2I} C_{\dot{\beta}}, \qquad k_2 = \frac{k_1}{\mu I}$$

Предполагаем, что компонента момента аэродинамических сил, содержащая Ω^2, много меньше момента упругих сил подвески ω^2. Тогда (2.1.8) перепишется в виде

$$\ddot{\beta} + \omega^2 \beta = \mu \left[\frac{v}{L}\left(1 - \delta\beta^2\right) - k_2 \right] \dot{\beta} \tag{2.1.9}$$

Если принять, что μ – малый параметр, то уравнение (2.1.9) может быть решено методом Крылова-Боголюбова. В результате получим дифференциальные уравнения колебаний для медленно меняющейся амплитуды A и фазы φ:

$$\dot{A} = \frac{A\mu}{2}\left[\left(\frac{v}{L} - k_2 \right) - \frac{v}{L} A^2 \frac{\delta}{4} \right], \quad \dot{\varphi} = 0. \tag{2.1.10}$$

В случае стационарных колебаний с постоянной амплитудой, производная $\dot{A}$ обращается в ноль и

$$A^2 = \frac{4}{\delta} - \frac{4k_2 L}{\delta} \frac{1}{v} \tag{2.1.11}$$

Поэтому A^2 является линейной функцией $\frac{1}{v}$.

Первое уравнение (2.1.10) можно переписать в таком виде

$$\eta = \frac{d \ln A}{dt} = \frac{\mu}{2}\left[\left(\frac{v}{L} - k_2 \right) - \frac{v}{L} A^2 \frac{\delta}{4} \right] \tag{2.1.12}$$

Коэффициенты данной модели находились методом наименьших квадратов.

Далее представлены графики зависимости логарифмического декремента затухания от амплитуды колебаний при разных скоростях набегающего потока. Точками показаны данные эксперимента, линиями – вычисленные значения по (2.1.12).

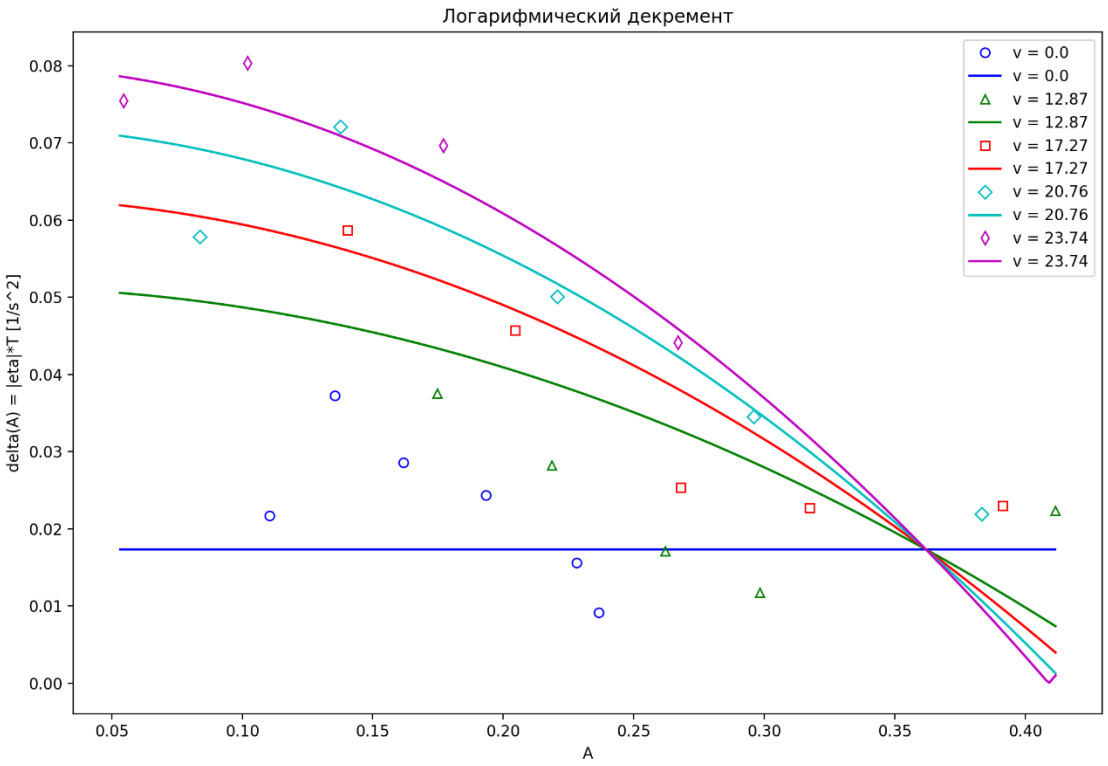

Рисунок 2.1.4. Логарифмический декремент

$$(\delta = |\eta| \mathrm{T})$$

При амплитуде меньшей 0.05 имеют место большие погрешности в вычислении коэффициентов модели из-за чего рассчитанный по (2.1.12) коэффициент затухания сильно отличается от действительного. Поэтому соответствующие точки на график не нанесены.

По графикам на рис. (2.1.4) можно увидеть, что модель, представленная в [7] достаточно хорошо аппроксимирует данные эксперимента. В данном случае, модель (2.1.12) лучше проявляет себя при больших скоростях набегающего потока. Таким образом, такой математической моделью можно

пользоваться, если нужно оценить, как зависит коэффициент затухания от амплитуды колебаний.

2.2 Определение аэродинамических коэффициентов и качественное поведение прямой призмы при поступательных колебаниях в потоке воздуха.

Для определения качественного поведения призмы в потоке при разных скоростях потока, был сделан эксперимент, результатом которого было определения аэродинамического коэффициента нормальной силы C_n. Для его определения находились коэффициенты лобового сопротивления C_x и подъемной силы C_y.

Определение аэродинамических коэффициентов проводилось, используя хорошо изученное квазистационарное приближение [1, 2]. Эксперименты проводились в дозвуковой аэродинамической трубе при скорости набегающего потока 25.7 м/с. Угол атаки изменялся от -20 до 20 градусов с шагом 2 градуса.

Таким образом, получены следующие графики для аэродинамических коэффициентов. Как и ранее, точками нанесены данные эксперимента, а линиями – аппроксимации данных эксперимента полиномами 5-й степени.

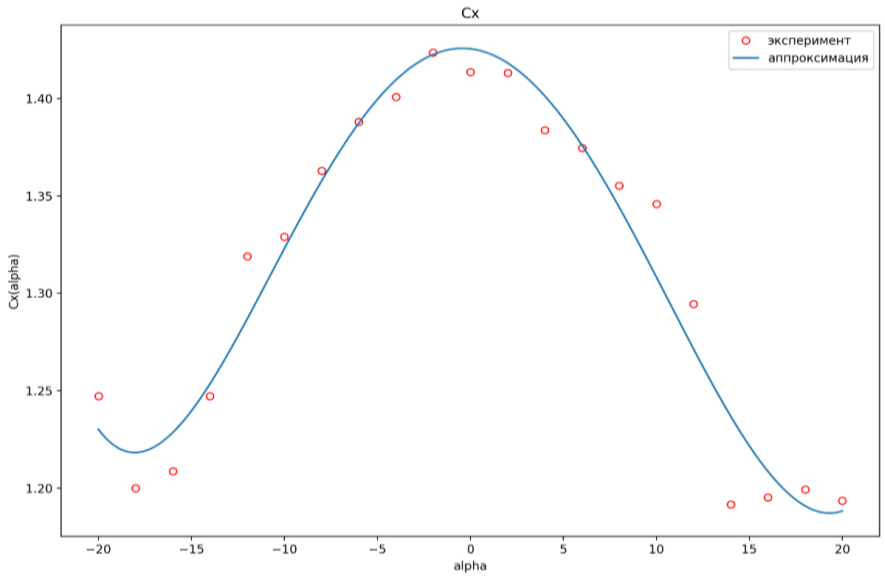

Рисунок 2.2.1. Коэффициент силы лобового сопротивления

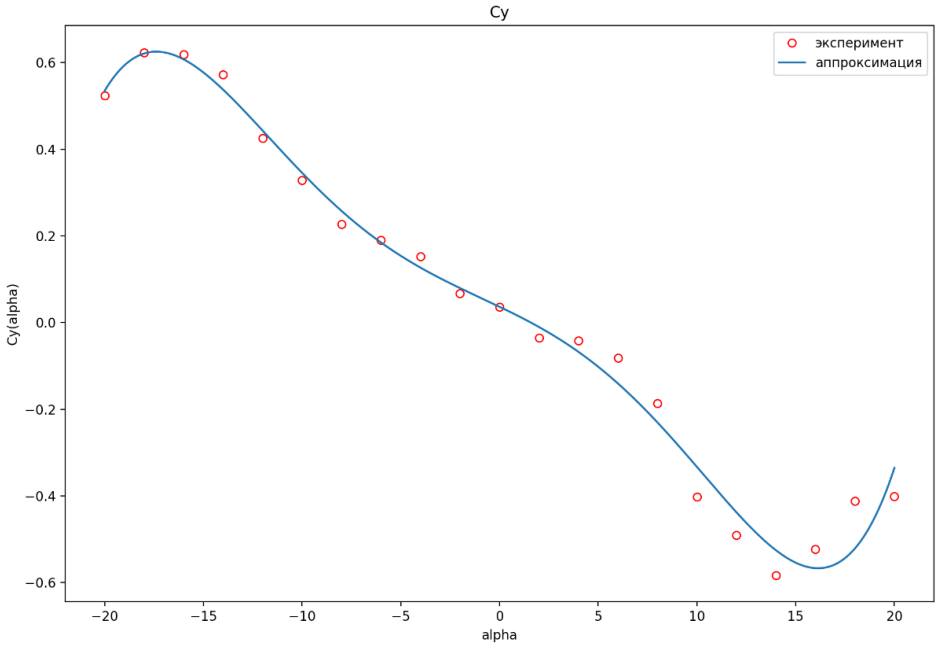

Рисунок 2.2.2. Коэффициент подъемной силы

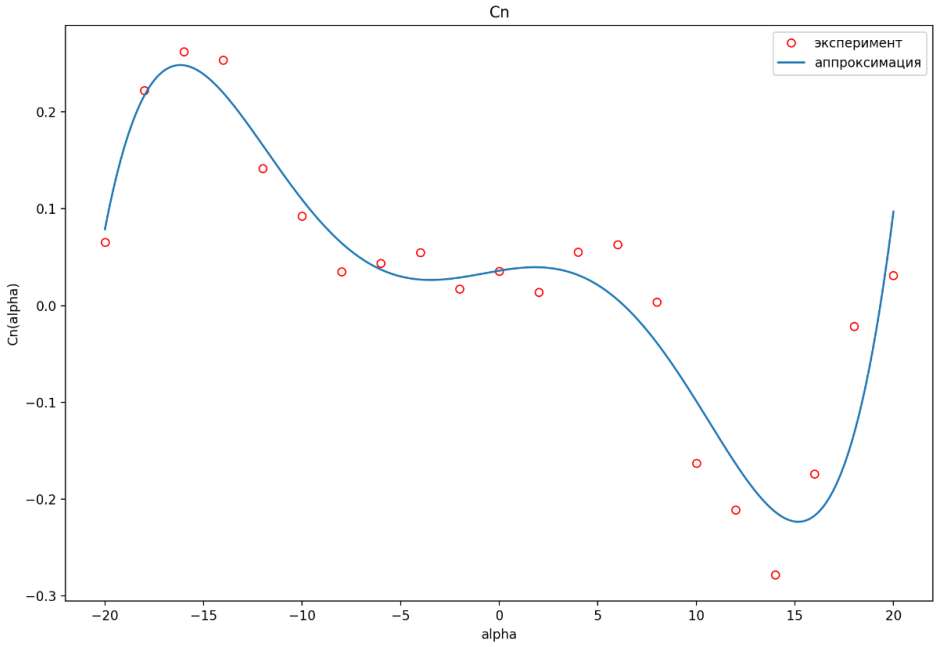

Рисунок 2.2.3. Коэффициент нормальной силы

Чтобы узнать, будет ли раскачиваться плохообтекаемое тело в потоке воздуха, нужно смотреть на производную коэффициента нормальной силы в нуле. По критерию Ден Гартога [8] колебания развиваются при некоторой критической скорости набегающего потока, если

$$\left[\frac{dC_n}{d\alpha}\cos\alpha+\frac{dC_t}{d\alpha}\sin\alpha\right]_{\alpha=\alpha_0}<0, \qquad (2.2.13)$$

где $C_n = C_x\cos\alpha + C_y\sin\alpha$, $C_t = C_x\cos\alpha - C_y\sin\alpha$ – коэффициент нормальной и тангенциальной силы соответственно. В нашем случае $\alpha_0 = 0$, поэтому (2.13) перепишется в виде

$$\left[\frac{dC_n}{d\alpha}\right]_{\alpha=0}<0. \qquad (2.2.14)$$

По рис. (2.2.3) видно, что производная будет больше нуля. А именно

$$\left[\frac{dC_n}{d\alpha}\right]_{\alpha=0}=0.19. \qquad (2.2.15)$$

В случае прямой призмы, колебания развиваться не будут.

Теперь посмотрим на зависимость амплитуды колебаний от скорости набегающего потока. Будем использовать уравнение движения призмы в поперечном по отношению к потоку направлении, описанное в [9]

$$m\ddot{y}b + r\dot{y}b + kyb = C_n\left(\frac{\dot{y}b}{v}\right)S\frac{\rho_0 v_r^2}{2} \qquad (2.2.16)$$

Коэффициент подъемной силы C_n представляется в виде полинома по $\left(\frac{\dot{y}b}{v}\right)$.

В работе [10, 4] показано, что в случае симметричной призмы в разложении присутствуют только нечетные степени. Четные степени в разложении не дают вклада в решение. Подставляя разложение для C_n в (2.2.16) получим

$$m\ddot{y} + r\dot{y} + ky = L\frac{\rho_0 v_r^2}{2}\left[A_1\left(\frac{\dot{y}b}{v}\right)+A_3\left(\frac{\dot{y}b}{v}\right)^3+A_5\left(\frac{\dot{y}b}{v}\right)^5+A_7\left(\frac{\dot{y}b}{v}\right)^7+\ldots\right] \qquad (2.2.17)$$

Уравнение (2.2.17) можно записать в безразмерном виде

$$\ddot{y} + 2\eta\dot{y} + y = n\nu^2\left[A_1\left(\frac{\dot{y}}{\nu}\right) + A_3\left(\frac{\dot{y}}{\nu}\right)^3 + A_5\left(\frac{\dot{y}}{\nu}\right)^5 + A_7\left(\frac{\dot{y}}{\nu}\right)^7 + \ldots\right] \quad (2.2.18)$$
$$n = \rho_0 Sb/(2m); \quad \nu = v/\left(b\sqrt{k/m}\right),$$

где ν – безразмерная скорость, η – коэффициент демпфирования колебаний конструкции. Критическая скорость будет равна $\nu_0 = 2\eta/(nA_1)$. Уравнение (2.2.18) решалось методом Ван-дер-Поля. Решение искалось в виде $y = \rho_y \cos(t-\varphi)$, предполагая, что амплитуда и фаза являются медленными функциями времени. Таким образом, получены уравнения установления Ван-дер-Поля

$$\dot{\rho}_y = nA_1\left[\frac{1}{2}\left(\nu - \frac{2\eta}{nA_1}\right)\rho_y + \frac{3}{8}\frac{A_3}{A_1\nu}\rho_y^3 + \frac{5}{16}\frac{A_5}{A_1\nu^3}\rho_y^5 + \frac{35}{128}\frac{A_7}{A_1\nu^5}\rho_y^7 + \ldots\right], \quad (2.2.19)$$
$$\rho_y\dot{\varphi} = 0,$$

Одним из стационарных решений (2.2.19) будет $\rho_y = 0$. Приравнивая к 0 правую часть первого уравнения и разделив на ρ_y, получим уравнение для нахождения стационарных решений не равных нулю. Оставим первые четыре члена. Тогда получим

$$1 - \frac{\nu_0}{\nu} = \frac{3}{4}\frac{A_3}{A_1}\left(\frac{\rho_y}{\nu}\right)^2 + \frac{5}{8}\frac{A_5}{A_1}\left(\frac{\rho_y}{\nu}\right)^4 + \frac{35}{64}\frac{A_7}{A_1}\left(\frac{\rho_y}{\nu}\right)^6 \quad (2.2.20)$$

Данное уравнение решалось параметрически. В итоге, получен следующий график для амплитуды

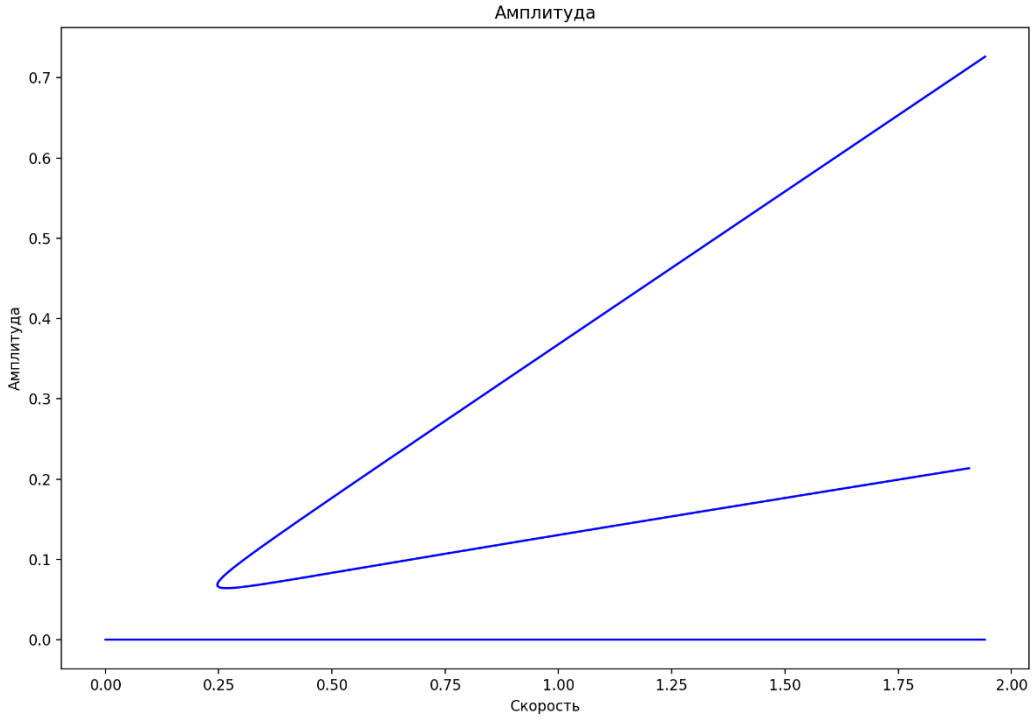

Рисунок 2.2.4. Зависимость амплитуды колебаний

от безразмерной скорости набегающего потока

По рис. (2.2.4) видно, что имеется 2 области: с одним стационарным решением и с тремя стационарными решениями. В данном случае нулевое стационарное решение является устойчивым в обеих областях. В области, где имеется 3 решения, устойчивыми будут нулевое решение и решение, которое соответствует самой верхней линии.

Физически это значит следующее: при безразмерной скорости до 0.25 как бы мы не раскачали призму, колебания будут затухающими и в итоге призма перестанет колебаться. При дальнейшем увеличении скорости набегающего потока, если призма в покое, то колебания не будут развиваться. Если теперь раскачать призму с амплитудой, значение которой лежит под средней прямой, то колебания будут также затухающими и призма перестанет колебаться. Но если раскачать призму до значения амплитуды, лежащего над средней прямой, колебания уже не будут затухать, а призма в итоге будет

колебаться с амплитудой, значение которой лежит на верхней прямой и соответствует данной скорости набегающего потока.

3. ПРОГРАММА ДЛЯ ОБРАБОТКИ ДАННЫХ

Когда все данные получены, для построения интересующих нас графиков требуется тщательная обработка данных. Обычно этот процесс занимает достаточно много времени, поэтому была написана программа на языке Python по обработке данных, которая делает это за несколько минут.

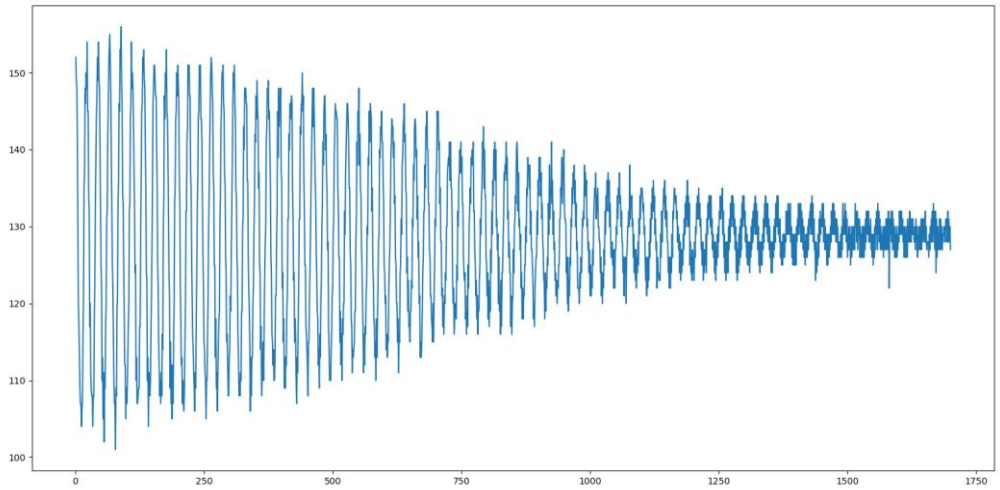

Рисунок 3.1. Пример записи сигнала

Первым делом, программа спрашивает, какие данные обрабатывать (самописца или осциллографа). Затем, анализируя имена файлов в директории, выбирает нужные для обработки файлы. С клавиатуры можно ввести количество данных (в процентах), которое мы хотим обрабатывать. Количество экспериментов, которое проводилось для каждого значения скорости набегающего потока, также вводится с клавиатуры.

Для получения зависимости логарифма амплитуды от времени требуется определить период колебаний. Обычно это делается вручную.

По данным на рис. 3.1 находится время, за которое совершается, например, 10 колебаний. Далее это время делят на количество колебаний и

получают период. Такая процедура выполняется для всех скоростей и для дальнейшей обработки требуется среднее значение периода по всем экспериментам. Как уже понятно, это занимает большое количество времени. Программа же делает это за секунды.

Сначала данные сглаживаются. Результат сглаживания показан на рисунке 3.2.

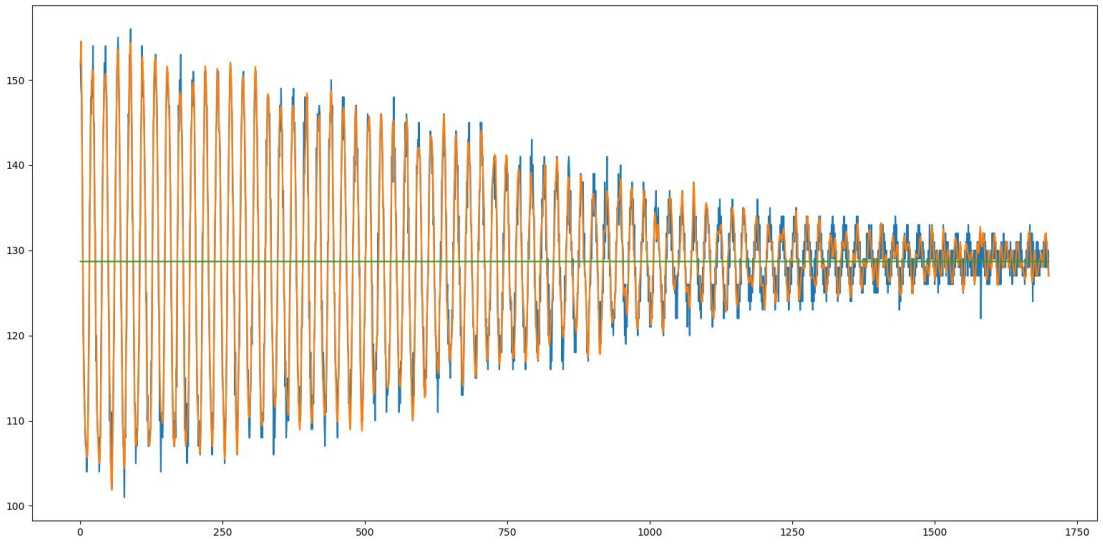

Рисунок 3.2. Пример интерполяции данных

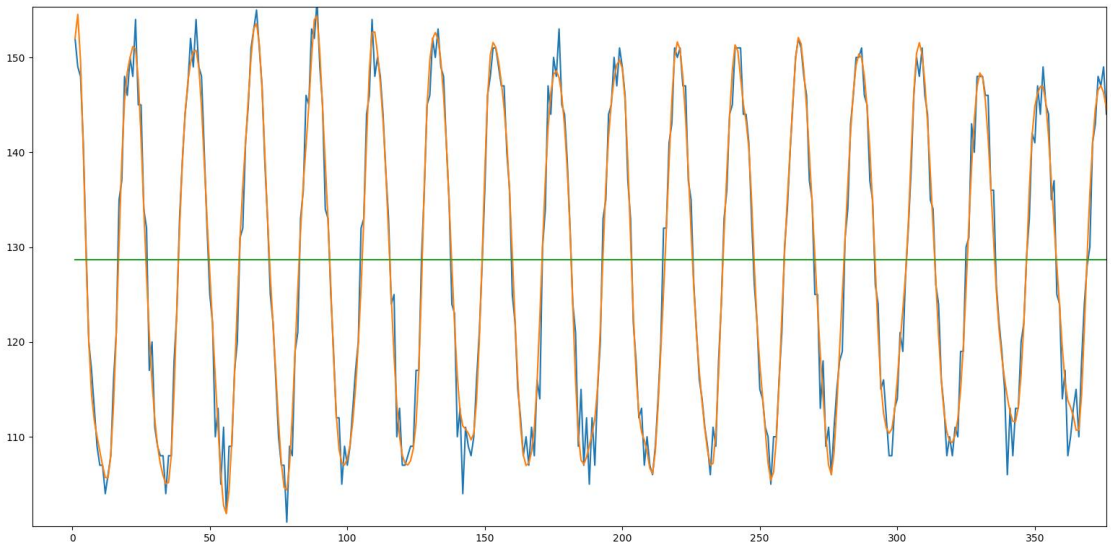

Рисунок 3.3. Пример интерполяции данных

(увеличенный масштаб)

Синим цветом отмечены данные эксперимента, оранжевым – интерполяция. Это делается для того, чтобы убрать осцилляции, которые мешают программно определять период. Зеленым обозначено среднее значение, которое соответствует нулевому отклонению призмы от состояния покоя. Далее считается количество пересечений графика интерполяции со средним значением и определяется период. Все это проделывается для всех данных и высчитывается среднее значение амплитуды.

Когда период вычислен, можно обрабатывать сигналы. После обработки данных сигналов по алгоритму, описанному в части 2.1, появляется текстовый файл в каталоге, к которому в дальнейшем программа неоднократно обращается.

После промежуточного формирования файлов с данными, которые представлены в более удобном виде, вычисляются коэффициенты модели (2.1.12). это уравнение переписывается в таком виде

$$\eta_i = -\frac{\mu}{2}k_2 + \frac{\mu}{2L}v_i - \frac{\mu}{2}\frac{\delta}{4L}v_i A_i^2, \qquad (3.1)$$

где i обозначает определенный участок времени, на котором мы хотим определить величины. Коэффициент затухания и амплитуда колебаний определяются исходя из графика зависимости логарифма амплитуды от времени.

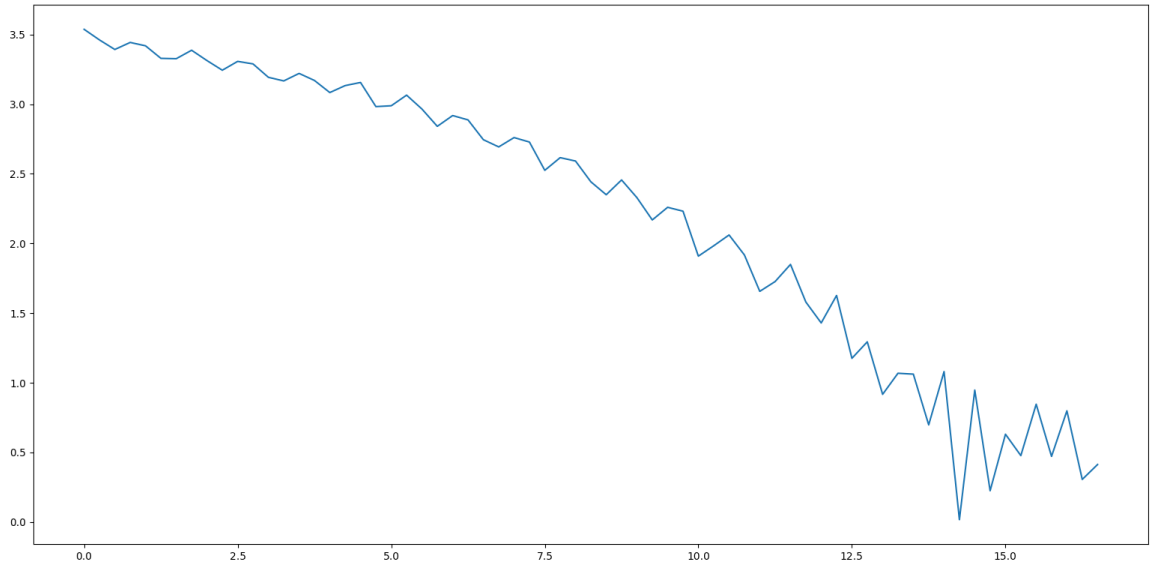

Рисунок 3.4. Пример графика зависимости

логарифма амплитуды от времени

На рис. 3.4 видно, что, начиная примерно с 12.5 секунды, начинаются погрешности (осцилляции). При подсчете коэффициентов модели эти погрешности учитывать не нужно. Т.е. начиная с 12.5 секунды нужно просто обрезать график. Данную процедуру нужно проделать для каждой скорости: построить график, найти момент времени, когда начинаются осцилляции, обрезать график. Написанная программа делает это автоматически, что значительно ускоряет процесс обработки результатов.

Далее, для определения η_i и A_i, весь, уже измененный, отрезок времени разбивается на несколько участков. На каждом участке зависимость логарифма амплитуды от времени приближается прямой методом наименьших квадратов. Программа позволяет регулировать количество точек на каждом участке. Это значение вводится с клавиатуры. В противном случае, количество точек выберется автоматически в интервале от 4 до 8. Для каждого случая будет вычислена ошибка аппроксимации и будет выбран вариант с наименьшей средней ошибкой по всем участкам времени.

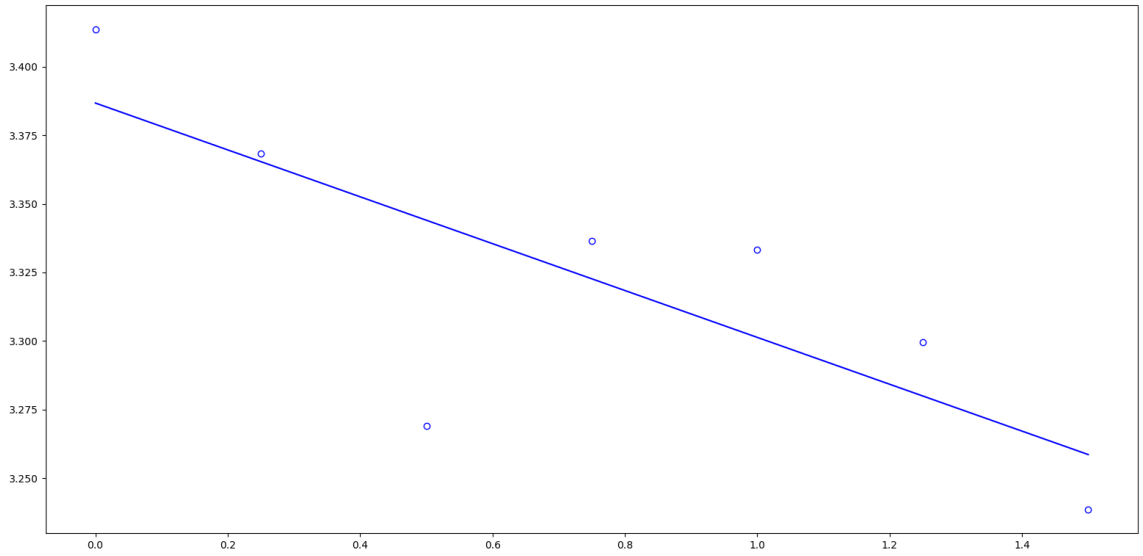

Рисунок 3.5. Пример аппроксимации зависимости

логарифма амплитуды от времени (7 точек)

На рис. 3.5 показан пример того, как аппроксимируется зависимость логарифма амплитуды от времени на участке с 7 точками. Амплитуда на данном участке будет равна e^{p_i}, где p_i – среднее значение логарифма амплитуды, а коэффициент затухания η_i будет равен тангенсу угла наклона этой прямой к оси Ox.

Таким образом мы имеем набор значений (η_i, ν_i, A_i). Для нахождения коэффициентов модели строится таблица такого вида $(\eta_i, \nu_i, \nu_i A_i^2)$. Далее коэффициенты модели ищутся методом наименьших квадратов. Таким образом, программа автоматически строит график, приведенный на рис. 2.1.3. При обработке данных также нужно учитывать градуировочный коэффициент. Для этого проводился отдельный эксперимент, данные которого также записаны в текстовый файл. Программа считывает этот файл и автоматически считает градуировочный коэффициент.

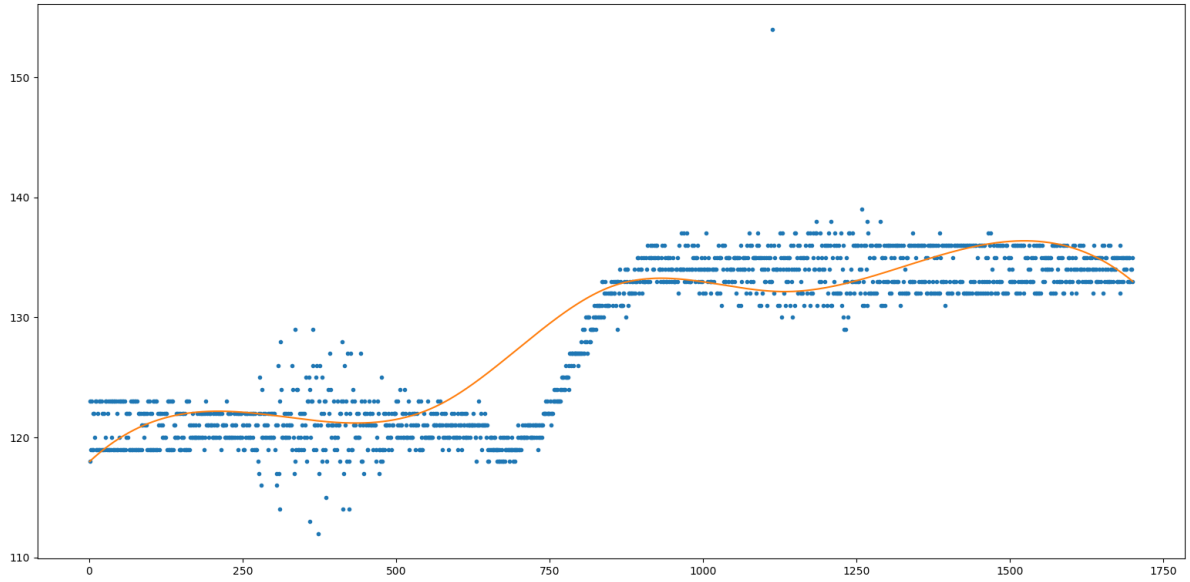

Рисунок 3.6. Градуировка (нагрузка)

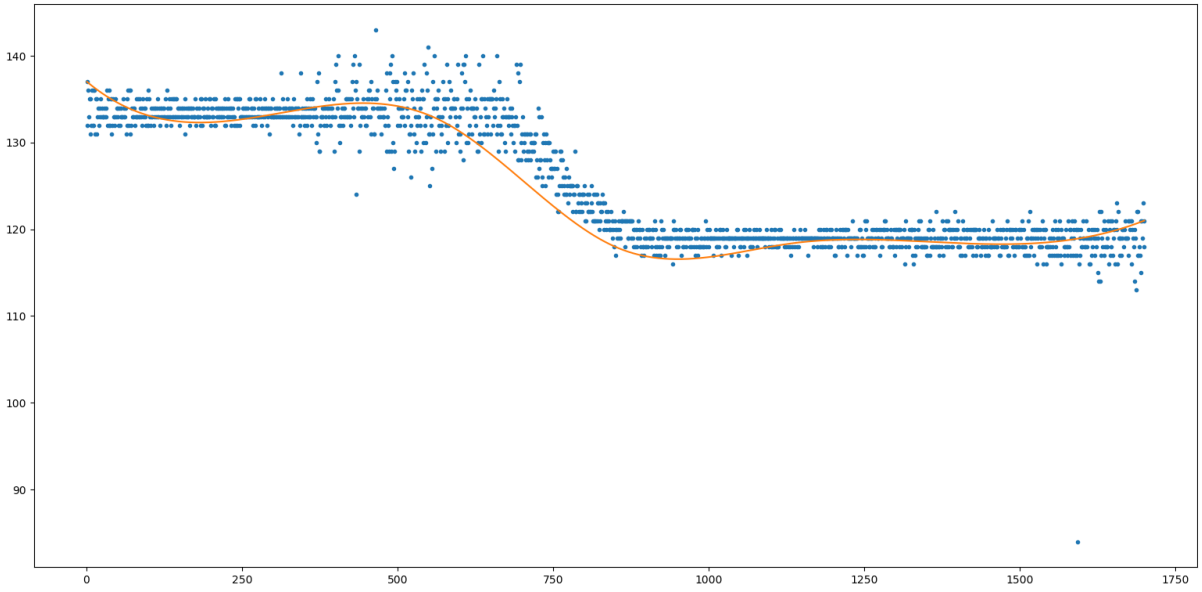

Рисунок 3.7. Градуировка (разгрузка)

На рис. 3.6 и 3.7 точками представлены данные эксперимента, линией – аппроксимация. Аппроксимация нужна для более точного определения высоты «ступеньки». После определения этой величины находится градуировочный коэффициент.

Для построения графиков для аэродинамических коэффициентов, данные экспериментов считываются и обрабатываются.

Программа позволяет вводить с клавиатуры длину и высоту модели, начальное и конечное значение угла атаки, а также шаг в градусах. Далее, с клавиатуры вводятся степени полинома, которым будут аппроксимированы данные эксперимента. По умолчанию степени полинома равны 0, 1, 2, 3, 4, 5. После этого строятся нужные графики, считается производная от коэффициента нормальной силы в нуле и все результаты помещаются в папку с именем «Results», которая автоматически создается, если ее нет в директории.

График для зависимости амплитуды от безразмерной скорости набегающего потока также строится автоматически. Для этого параметрически решается уравнение (2.2.20). Полученный график также помещается в папку со всеми результатами.

Таким образом, написанная программа выполняет такие функции:

- Спрашивает, какие файлы нужно обрабатывать;
- по имени определяет нужные файлы и обрабатывает их;
- позволяет задавать процент данных, который нужно обрабатывать;
- позволяет задать количество экспериментов, которое проводилось для каждого значения скорости;
- автоматически определяет период колебаний, который нужен для дальнейшей обработки;
- определяет нужные для построения графиков величины;
- отсекает ненужные для поиска коэффициентов модели данные (осцилляции);
- ищет коэффициенты модели методом наименьших квадратов;

- строит график для коэффициента затухания, учитывая, что при малых амплитудах данные эксперимента и модели расходятся, и не наносит соответствующие точки на график;
- считывает данные для определения аэродинамических коэффициентов;
- считает значения этих коэффициентов;
- строит их графики в зависимости от угла атаки;
- аппроксимирует данные полиномом степени, которую можно задать с клавиатуры;
- параметрически решает уравнение для зависимости амплитуды от скорости и строит график этой зависимости;
- сохраняет все построенные графики и помещает их в отдельную папку.

ЗАКЛЮЧЕНИЕ

В результате исследования, проведен ряд экспериментов по исследованию вращательных колебаний прямой призмы в потоке; проведен ряд экспериментов для определения аэродинамических коэффициентов, используя квазистационарное приближение; проверена модель зависимости коэффициента затухания от амплитуды при скоростях от 0 до 23.7 м/с.

Исходя из критерия Ден Гартога, сделан вывод об отсутствии поступательного галопирования прямой призмы с рассматриваемыми параметрами при малых углах атаки.

Написана программа на языке Python, которая полностью обрабатывает полученные данные, строит все нужные графики и помещает все результаты в отдельную папку. Программа успешно работает и на других данных других экспериментов, поэтому может быть использована для быстрой и качественной обработки данных дальнейших экспериментов.

СПИСОК ЛИТЕРАТУРЫ

1. *Люсин В. Д., Рябинин А. Н.* Исследование влияния удлинения призмы на ее аэродинамические характеристики и амплитуду колебаний при галопировании // Вестн. С.-Петерб. ун-та. Сер. 1. 2011. Вып. 2. С. 139–145.
2. *Рябинин А.Н., Киселев Н.А*. Влияние положения оси вращения цилиндра на его вращательные колебания в воздушном потоке // Вестник СПбГУ. Сер. 1, — 2016. — Т. 3(61), — Вып. 2. — С. 315-323.
3. *Белоцерковский С.М., Скрипач Б.К., Табачников В.Г.* Крыло в нестационарном потоке газа. М.: Наука, 1971. — 768 с.
4. *Parkinson G. V., Brooks N. P.* On the Aeroelastic Instability of Bluff Cylinders // J. Appl. Mech. 1961. Vol. 28. P. 252-258.
5. *Parkinson G. V., Smith J. D.* The square prism as an aerolastic non-linear oscillator // Quarterly J. Mech. Applied Math. 1964. Vol. XVII Pt. 2. P. 225-239.
6. *Novak M.* Aeroelastic galloping of prismatic bodies // J. Engineering Mech. Division ASCE. 1969. Vol. 95. P. 115-142.
7. *Ryabinin A. N., Kiselev N. A.* Rotational oscillation of a cylinder in air flow // J. of Engineering and Applied Sciences ARPN. 2017. Vol. 12. NO. 23. P. 6803-6808.
8. *Панарьина Е. С., Рябинин А. Н.* Галопирование пятиугольной призмы // Сб. статей / Под ред. *Мирошина Р. Н.* СПб.: Изд-во ВВМ. С. 11-17.
9. *Рябинин А. Н.* Некоторые задачи аэродинамики плохообтекаемых тел. – СПб.: Изд-во С.-Петербургского угиверситета, 1997, 144 с.
10. *Томпсон Дж. М. Т.* Неустойчивости и катастрофы в науке и технике. М., 1985. 254 с.

ПРИЛОЖЕНИЕ. КОД ПРОГРАММЫ.

```
# -*- coding: utf-8 -*-
import re
import os
import numpy as np
import time
from scipy import interpolate, linspace, polyfit
import matplotlib.pyplot as plt
import sklearn.linear_model as lm
import shutil
# from matplotlib import pylab

def get_files(name_end_file):
    cwd = os.getcwd()
    files = os.listdir(cwd)
    f = open(name_end_file,'w+')
    for elem in files:
        f.write(elem + '\n')
    f.close()
    print "OK"
def interpol(X, y):
    f = open('t_o.txt', 'r')
    t_o = f.read().strip()
    f.close()
    if t_o == 'o':
        omega = 1250
    elif t_o == 't':
        omega = 100
    tck_temp = interpolate.splrep(X, y, s=0)
    x_temp = np.linspace(X[0], X[-1], num=len(X) / omega * 40, endpoint=True)# 40 это магическое число. отвечает за
сглаживание
                                                    #  графика. смотреть при новых данных на графики!!!
    y_temp = interpolate.splev(x_temp, tck_temp, der=0)
    tck_new = interpolate.splrep(x_temp, y_temp, s=0)
    x_new = np.linspace(X[0], X[-1], num=len(X), endpoint=True)
    y_new = interpolate.splev(x_new, tck_new, der=0)
    return x_new, y_new
def find_rho_K_P_T():
    f = open('date.txt', 'r')
    data = f.readlines()
    f.close()
    for line in data:
        temp = re.split(r'\s+', line)
        if 'T' in temp:
            T = float(temp[1])
        if 'P' in temp:
            P = float(temp[1])
        if 'K' in temp:
            K = float(temp[1])
    rho = 0.125 * (1 + 0.0013 * (P - 760) - 0.0036 * (T - 15))
    return rho, K, P, T
def find_period(name):
    omega_o = 1250 # частота считывания в секунду у осцилографа
    omega_t = 100 # частота считывания в секунду у самописца
    num_T = 10  # количество колебаний, по которым хотим определить период (или просто кол-во периодов)
    f = open(name, 'r')
    data = f.readlines()
    f.close()
    f = open('t_o.txt', 'r')
    t_o = f.read().strip()
    f.close()
    X, y = [], []
```

```
    for line in data:
        temp = re.findall(r'\d+\t\d+\t\d+', line)
        if temp:
            temp = temp[0].split('\t')
            X.append(float(temp[0]))
            y.append(float(temp[1]))
    if t_o == 'o':
        omega = omega_o
    else:
        omega = omega_t
    num_all_seconds = float(X[-1]) / omega
    x_new, y_new = interpol(X, y)
    average = np.average(y_new)
    y_max = np.max(y_new)
    y_average = average * np.ones(len(x_new))
    N = len(X)
    count, seconds, m_seconds_for_percent = 0, 0, 0
    for i in range(N-1):
        if (count-1)/2 <= num_T:
            if y_new[i] <= average and y_new[i+1] >= average:
                count += 1
            elif y_new[i] >= average and y_new[i+1] <= average:
                count += 1
        else:
            break
        seconds += 1.0 / omega #* 10
    for i in range(N-1):
        if (max(y_new[i:])-average) / (y_max-average) > 0.5:
            m_seconds_for_percent += 1.0 / omega #* 10
        else:
            break
    percent = round(float(m_seconds_for_percent) / num_all_seconds, 0)
    period = round(float(seconds) / num_T, 2)
    return period, int(percent)
def make_new_file_list():
    f = open('files.txt', 'r')
    files = f.readlines()
    f.close()
    f = open('t_o.txt', 'r')
    t_o = f.read().strip()
    f.close()
    if t_o == 'o':
        omega = 1250
    elif t_o == 't':
        omega = 100
    new_file, number_files = "", 0
    for line in files:
        if '.txt' in line:
            temp = re.findall(r'[%s]{1}_\d{3}_\d{1,2}_\d{1,2}\.txt' % t_o, line)
            if temp:
                new_file += temp[0]
                new_file += '\n'
                number_files += 1
    new_file = new_file.rstrip()
    names = new_file.split('\n')
    period, percent = [], []
    for name in names:
        temp_period, temp_percent = find_period(name)
        period.append(temp_period)
        percent.append(temp_percent)
    period = round(np.average(period), 2)
    num_f_per_period = period * 100 #omega
    percent = int(np.min(percent) + 1)
    case = 1
```

```
    percent_2 = raw_input(u"Сколько процентов данных анализировать (1-100)?: ")
    if percent_2:
        percent = percent_2
    else:
        percent = 100
        print u"Выбрано по умолчанию: %s" % (str(percent))
    if t_o == 'o':
        number_reading = 4095
    elif t_o == 't':
        number_reading = 1700
    f = open('files.txt', 'w+')
    f.write("%s %s %s %s %s\n" % (number_files, num_f_per_period, case, percent, number_reading))
    f.write(new_file)
    f.close()
    f = open("average_period.txt", 'w+')
    f.write(str(num_f_per_period))
    f.close()
    print "OK"
def all_files_to_handle():
    f = open('is_all_files.txt', 'w')
    f.write("yes")
    f.close()
    try:
        f = open('t_o.txt', 'r')
    except:
        t_o_new = 'o'
        f = open('t_o.txt', 'w+')
        f.write(t_o_new)
        f.close()
    else:
        t_o = f.read().strip()
        f.close()
        if t_o == 'o':
            t_o_new = 't'
        else:
            t_o_new = 'o'
        f = open('t_o.txt', 'w')
        f.write(t_o_new)
        f.close()
    if t_o_new == 'o':
        res = u"********************ОСЦИЛЛОГРАФ********************".upper()
    elif t_o_new == 't':
        res = u"********************САМОПИСЕЦ********************".upper()
    print res
def what_files_to_explore():
    case = False
    while not case:
        try:
            f_1 = open('is_all_files.txt', 'r')
        except:
            case = raw_input(u"Какие данные изучать?\n1 - осциллограф\n2 - самописец\n3 - осциллограф и самописец\n0 -
Выход\n"
                             u"Ваш выбор: ")
            if not case:
                case = 0
                print u"Выбрано по умолчанию: %s" % (str(case))
            if int(case) == 3:
                all_files_to_handle()
                break
            elif int(case) == 1:
                t_o = 'o'
                res = u"********************ОСЦИЛЛОГРАФ********************".upper()
            elif int(case) == 2:
                t_o = 't'
```

```
            res = u"******************САМОПИСЕЦ******************".upper()
        elif int(case) == 0:
            exit()
        f = open('t_o.txt', 'w+')
        f.write(t_o)
        f.close()
        print res
      else:
        is_all_files = f_1.read().strip()
        if is_all_files == 'yes':
          all_files_to_handle()
          case = 3
          break
  print "OK"
def explore_data():
  os.system('treat0')
def data_for_plot_LnA_ot_t():
  f = open('result3_1.txt', 'r')
  data = f.readlines()
  f.close()
  time, lnA, TIME, LNA = [], [], [], []
  number = 0
  number_of_experiments = raw_input(u"Количество экспериментов для каждой скорости: ")
  if not number_of_experiments:
    number_of_experiments = 2
    print u"Выбрано по умолчанию: %s" % str(number_of_experiments)
  number_of_experiments = int(number_of_experiments)
  f = open('number_of_experiments.txt', 'w+')
  f.write(str(number_of_experiments))
  f.close()
  last_line = False
  for line in data:
    temp = re.findall(r'\d+\.\d+\s+', line)
    if temp:
      temp = re.split(r'\s+', line.strip())
      time.append(float(temp[-3]))
      lnA.append(float(temp[-2]))
      if line == data[-1]:
        last_line = True
    if not temp or last_line:
      if time and lnA:
        number += 1
        TIME.append(time)
        LNA.append(lnA)
      time, lnA = [], []
      if number == number_of_experiments:
        f = open(name[:-6]+'_for_plot(LnA(t)).txt', 'w+')
        for i in range(len(TIME[0])):
          f.write(str(round(np.average([TIME[0][i],TIME[1][i]]),5)))
          f.write('\t' + str(round(np.average([LNA[0][i],LNA[1][i]]),5)))
          if i<len(TIME[0])-1:
            f.write('\n')
        f.close()
        number = 0
        TIME, LNA = [], []
      if number == 0:
        name = line.strip()
  print "OK"

def make_result_average():
  f = open('result3.txt', 'r')
  data = f.readlines()
  f.close()
  data_for_all_results, new_data = "", ""
```

```
    col_1, col_2 = [], []
    count = 0
    f = open('number_of_experiments.txt', 'r')
    number_of_experiments = int(f.readlines()[0])
    f.close()
    for line in data:
        count += 1
        temp = line.strip()
        temp = re.split('\s+', temp)
        name = temp[0][:-6] + '_average_%s.txt' % count
        col_1.append(float(temp[1]))
        col_2.append(float(temp[2]))
        data_for_all_results += name + '\t%s\t%s\n' \
                                % (round(float(temp[1]), 5), round(float(temp[2]), 5))
        if len(col_1) == number_of_experiments:
            name = temp[0][:-6] + '_average.txt'
            new_data += name + '\t%s\t%s\n' \
                        % (round(np.average(col_1), 5), round(np.average(col_2), 5))
            col_1, col_2 = [], []
            count = 0
    f = open('result_average.txt', 'w+')
    f.write(new_data.strip())
    f.close()
    f = open('results_all_for_plot.txt', 'w+')
    f.write(data_for_all_results.strip())
    f.close()
    print "OK"
def data_for_plot_koef_ot_velocity():
    f = open('results_all_for_plot.txt', 'r')
    data = f.readlines()
    f.close()
    velocity, angle = [], []
    for line in data:
        temp = re.split(r'\s+', line.strip())
        name = temp[0]
        subnames = re.split(r'_', name)
        if subnames[1] not in velocity:
            velocity.append(subnames[1])
        if subnames[2] not in angle:
            angle.append(subnames[2])
    for ang in angle:
        res = ""
        X, y = [], []
        for vel in velocity:
            for line in data:
                if re.findall(r'_%s_%s_' % (vel, ang), line):
                    koef = re.split(r'\s+', line.strip())[1]
                    res += str(vel) + '\t' + str(koef) + '\n'
        name = 'plot_koef_ot_vel_to_%s.txt' % ang
        f = open(name, 'w+')
        f.write(res.strip())
        f.close()
    print "OK"
def make_plot_names():
    f = open('plot_names.txt', 'r')
    data = f.readlines()
    f.close()
    res = ""
    for line in data:
        temp = re.findall(r'plot_koef_ot_vel_to_\d+\.txt', line.strip())
        if temp:
            res += temp[0]
            res += '\n'
    f = open('plot_names.txt', 'w+')
```

```
    f.write(res.strip())
    f.close()
    print "OK"
def plot_koef_ot_vel():
    f = open('plot_names.txt')
    names = f.readlines()
    f.close()
    f = open('t_o.txt', 'r')
    t_o = f.read().strip()
    f.close()
    files = []
    for name in names:
        f = open(name.strip(), 'r')
        files.append(f.readlines())
        f.close()
    X, y = [], []
    XX, yy = [], []
    rho_air, K, _, _T = find_rho_K_P_T()
    mu = 0.992
    COEF_FOR_VEL = float(2 * mu * K) / rho_air
    count = 0
    for data in files:
        name = names[count].strip()[:-4]
        count += 1
        for elem in data:
            temp = re.split(r'\s+', elem)
            X.append(float(temp[0]))
            y.append(float(temp[1]))
        XX.append(np.sqrt(COEF_FOR_VEL * np.array(X)))
        yy.append(np.array(y))
        X, y = [], []
    angle = []
    for name in names:
        temp = name.split('_')
        angle.append(float(temp[-1][:-4]) / 2)
    max_len_elem = len(XX[0])
    for elem in XX:
        if len(elem) >= max_len_elem:
            max_len_elem = len(elem)
            ind = XX.index(elem)
            x = np.linspace(min(elem), max(elem), 300)
    colors = ['b', 'g', 'm', 'c', 'r', 'y', 'k', 'w']
    markers = ['o', '^', 's', 'D', 'd']
    lines = ['-', '--', '-.', ':']
    BB = {}
    plt.figure(2)
    for i in range(len(XX)):
        if len(XX[i]) < max_len_elem:
            X_new = [XX[ind][0]]
            y_new = [yy[ind][0]]
            X_new = X_new + list(XX[i])
            y_new = y_new + list(yy[i])
            X_new = np.array(X_new)
            y_new = np.array(y_new)
            A = np.vstack([X_new, np.ones(len(X_new))]).T
            B, c = np.linalg.lstsq(A, y_new)[0]
            plt.plot(X_new, y_new, markers[i], color=colors[i],
                     fillstyle='none', label='alpha = %s' % angle[i])
        else:
            plt.plot(XX[i], yy[i], markers[i], color=colors[i],
                     fillstyle='none', label='alpha = %s' % angle[i])
            A = np.vstack([XX[i], np.ones(len(XX[i]))]).T
            B, c = np.linalg.lstsq(A, yy[i])[0]
        plt.plot(x, B * x + c, lines[i], color=colors[i],
```

```
                label='alpha = %s (eta(V))' % angle[i])
            BB[angle[i]] = round(B, 6)
        plt.title(u'Коэффициент затухания')
        plt.xlabel('velocity [m/s]')
        plt.ylabel('eta')
        plt.legend(loc='best', prop={"size": 10})
        path = os.getcwd()
        if 'Results' not in os.listdir(path):
            os.mkdir('Results')
        plt.savefig('Results\%s_koef_ot_vel_plot.png' % t_o, dpi=200)
        # plt.show()
        plt.close()
        text = ''
        for name in BB.keys():
            text += '%s\t%s\n' % (name, BB[name])
        f = open('B.txt', 'w+')
        f.write(text.strip())
        f.close()
        print "OK"
    def make_count_1():
        f = open('count_1.txt', 'w+')
        f.write('yes')
        f.close()
    def make_plot_names_LnA_ot_t():
        f = open('t_o.txt', 'r')
        t_o = f.read().strip()
        f.close()
        f = open('plot_names_LnA_ot_t.txt', 'r')
        data = f.readlines()
        f.close()
        res = ""
        for line in data:
            temp = re.findall(r'[%s]{1}_\d{3}_\d{1,2}_for_plot\(LnA\(t\)\)\.txt' % t_o, line)
            if temp:
                res += temp[0]
                res += '\n'
        f = open('plot_names_LnA_ot_t.txt', 'w+')
        f.write(res.strip())
        f.close()
        print "OK"
    def find_eta_i(X, y):
        colors = 'b'
        markers = 'o'
        lines = '-'
        x = np.linspace(min(X), max(X), 100)
        A = np.vstack([X, np.ones(len(X))]).T
        (b, c), residuals, rank, ss = np.linalg.lstsq(A, y)
        if not residuals:
            residuals = [0.0,]
        eta = b
        return eta, residuals[0]
    def chenge_data(X, y, ERROR_COEFF):
        R = []
        ind_max = len(X)
        for i in range(int(len(X) // 1.5)):
            r = np.sqrt((X[i + 1] - X[i]) ** 2 + (y[i + 1] - y[i]) ** 2)
            R.append(r)
        r_mean_max = max(R)
        for i in range(int(len(X) // 1.5) + 1, len(X)-1):
            r = np.sqrt((X[i + 1] - X[i]) ** 2 + (y[i + 1] - y[i]) ** 2)
            if r >= r_mean_max*ERROR_COEFF:
                ind_max = i-1
                break
        XX = X[:ind_max]
```

```
    yy = y[:ind_max]
    return XX, yy
def make_data_for_func_eta():
    f = open('plot_names_LnA_ot_t.txt', 'r')
    names = f.readlines()
    f.close()
    files = []
    v = []
    for name in names:
        v_file = re.split(r'_',name)[1]
        while v_file.startswith('0') and len(v_file) > 1:
            v_file = v_file[1:]
        v_file = float(v_file)
        v.append(v_file)
        f = open(name.rstrip(), 'r')
        temp = f.readlines()
        files.append({'v': v_file, 'points': temp})
        f.close()
    eta, residuals = [], []
    max_all_residuals = []
    all_eta = []
    result_text_list = []
    res = []
    ERROR_COEFF = raw_input(u"'Коэффициент ошибки' (для отсечения осцилляций, 1-3): ")
    if not ERROR_COEFF:
        ERROR_COEFF = 2
        print u"Выбрано по умолчанию: %s" % str(ERROR_COEFF)
    ERROR_COEFF = float(ERROR_COEFF)
    num_a, num_b = 4, 8
    num_points_str = raw_input(
        u"По скольким точкам линейно аппроксимировать зависимость LnA(t) (по умолчанию число в пределах (%s;%s)?: "
        % (num_a, num_b))
    if num_points_str:
        num_points = int(num_points_str)
        num_a = num_points
        num_b = num_points + 1
    for j in range(num_a, num_b):
        num_points = j
        ss = ''
        for data in files:
            file_dict = {}
            X, y = [], []
            eta_file, residuals_file = [], []
            for elem in data['points']:
                temp = re.split(r'\s+', elem)
                X.append(float(temp[0]))
                y.append(float(temp[1]))
            # interpol_for_cut(X, y)
            X, y = chenge_data(X, y, ERROR_COEFF)
            for i in range(0, len(X), num_points):
                try:
                    XX = X[i:i+num_points]
                    yy = y[i:i+num_points]
                except:
                    XX = X[i:]
                    yy = y[i:]
                eta_i, residuals_i = find_eta_i(XX, yy)
                A_i = np.exp(np.mean(yy))
                ss += str(eta_i) + '\t' + str(data['v']) + '\t' + str(A_i) + '\n'
                eta_file.append(eta_i)
                residuals_file.append(residuals_i)
            eta.append(eta_file)
            residuals.append(max(residuals_file))
        all_eta.append(eta)
```

```
        max_all_residuals.append(max(residuals)/num_points)
        result_text_list.append(ss)
    min_max_residuals = min(max_all_residuals)
    ind_min_max_residuals = max_all_residuals.index(min_max_residuals)
    eta_result = all_eta[ind_min_max_residuals]
    text_result = result_text_list[ind_min_max_residuals]
    f = open('data_for_func_eta.txt', 'w+')
    f.write(text_result)
    f.close()
def grad_coeff():
    I = 300 # расстояние от оси вращения до конца концевой державки (мм)
    get_files('names_for_grad.txt')
    f = open('names_for_grad.txt', 'r')
    data = f.readlines()
    f.close()
    f = open('t_o.txt', 'r')
    t_o = f.read().strip()
    f.close()
    names = []
    for line in data:
        temp = re.findall(r'[%s]{1}_grad_\d+.txt' % t_o, line)
        if temp:
            names.append(line.strip())
    files = []
    for name in names:
        f = open(name, 'r')
        files.append(f.readlines())
        f.close()
    XX, yy = [], []
    H = []
    for data in files:
        X, y = [], []
        for line in data:
            temp = re.findall(r'\d{1,5}\s+\d{1,5}\s+\d{1,5}', line)
            if temp:
                temp2 = re.split(r'\s+', temp[0])
                X.append(float(temp2[0]))
                y.append(float(temp2[1]))
                # plt.plot(X, y)
        XX.append(X)
        yy.append(y)
        tck_temp = interpolate.splrep(X, y, s=0)
        x_temp = np.linspace(X[0], X[-1], num=7,
                             endpoint=True)
        y_temp = interpolate.splev(x_temp, tck_temp, der=0)

        tck_new = interpolate.splrep(x_temp, y_temp, s=0)
        x_new = np.linspace(X[0], X[-1], num=len(X), endpoint=True)
        y_new = interpolate.splev(x_new, tck_new, der=0)
        H.append(abs(np.max(y_new) - np.min(y_new)))
    H = np.mean(H)
    f = open('date.txt', 'r')
    data = f.readlines()
    f.close()
    for line in data:
        if 'h' in line:
            temp = re.split(r'\s+', line)
            if temp:
                h = float(temp[1])
    GRAD_COEFF = h / (I*H)
    return GRAD_COEFF
def find_coeff_model():
    f = open('data_for_func_eta.txt', 'r')
    data = f.readlines()
```

```
    f.close()
    mu = 0.992
    rho_air, K, _, _T = find_rho_K_P_T()
    COEF_FOR_VEL = float(2 * mu * K) / rho_air
    GRAD_COEFF = grad_coeff()
    eta, A, v = [], [], []
    x1, x2, y = [], [], []
    eta_temp, A_temp = [], []
    for line in data:
        temp = re.split(r'\s+', line)
        if temp:
            y.append(float(temp[0]))
            x1.append(np.around(np.sqrt(COEF_FOR_VEL * float(temp[1])), decimals=2))
            x2.append(np.around(np.sqrt(COEF_FOR_VEL * float(temp[1])), decimals=2)*(GRAD_COEFF*float(temp[2]))**2)
            eta_temp.append(float(temp[0]))
            A_temp.append(GRAD_COEFF*float(temp[2]))
            if len(x1) > 1:
                if x1[-1] != x1[-2] or line == data[-1]:
                    eta.append(eta_temp)
                    A.append(A_temp)
                    v.append(x1[-2])
                    eta_temp, A_temp = [], []
    X_temp = np.matrix([x1, x2])
    y_temp = np.matrix([y])
    X = X_temp.getT()
    y = y_temp.getT()
    skm = lm.LinearRegression()
    skm.fit(X, y)
    coeff = [skm.intercept_[0], skm.coef_[0][0], skm.coef_[0][1]]
    a = coeff[0]
    b = coeff[1]
    c = coeff[2]
    return eta, A, v, a, b, c,
def plot_eta_ot_A():
    f = open('t_o.txt', 'r')
    t_o = f.read().strip()
    f.close()
    f = open("average_period.txt", 'r')
    average_period = 0.01 * float(f.read().strip()) # перевод из сотых долей секунды в секунды
    f.close()
    eta, A, v, a, b, c = find_coeff_model()
    colors = ['b', 'g', 'r', 'c', 'm']
    markers = ['o', '^', 's', 'D', 'd']
    lines = ['-', '--', '-.', ':']
    A_max, A_min = A[0][0], A[0][0]
    for elem in A:
        for subelem in elem:
            if subelem > A_max:
                A_max = subelem
            if subelem < A_min:
                A_min = subelem
    Ax = np.linspace(A_min, A_max, 300)
    plt.figure(figsize=(12, 8))
    delta_all_plot = []
    for i in range(len(eta)):
        if i % 4 == 0:
            ii = i // 4
            is_label = False
            for j in range(len(eta[i])):
                if A[i][j] != A_min:  # обрезаем концы (слева)
                    if not is_label:  # для подписи графика
                        plt.plot(A[i][j], abs(eta[i][j]) * average_period, markers[ii], color=colors[ii],
                                 fillstyle='none',
                                 label='v = %s' % v[i])
```

```
                    is_label = True  # для определения заходило в if или нет. для подписи графика
                else:
                    plt.plot(A[i][j], abs(eta[i][j]) * average_period, markers[ii], color=colors[ii],
                             fillstyle='none')
            plt.plot(Ax, abs(a + b * v[i] + c * v[i] * Ax ** 2) * average_period, lines[0], color=colors[ii],
                     label='v = %s' % v[i])
            delta_plot = ""
            for k in range(len(Ax)):
                delta_plot += str(Ax[k]) + '\t' + str(abs(a + b * v[i] + c * v[i] * Ax[k] ** 2) * average_period) + '\n'
            delta_all_plot.append(delta_plot.strip())
    f = open('delta_plot_without_plates.txt', 'w+')
    f.write(delta_all_plot[-1])
    f.close()
    zero_level = []
    for elem in re.split(r'\n', delta_all_plot[0]):
        temp = re.split(r'\s', elem)
        zero_level.append(float(temp[1]))
    f = open('zero_level_wt_p.txt', 'w+')
    f.write(str(np.average(zero_level)))
    f.close()
    path = os.getcwd()
    try:
        path_M = re.sub(u'Борис', u'Михаил', path)
        shutil.move(r'delta_plot_without_plates.txt', r'D:\Учёба\СПб\НИП\Новая программа\Михаил\2-й курс\koleb\date')
    except:
        pass
    plt.title(u'Логарифмический декремент')
    plt.xlabel('A')
    plt.ylabel('delta(A) = |eta|*T [1/s^2]')
    plt.legend(loc='best', prop={"size": 10})
    path = os.getcwd()
    if 'Results' not in os.listdir(path):
        os.mkdir('Results')

    plt.savefig('Results\%s_koef_ot_A_plot.png' % t_o, dpi=200)
    plt.show()
    plt.close()
def load_data_XY():
    f = open("data_XY.txt", 'r')
    data = f.readlines()[1:]
    f.close()
    X1, Y1, Y2 = [], [], []
    for line in data:
        temp = re.split(r'\s+', line)
        if temp:
            X1.append(float(temp[0]) * 1e-3)
            Y1.append(float(temp[1]) * 1e-3)
            Y2.append(float(temp[2]) * 1e-3)
    return np.array(X1), np.array(Y1), np.array(Y2)
def load_data_tarirovka():
    f = open("data_tarirovka.txt", 'r')
    data = f.readlines()[1:]
    f.close()
    X, y = [], []
    for line in data:
        temp = re.split(r'\s+', line)
        if temp:
            X.append(float(temp[0]))
            y.append(float(temp[1]))
    return np.array(X), np.array(y)
def load_data_podveska():
    f = open("data_podveska.txt", 'r')
    data = f.readlines()[1:]
    f.close()
```

```
    hx, XX = [], []
    for line in data:
        temp = re.split(r'\s+', line)
        if temp:
            hx.append(float(temp[0]))
            XX.append(float(temp[1]) * 1e-3)
    return np.array(hx), np.array(XX)
def load_data_XY0():
    f = open("data_XY0.txt", 'r')
    data = f.readlines()[1:]
    f.close()
    X10, Y10, Y20 = [], [], []
    for line in data:
        temp = re.split(r'\s+', line)
        if temp:
            X10.append(float(temp[0]) * 1e-3)
            Y10.append(float(temp[1]) * 1e-3)
            Y20.append(float(temp[2]) * 1e-3)
    return np.array(X10), np.array(Y10), np.array(Y20)
def load_data_h():
    f = open("data_h.txt", 'r')
    data = f.readlines()[1:]
    f.close()
    h = []
    for line in data:
        temp = re.split(r'\s+', line)
        if temp:
            h.append(float(temp[0]))
    return np.array(h)
def find_data_for_amplitude(c_Cn_for_amplitude):
    c1 = c_Cn_for_amplitude[1]
    c3 = c_Cn_for_amplitude[3]
    c5 = c_Cn_for_amplitude[5]
    c7 = 0 #c_Cn_for_amplitude[7]
    v0 = -1
    A_v, v, A = [], [], []
    A_v = np.array(range(0, 1000, 1)) / 1000.0
    for elem in A_v:
        rhs = -(3 * c3 * np.power(elem, 2) / (4 * c1) + 5 * c5 * np.power(elem, 4) / (8 * c1) +
              35 * c7 * np.power(elem, 6) / (64 * c1))
        v_temp = v0 / (1 - rhs)
        A_temp = v_temp * elem
        if v_temp >= 0 and A_temp < 1 and v_temp <= 2:
            v.append(v_temp)
            A.append(A_temp)
    return v, A
def find_C_n():
    rho, K, P, T = find_rho_K_P_T() # находим плотность, коэфф манометра, давление и температуру
    mu = 0.992                  # задаем нужные для вычислений коэффициенты (этот относится к микроманометру)
    L, b = None, None
    while not L or not b:
        L_user = raw_input(u"Длина тела (горизонтальная, поперек потока) в м: ")
        b_user = raw_input(u"Высота (толщина) тела (вертикальная, поперек потока) в м: ")
        try:
            L = float(L_user)
            b = float(b_user)
        except:
            print u"Введите корректные размеры!\n"
        if not L_user and not b_user:
            L = 0.445
            b = 0.1
            print u"Выбраны размеры по умолчанию: L = %s, b = %s\n" % (L, b)
    S = L * b
    gamma = 0.812 * (1 - 0.0011 * (T - 15))
```

```
gamma0 = 0.8095
PP = gamma / gamma0
K = 0.4 # Почему не 0.2???
F = K / gamma0
alpha_0, alpha_end, delta_alpha = None, None, None
while not alpha_0 or not alpha_end or not delta_alpha:
    alpha_0 = raw_input(u"Начальный угол (в градусах): ")
    alpha_end = raw_input(u"Конечный угол (в градусах): ")
    delta_alpha = raw_input(u"Шаг (в градусах): ")
    try:
        alpha_0 = int(alpha_0)
        alpha_end = int(alpha_end)
        delta_alpha = int(delta_alpha)
    except:
        print u"Введите корректные углы!\n"
    if not alpha_0 and not alpha_end and not delta_alpha:
        alpha_0 = -20
        alpha_end = 20
        delta_alpha = 2
        print u"Выбраны углы по умолчанию: alpha_0 = %s, alpha_end = %s, delta_alpha = %s\n"\
            % (alpha_0, alpha_end, delta_alpha)
alpha = range(alpha_0, alpha_end + delta_alpha, delta_alpha)
alpha_1 = np.array(alpha) * np.pi / 180.0 # градусы в радианы
alpha_all = np.linspace(min(alpha), max(alpha), 300)
alpha_all_1 = np.linspace(min(alpha_1), max(alpha_1), 300)
# Тарировка
X, y = load_data_tarirovka()
# Подвеска
hx, XX = load_data_podveska()
# эксперимент
X1, Y1, Y2 = load_data_XY()
X10, Y10, Y20 = load_data_XY0()
h = load_data_h()
colors = 'r'
markers = 'o'
lines = '-'
x = np.linspace(min(X), max(X), 100)
A = np.vstack([X, np.ones(len(X))]).T
(a, b), residuals, rank, ss = np.linalg.lstsq(A, y)
plt.figure(figsize=(12, 8))
plt.plot(X, y, markers, color=colors, fillstyle='none', label=u'эксперимент')
plt.plot(x, a*x+b, '-', label=u'аппроксимация')
plt.title(u'Тарировка')
plt.xlabel(u'г')
plt.ylabel(u'г')
plt.legend(loc='best', prop={"size": 10})
path = os.getcwd()
if 'Results' not in os.listdir(path):
    os.mkdir('Results')
plt.savefig(u'Results\тарировка.png', dpi=200)
plt.close()
hxx = np.linspace(min(hx), max(hx), 100)
A_1 = np.vstack([hx, np.ones(len(hx))]).T
(a_1, b_1), residuals_1, rank_1, ss_1 = np.linalg.lstsq(A_1, XX)
plt.figure(figsize=(12, 8))
plt.plot(hx, XX, markers, color=colors, fillstyle='none', label=u'эксперимент')
plt.plot(hxx, a_1*hxx+b_1, '-', label=u'аппроксимация')
plt.title(u'Подвеска')
plt.xlabel('X')
plt.ylabel('h(X)')
plt.legend(loc='best', prop={"size": 10})
path = os.getcwd()
if 'Results' not in os.listdir(path):
    os.mkdir('Results')
```

```
    plt.savefig(u'Results\подвеска.png', dpi=200)
    plt.close()
    q = a_1 * h
    V_temp = np.power((2 * mu * h * K * PP / rho), 0.5)
    Cx_temp = (a * (X1 - X10) - q) / (rho * V_temp ** 2 / 2 * S)
    Cy_temp = 2.5 * (Y1 - Y10 + Y2 - Y20) / (rho * V_temp ** 2 / 2 * S)
    Cn_temp = Cy_temp * np.cos(alpha_1) + Cx_temp * np.sin(alpha_1)
    deg_str = raw_input(u"Степени полинома для аппроксимации (первую степень обязательно; через пробел): ")
    if not deg_str:
        deg = [0, 1, 2, 3, 4, 5]
        print u"Выбраны степени по умолчанию: %s" % (str(deg).strip('[]'))
    else:
        deg_temp_list = re.split(r'\s+', deg_str)
        deg = []
        for elem in deg_temp_list:
            deg.append(int(elem))
    deg_for_amplitude = [1, 3, 5]
    deg_for_amplitude_str = raw_input(u"Степени полинома для аппроксимации (для амплитуды; по умолчанию %s; через
пробел): "
                                      % str(deg_for_amplitude).strip('[]'))
    if not deg_for_amplitude_str:
        print u"Выбраны степени по умолчанию: %s" % (str(deg_for_amplitude).strip('[]'))
    else:
        deg_temp_list = re.split(r'\s+', deg_for_amplitude_str)
        deg_for_amplitude = []
        for elem in deg_temp_list:
            deg_for_amplitude.append(int(elem))
    c_V, stats_V = np.polynomial.polynomial.polyfit(alpha_1, V_temp, deg, full=True)
    c_Cx, stats_Cx = np.polynomial.polynomial.polyfit(alpha_1, Cx_temp, deg, full=True)
    c_Cy, stats_Cy = np.polynomial.polynomial.polyfit(alpha_1, Cy_temp, deg, full=True)
    c_Cn, stats_Cn = np.polynomial.polynomial.polyfit(alpha_1, Cn_temp, deg, full=True)
    c_Cn_for_amplitude, stats_Cn_for_amplitude = np.polynomial.polynomial.polyfit(alpha_1, Cn_temp, deg_for_amplitude,
full=True)
    print u"Коэффициенты в аппроксимации Cn для графика амплитуды: %s" % str(c_Cn_for_amplitude)
    v_for_amplitude, A_for_amplitude = find_data_for_amplitude(c_Cn_for_amplitude)
    try:
        v0_for_amplitude = np.linspace(0, max(v_for_amplitude), 300)
    except:
        v0_for_amplitude = np.linspace(0, 2, 300)
    A0_for_amplitude = np.zeros(300)
    print u"Производная коэффициента нормальной силы в 0:\nd(Cn)/d(alpha)(0) = %s" % str(c_Cn[1])
    V_, Cx_, Cy_, Cn_ = [], [], [], []
    for i in deg:
        V_.append(c_V[i] * np.power(alpha_all_1, i))
        Cx_.append(c_Cx[i] * np.power(alpha_all_1, i))
        Cy_.append(c_Cy[i] * np.power(alpha_all_1, i))
        Cn_.append(c_Cn[i] * np.power(alpha_all_1, i))
    V = np.sum(V_, axis=0)
    Cx = np.sum(Cx_, axis=0)
    Cy = np.sum(Cy_, axis=0)
    Cn = np.sum(Cn_, axis=0)
    plt.figure(figsize=(12, 8))
    plt.plot(v_for_amplitude, A_for_amplitude, 'b-')
    plt.plot(v0_for_amplitude, A0_for_amplitude, 'b-')
    plt.title(u'Амплитуда')
    plt.xlabel(u'Скорость')
    plt.ylabel(u'Амплитуда')
    path = os.getcwd()
    if 'Results' not in os.listdir(path):
        os.mkdir('Results')
    plt.savefig(u'Results\амплитуда.png', dpi=200)
    plt.close()
    plt.figure(figsize=(12, 8))
    plt.plot(alpha, V_temp, markers, color=colors, fillstyle='none', label=u'эксперимент')
```

```
    plt.title(u'Скорость')
    plt.xlabel('alpha')
    plt.ylabel('V(alpha)')
    plt.legend(loc='best', prop={"size": 10})
    path = os.getcwd()
    if 'Results' not in os.listdir(path):
        os.mkdir('Results')
    plt.savefig(u'Results\скорость.png', dpi=200)
    plt.close()
    plt.figure(figsize=(12, 8))
    plt.plot(alpha, Cx_temp, markers, color=colors, fillstyle='none', label=u'эксперимент')
    plt.plot(alpha_all, Cx, '-', label=u'аппроксимация')
    plt.title('Cx')
    plt.xlabel('alpha')
    plt.ylabel('Cx(alpha)')
    plt.legend(loc='best', prop={"size": 10})
    path = os.getcwd()
    if 'Results' not in os.listdir(path):
        os.mkdir('Results')
    plt.savefig(u'Results\Cx.png', dpi=200)
    plt.close()
    plt.figure(figsize=(12, 8))
    plt.plot(alpha, Cy_temp, markers, color=colors, fillstyle='none', label=u'эксперимент')
    plt.plot(alpha_all, Cy, '-', label=u'аппроксимация')
    plt.title('Cy')
    plt.xlabel('alpha')
    plt.ylabel('Cy(alpha)')
    plt.legend(loc='best', prop={"size": 10})
    path = os.getcwd()
    if 'Results' not in os.listdir(path):
        os.mkdir('Results')
    plt.savefig(u'Results\Cy.png', dpi=200)
    plt.close()
    plt.figure(figsize=(12, 8))
    plt.plot(alpha, Cn_temp, markers, color=colors, fillstyle='none', label=u'эксперимент')
    plt.plot(alpha_all, Cn, '-', label=u'аппроксимация')
    plt.title('Cn')
    plt.xlabel('alpha')
    plt.ylabel('Cn(alpha)')
    plt.legend(loc='best', prop={"size": 10})
    path = os.getcwd()
    if 'Results' not in os.listdir(path):
        os.mkdir('Results')
    plt.savefig(u'Results\Cn.png', dpi=200)
    plt.close()
def main():
    get_files('files.txt')
    os.system('clear')
    what_files_to_explore()
    make_new_file_list()
    explore_data()
    data_for_plot_LnA_ot_t()
    make_result_average()
    data_for_plot_koef_ot_velocity()
    get_files('plot_names.txt')
    make_plot_names()
    plot_koef_ot_vel()
    f = open('t_o.txt', 'r')
    t_o = f.read().strip()
    f.close()
    if t_o == 't':
        get_files('plot_names_LnA_ot_t.txt')
        make_plot_names_LnA_ot_t()
        make_data_for_func_eta()
```

```
            plot_eta_ot_A()
        is_remove = False
        if os.path.exists('is_all_files.txt'):
            if not os.path.exists('count_1.txt'):
                make_count_1()
                os.system('clear')
                main()
            else:
                os.remove('count_1.txt')
                os.remove('is_all_files.txt')
                os.remove('t_o.txt')
                os.remove('average_period.txt')
                os.remove('data_for_func_eta.txt')
                os.remove('names_for_grad.txt')
                os.remove('number_of_experiments.txt')
                os.remove('plot_koef_ot_vel_to_0.txt')
                os.remove('plot_names.txt')
                os.remove('plot_names_LnA_ot_t.txt')
                os.remove('result_average.txt')
                os.remove('results_all_for_plot.txt')
                os.system('clear')
                is_remove = True
        else:
            os.remove('average_period.txt')
            os.remove('data_for_func_eta.txt')
            os.remove('names_for_grad.txt')
            os.remove('number_of_experiments.txt')
            os.remove('plot_koef_ot_vel_to_0.txt')
            os.remove('plot_names.txt')
            os.remove('plot_names_LnA_ot_t.txt')
            os.remove('result_average.txt')
            os.remove('results_all_for_plot.txt')
        os.system('clear')
        if os.path.exists('t_o.txt'):
            os.remove('t_o.txt')
        os.system('clear')
    import sys
    reload(sys)
    sys.setdefaultencoding('cp866')
    res = raw_input(u"Вычислить Cn?\n1) Да\n2) Нет\nОтвет: ")
    if not res:
        res = 2
        print u"Выбрано по умолчанию: Нет"
    res = int(res)
    if res == 1:
        find_C_n()
    elif res == 2:
        pass
    os.system('pause')
    main()
```