**МОСКОВСКИЙ АВИАЦИОННЫЙ ИНСТИТУТ**

**(Национальный Исследовательский Университет)**

**Факультет №3** «Системы управления,

информатика и электроэнергетика»

**Кафедра №308** «Информационные технологии»





Курсовая работа

по курсу

« Методы и алгоритмы обработки информации »

**Выполнил:**  *Громов Павел Дмитриевич*

**Группа:** *03-417*

Москва 2013 год

Оглавление

1. Постановка задачи 3

2. МНК-алгоритм 4

3. Практическая часть 7

3.1. Исследование для первого воздействия 7

3.1.1. Построение имитатора объекта и ИИС 7

3.1.2. Проверка условий наблюдаемости и идентифицируемости объекта 11

3.1.3. Вычисление матрицы измерений *С* 12

3.1.4. Реализация МНК-алгоритма 14

3.1.5. Построение эллипса рассеяния 15

3.2. Исследование для второго воздействия 17

3.2.1. Построение имитатора объекта и ИИС 17

3.2.2. Проверка условий наблюдаемости и идентифицируемости объекта 21

3.2.3. Вычисление матрицы измерений *С* 23

3.2.4. Реализация МНК-алгоритма 25

3.2.5. Построение эллипса рассеяния 26

# 1. Постановка задачи

**Дано:**

- математическая модель ;

- модель измерений **, где η∈N[0, 0.01] - ошибки измерений;

- начальные условия *y(t=0)=y(0)=1;*

;

t∈[0; T], где Т =10с ;

- среднеквадратическое отклонение *σ2=0,01*

- входное воздействие

*;*

*x(t) = sin t;*

- номинальные (априорные) значения параметров

*θ1=0.3, θ2=0.2, θ3=10.*

**Найти:**

оценки параметров θ1 и θ2 математической модели объекта.

**Построить:**

эллипсы рассеяния.

**Проверить:**

выполнение условий линейности, наблюдаемости, идентифицируемости и устойчивости.

**Метод решения:**

алгоритм МНК.

# 2. МНК-алгоритм

Рассмотрим задачу идентификации параметров линейных динамических систем с помощью метода наименьших квадратов (МНК). Математическая модель линейной динамической системы описывается дифференциальным уравнением вида:

,

,

(1)

где

, , .

Известно, что она эквивалентна модели

,

(2)

которая не является линейной по параметрам. В процессе идентификации параметра  проводятся измерения координат вектора состояния или их линейных комбинаций.

Модель измерений, нелинейная по параметрам:

,

,

(3)

здесь . Модель (3) принято обозначать следующей тройкой:



(4)

Предполагается, что  удовлетворяет следующим ограничениям:

1.  должна быть дифференцируема по θ;

2. ,

здесь - обычно называемая матрицей функций чувствительности, элементы которой удовлетворяют соответствующим уравнениям чувствительности.

Будем предполагать, что в некоторой достаточно малой окрестности истинного значения  выбрана точка  (априорное значение вектора неизвестных параметров). В окрестности  проводим разложение  в ряд:

.

(5)

Обозначим вектора фиктивного измерения, поправки априорного значения неизвестных параметров и матрицы измерений , ,  соответственно.

С учетом введенных обозначений модель измерений принимает следующий вид:

.

(6)

Таким способом может быть осуществлено сведение нелинейной по параметру  модели (3) к модели (6), линейной по параметру . Она будет обозначаться тройкой

.

В этом случае, МНК-оценка вектора неизвестных параметров  определяется соотношением:



или



и, следовательно,

.

(7)

Точностные характеристики  задаются ковариационной матрицей



Если точность полученной оценки неудовлетворительна, то процесс повторяется, при этом в качестве новой точки разложения  принимается оценка, полученная на предыдущем шаге:

.

Такая процедура называется итерационной процедурой определения . После k повторений будем иметь:

 .

(8)

Рассмотрим подробнее построение алгоритма МНК для системы второго порядка:



(9)

Обозначим

, , ,

тогда

.

(10)

Линеаризуем это уравнение в окрестности a10, а20, b0:

.

(11)

Обозначим

, , .

Тогда модель измерений (12) принимает вид:

,

(12)

к этой модели применяется итерационная процедура (8) определения МНК-оценок параметров динамической системы (9).

В процессе оценки параметров кроме задания вектора измерений  возникает необходимость в вычислении вектора  (N+1×1) и матрицы **. Вектор  представляет собой дискретные отсчеты с шагом Δt решения уравнения (9) на интервале [0,NΔt] при a1=a10, а2=а20,b=b0. Матрица С представляет собой дискретные отсчеты с шагом Δt на интервале [0,NΔt] решения уравнений чувствительности.

# 3. Практическая часть

## 3.1. Исследование для первого воздействия

### 3.1.1. Построение имитатора объекта и ИИС

Практическая часть написана на языке Python версии 2.7.

Подключаем необходимые библиотеки.

*from scipy.integrate import odeint*

*from pylab import \**

*import numpy as np*

*import math*

*import random*

Установим объем выборки, предъявляемой к измерению:

*N = 1000*

Зададим исходные данные θ1 = 0.3, θ2 = 0.2, θ3 = 10:

*teta = [0.3, 0.2, 10.0]*

Опишем входное воздействие:

*def x(t):*

*if t >= 0:*

*return 1*

*else:*

*return 0*

Смоделируем эксперимент, решив методом Рунге-Кутта систему дифференциальных уравнений , где Y[0] = y(t), Y[1] = y'(t):

*def deriv(Y, t):*

*return array([ Y[1], - teta[1] \* Y[0] - teta[0] \* Y[1] + teta[2] \* x(t)])*

Зададим вектор начальных условий:

*yinit = array([1, 1])*

Зададим временные границы эксперимента:

*tn = 0*

*tk = 100*

*time = linspace(tn, tk, N)*

Решаем получившееся ДУ:

*y = odeint(deriv, yinit, time)*

*y1 = y[:, 0]*

*y2 = y[:, 1]*

Рисуем график зависимости результатов моделирования от времени: у1 - выход модели, у2 - его производная:

*fig = figure()*

*subplot(111)*

*plot(time, y1, label="y1")*

*plot(time, y2, label="y2")*

*legend(bbox\_to\_anchor=(0., 1.02, 1., .102), loc=3,*

*ncol=2, mode="expand", borderaxespad=0.)*

*xlabel('t')*

*savefig("1.1.png")*

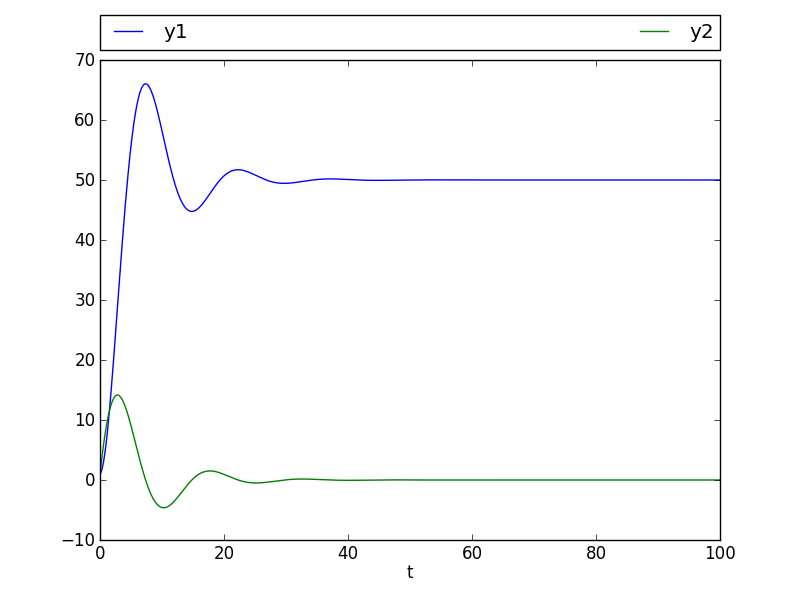
**

Рис. 1. График зависимости результатов моделирования от времени: у1 - выход модели, у2 - его производная

Построим имитатора объекта:

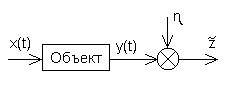


Рис. 2. Схема имитатора объекта

Предположим, что ошибки распределены по нормальному закону:

*n = np.random.normal(0, math.sqrt(0.01), N)*

Вектор измерений:

*Z = [y1[i] + n[i] for i in range(N)]*

Построим график имитатора объекта при наличии случайных возмущений на интервале 0..100 с delta = 0.1 с

*fig.clear()*

*plot(time, Z)*

*xlabel('t')*

*ylabel('Z')*

*savefig("1.2.png")*

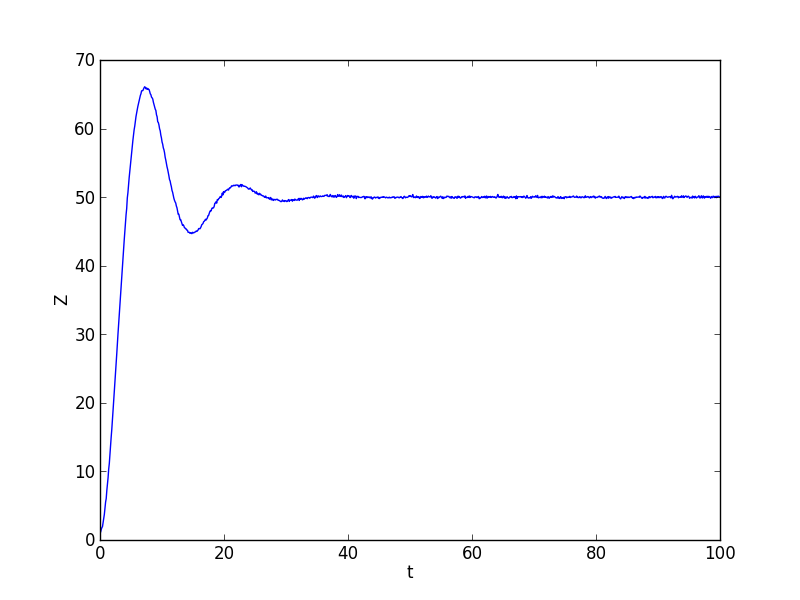
**

Рис. 3. график имитатора объекта при наличии случайных возмущений на интервале 0..100 с delta = 0.1 с

Построим график имитатора объекта при наличии случайных возмущений на интервале 0..2 с delta = 0.1 с

*fig.clear()*

*time = linspace(0.0, 2.0, N / 50)*

*subplot(111)*

*plot(time, y1[:N / 50], label="y1")*

*plot(time, Z[:N / 50], label="Z")*

*legend(bbox\_to\_anchor=(0., 1.02, 1., .102), loc=3,*

*ncol=2, mode="expand", borderaxespad=0.)*

*xlabel('t')*

*savefig("1.3.png")*

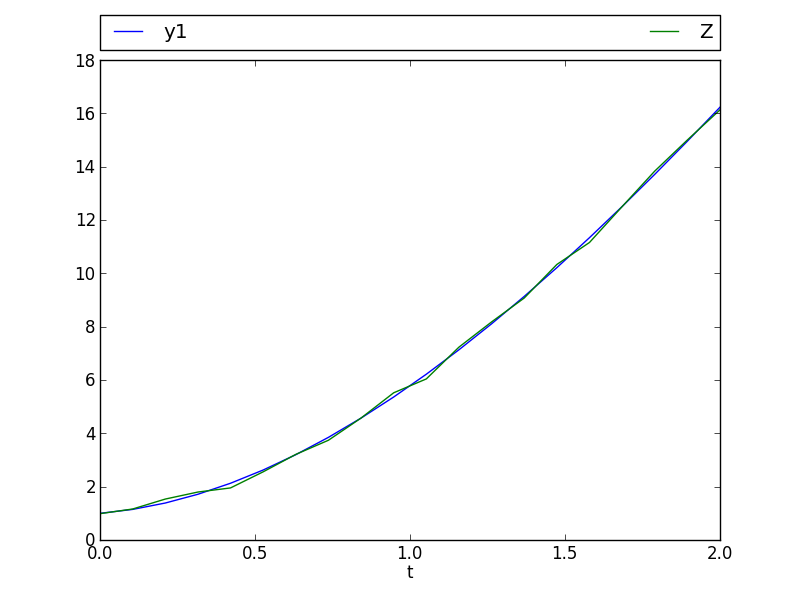
**

Рис. 4. График имитатора объекта при наличии случайных возмущений на интервале 0..2 с delta = 0.1 с

Этап построение имитатора объекта заканчивается на получении вектора измерений.

|  |  |  |
| --- | --- | --- |
| y1 | n | Z |
| 1.0 | -0.0342584650678 | 0.965741534932 |
| 1.14718130794 | -0.0222210525871 | 1.12496025535 |
| 1.38643833927 | 0.0530292502796 | 1.43946758955 |
| 1.71457487074 | 0.0159401899545 | 1.7305150607 |
| 2.12831382029 | -0.0671290286558 | 2.06118479163 |
| … | … | … |
| 50.0000113794 | -0.0766240517088 | 49.9233873277 |
| 50.0000107007 | 0.0551627118108 | 50.0551734125 |
| 50.0000100209 | -0.178441094674 | 49.8215689262 |
| 50.0000093412 | -0.107150996345 | 49.8928583449 |
| 50.000008663 | -0.00442332863421 | 49.9955853344 |

### 3.1.2. Проверка условий наблюдаемости и идентифицируемости объекта

**Проверим условие наблюдаемости:**

Выпишем матрицу А и матрицу измерений С:

Матрица А:

*A = np.array([[0.0, 1.0], [-teta[1], -teta[0]]])*

Матрица измерений C:

*C = np.array([1, 0])[np.newaxis]*

Сформируем матрицу наблюдаемости Калмана S = (CT AT\* CT):

*S = np.column\_stack((C.T, A.T.dot(C.T)))*

Если ее ранг равен 2, то условие наблюдаемости выполняется

*rank = np.linalg.matrix\_rank(S)*

Ранг равен 2, отсюда следует, что объект является наблюдаемым.

**Проверим условие идентифицируемости:**

Выпишем матрицу А и вектор начальных условий:

Матрица А:

Вектор начальных условий:

*y0 = np.array([1, 1])*

Сформируем матрицу наблюдаемости Калмана S = (y(0) A\* y(0)):

*S = np.column\_stack((y0.T, A.T.dot(y0.T)))*

Если ее ранг равен 2, то условие интифицируемости выполняется.

*rank = np.linalg.matrix\_rank(S)*

Ранг равен 2, отсюда следует, что объект является идентифицируемым.

### 3.1.3. Вычисление матрицы измерений *С*

Составим и решим методом Рунге-Кутта систему уравнений чувствительности:

Зададим матрицу частных производных:

*def deriv2(Y, t):*

*return array([ Y[1],*

*- teta[0] \* Y[1] - teta[1] \* Y[0] + teta[2] \* x(t),*

*Y[4],*

*Y[5],*

*- teta[1] \* Y[2] - teta[0] \* Y[4] - Y[1],*

*- teta[1] \* Y[3] - teta[0] \* Y[5] - Y[0]])*

Зададим вектор начальных условий:

*U0 = array([1, 1, 0, 0, 0, 0])*

Зададим временные границы эксперимента:

*time = linspace(tn, tk, N)*

Решение системы ДУ проводится с помощью функции odeint() посредством которой, выводится матрица U, столбцы которой содержат значения решений и производные.

*U = odeint(deriv2, U0, time)*

Присвоим интересующим нас параметрам значения матрицы U:

*y1a = U[:, 0]*

*C1 = np.array([U[:, 2]])*

*C2 = np.array([U[:, 3]])*

*Получим матрицу измерений из параметров матрицы U:*

*C = np.column\_stack((C1.T, C2.T))*

|  |  |
| --- | --- |
| C1 | C2 |
| 0.00000000e+00 | 0.00000000e+00 |
| -6.52274748e-03 | -5.16456501e-03 |
| -3.19388160e-02 | -2.15686895e-02 |
| … | … |
| -2.51462111e-04 | -2.50001575e+02 |
| -2.13695096e-04 | -2.50001598e+02 |
| - 1.76555994e-04 | -2.50001618e+02 |

Построим график чувствительности

*fig.clear()*

*time = linspace(0.0, 100.0, N)*

*plot(time, C[:, 0][:])*

*xlabel('t')*

*ylabel('C1')*

*savefig("1.4.png")*

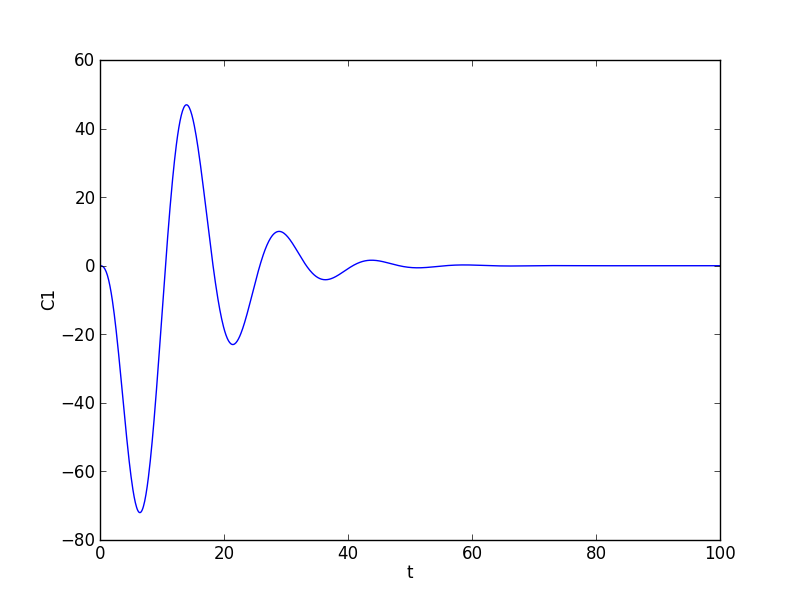
**

Рис. 5. График чувствительности

Построим график чувствительности

*fig.clear()*

*time = linspace(0.0, 100.0, N)*

*plot(time, C[:, 1][:])*

*xlabel('t')*

*ylabel('C2')*

*savefig("1.5.png")*

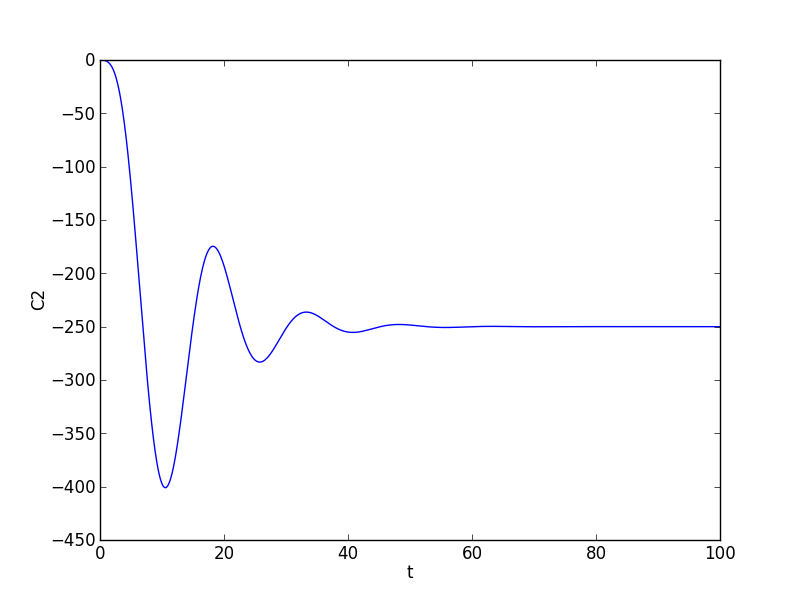
**

Рис. 6. График чувствительности

### 3.1.4. Реализация МНК-алгоритма

Определим требуемые оценки параметров:

Q = CT \* C:

*Q = np.matrix(C.T.dot(C))*

A = Q-1:

*A = Q.I*

b = CT \* Z:

*b = C.T.dot(Z)*

teta\_mnk = A \* b:

*teta\_mnk = A.dot(b)*

θ1 = , θ2 =

### 3.1.5. Построение эллипса рассеяния

det(lambda\*E - A) = 0

(lamda - 3.11617423e-06)\*(lambda - 1.60457589e-08) - 1.57025125e-08 \* 1.57025125e-08 = lambda^2 - 1.60457589e-08\*lambda - 3.11617423e-06\*lambda + 5.0001380384973146e-14 - 2.4656889881265625e-16 = lambda^2 - 3.1322199889e-06\*lambda + 4.975481148616049e-14

Подсчет и вывод корней уравнения:

*def equation\_solve(a,b,c):*

*D=b\*\*2-4\*a\*c*

*if a:*

*if D>0:*

*x1=(-b+D\*\*0.5)/(2\*a)*

*x2=(-b-D\*\*0.5)/(2\*a)*

*print "Корни уравнения:\n","x1 =",x1,"\nx2 =",x2*

*return [x1, x2]*

*if D==0:*

*x1=(-b)/(2\*a)*

*print "Корень уравнения:\n","x1 = x2 =",x1*

*if D<0:*

*print "Корни уравнения:"*

*print 'x1 = '-b/2\*a+math.sqrt(D)/2\*abs(a)*

*print 'x2 = '-b/2\*a-math.sqrt(D)/2\*abs(a)*

*elif b:*

*x1=-c/b*

*print'Корень уравнения:\n','x =',x1*

*elif c:*

*print'Уравнение неверно'*

*else:*

*print'Уравнение верно'*

Находим длины полуосей:

*lambda1, lambda2 = equation\_solve(1, - 3.1322199889e-06, 4.975481148616049e-14)*

*a1 = math.sqrt(lambda1)*

*a2 = math.sqrt(lambda2)*

a1 = 0.00176529141023

a2 = 0.000126357531956

Находим направления полуосей:

Для lambda1 = 3.11625376302e-06:

f1 = 1

f2 = - 197.43387714907547

Для lambda2 = *1.59662258821e-08*:

f1 = 1

f2 = *0.005064986759284622*

Строим эллипс рассеяния:

*npts = 250*

*theta = np.arange(npts)\*2.0\*math.pi/(npts-1)*

*angle = np.arctan(- 197.43387714907547)*

*x = 0 + 0.00176529141023\*np.cos(theta)\*np.cos(angle) - 0.000126357531956\*np.sin(theta)\*np.sin(angle)*

*y = 0 + 0.00176529141023\*np.cos(theta)\*np.sin(angle) - 0.000126357531956\*np.sin(theta)\*np.cos(angle)*

*clf()*

*plot(x,y,color="r")*

*axis([-0.002, 0.002, -0.002, 0.002])*

*grid(True)*

*savefig("1.ellipse.png")*

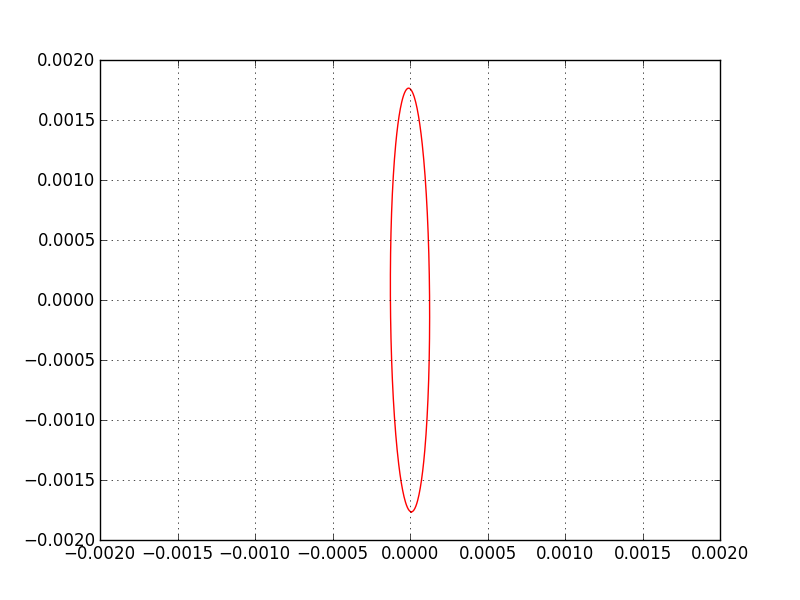
**

Рис. 7. Расположение эллипса рассеяния

## 3.2. Исследование для второго воздействия

### 3.2.1. Построение имитатора объекта и ИИС

Подключаем необходимые библиотеки.

*from scipy.integrate import odeint*

*from pylab import \**

*import numpy as np*

*import math*

*import random*

Установим объем выборки, предъявляемой к измерению:

*N = 1000*

Зададим исходные данные θ1 = 0.3, θ2 = 0.2, θ3 = 10:

*teta = [0.3, 0.2, 10.0]*

Опишем входное воздействие:

*def x(t):*

*return math.sin(t)*

Смоделируем эксперимент, решив методом Рунге-Кутта систему дифференциальных уравнений , где Y[0] = y(t), Y[1] = y'(t):

*def deriv(Y, t):*

*return array([ Y[1], - teta[1] \* Y[0] - teta[0] \* Y[1] + teta[2] \* x(t)])*

Зададим вектор начальных условий:

*yinit = array([1, 1])*

Зададим временные границы эксперимента:

*tn = 0*

*tk = 100*

*time = linspace(tn, tk, N)*

Решаем получившееся ДУ:

*y = odeint(deriv, yinit, time)*

*y1 = y[:, 0]*

*y2 = y[:, 1]*

Рисуем график зависимости результатов моделирования от времени: у1 - выход модели, у2 - его производная:

*fig = figure()*

*subplot(111)*

*plot(time, y1, label="y1")*

*plot(time, y2, label="y2")*

*legend(bbox\_to\_anchor=(0., 1.02, 1., .102), loc=3,*

*ncol=2, mode="expand", borderaxespad=0.)*

*xlabel('t')*

*savefig("2.1.png")*

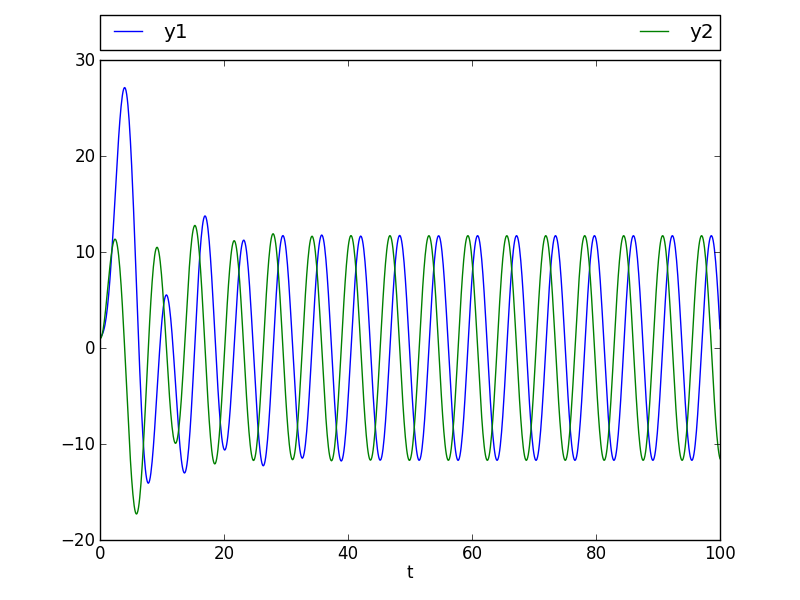
**

Рис. 8. График зависимости результатов моделирования от времени: у1 - выход модели, у2 - его производная

Построим имитатора объекта:

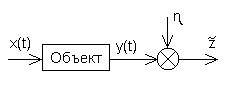


Рис. 9. Схема имитатора объекта

Предположим, что ошибки распределены по нормальному закону:

*n = np.random.normal(0, math.sqrt(0.01), N)*

Вектор измерений:

*Z = [y1[i] + n[i] for i in range(N)]*

Построим график имитатора объекта при наличии случайных возмущений на интервале 0..100 с delta = 0.1 с

*fig.clear()*

*plot(time, Z)*

*xlabel('t')*

*ylabel('Z')*

*savefig("2.2.png")*

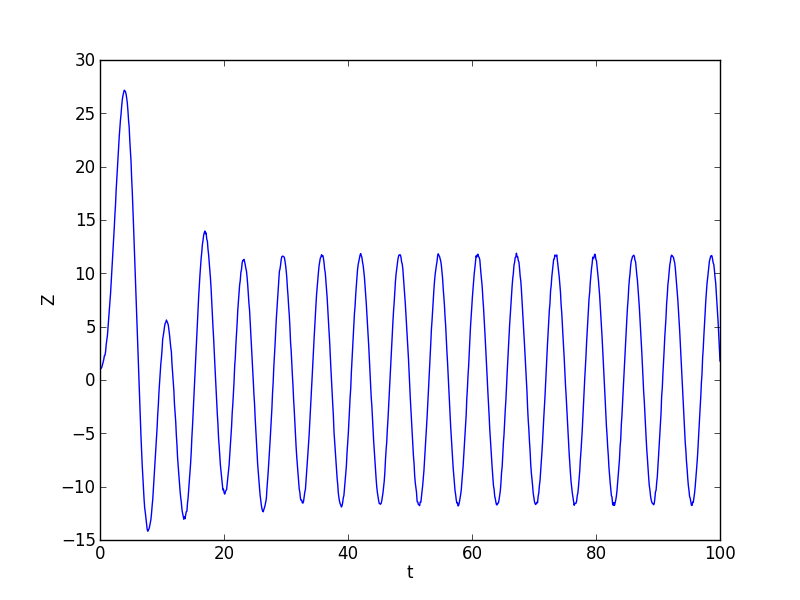
**

Рис. 10. график имитатора объекта при наличии случайных возмущений на интервале 0..100 с delta = 0.1 с

Построим график имитатора объекта при наличии случайных возмущений на интервале 0..2 с delta = 0.1 с

*fig.clear()*

*time = linspace(0.0, 2.0, N / 50)*

*subplot(111)*

*plot(time, y1[:N / 50], label="y1")*

*plot(time, Z[:N / 50], label="Z")*

*legend(bbox\_to\_anchor=(0., 1.02, 1., .102), loc=3,*

*ncol=2, mode="expand", borderaxespad=0.)*

*xlabel('t')*

*savefig("2.3.png")*

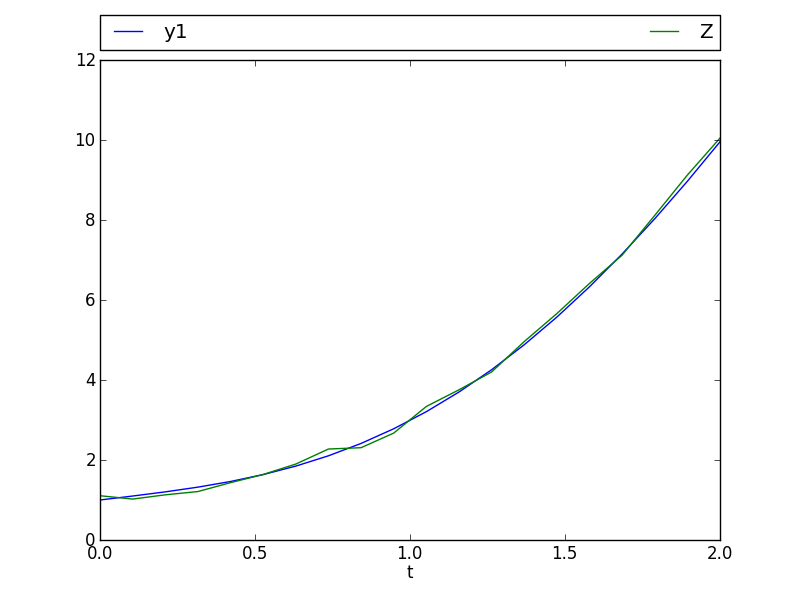
**

Рис. 11. График имитатора объекта при наличии случайных возмущений на интервале 0..2 с delta = 0.1 с

Этап построение имитатора объекта заканчивается на получении вектора измерений.

|  |  |  |
| --- | --- | --- |
| y1 | n | Z |
| 1.0 | 0.0354379859781 | 1.03543798598 |
| 1.09924544506 | 0.00488465098748 | 1.10413009605 |
| 1.20326420482 | 0.0106819138099 | 1.21394611863 |
| 1.32146533732 | -0.0738305632663 | 1.24763477405 |
| 1.46275491169 | 0.128857206006 | 1.5916121177 |
| … | … | … |
| 6.34145178438 | 0.0124722285569 | 6.35392401294 |
| 5.32663799221 | -0.175077537279 | 5.15156045493 |
| 4.25849560608 | -0.125201872874 | 4.13329373321 |
| 3.14771850719 | 0.0484748217445 | 3.19619332893 |
| 2.00542742437 | -0.0440096380355 | 1.96141778633 |

### 3.2.2. Проверка условий наблюдаемости и идентифицируемости объекта

**Проверим условие наблюдаемости:**

Выпишем матрицу А и матрицу измерений С:

Матрица А:

*A = np.array([[0.0, 1.0], [-teta[1], -teta[0]]])*

Матрица измерений C:

*C = np.array([1, 0])[np.newaxis]*

Сформируем матрицу наблюдаемости Калмана S = (CT AT\* CT):

*S = np.column\_stack((C.T, A.T.dot(C.T)))*

Если ее ранг равен 2, то условие наблюдаемости выполняется

*rank = np.linalg.matrix\_rank(S)*

Ранг равен 2, отсюда следует, что объект является наблюдаемым.

**Проверим условие идентифицируемости:**

Выпишем матрицу А и вектор начальных условий:

Матрица А:

Вектор начальных условий:

*y0 = np.array([1, 1])*

Сформируем матрицу наблюдаемости Калмана S = (y(0) A\* y(0)):

*S = np.column\_stack((y0.T, A.T.dot(y0.T)))*

Если ее ранг равен 2, то условие интифицируемости выполняется.

*rank = np.linalg.matrix\_rank(S)*

Ранг равен 2, отсюда следует, что объект является идентифицируемым.

### 3.2.3. Вычисление матрицы измерений *С*

Составим и решим методом Рунге-Кутта систему уравнений чувствительности:

Зададим матрицу частных производных:

*def deriv2(Y, t):*

*return array([ Y[1],*

*- teta[0] \* Y[1] - teta[1] \* Y[0] + teta[2] \* x(t),*

*Y[4],*

*Y[5],*

*- teta[1] \* Y[2] - teta[0] \* Y[4] - Y[1],*

*- teta[1] \* Y[3] - teta[0] \* Y[5] - Y[0]])*

Зададим вектор начальных условий:

*U0 = array([1, 1, 0, 0, 0, 0])*

Зададим временные границы эксперимента:

*time = linspace(tn, tk, N)*

Решение системы ДУ проводится с помощью функции odeint() посредством которой, выводится матрица U, столбцы которой содержат значения решений и производные.

*U = odeint(deriv2, U0, time)*

Присвоим интересующим нас параметрам значения матрицы U:

*y1a = U[:, 0]*

*C1 = np.array([U[:, 2]])*

*C2 = np.array([U[:, 3]])*

*Получим матрицу измерений из параметров матрицы U:*

*C = np.column\_stack((C1.T, C2.T))*

|  |  |
| --- | --- |
| C1 | C2 |
| 0.00000000e+00 | 0.00000000e+00 |
| -4.91757808e-03 | -5.12406282e-03 |
| 1.96225255e-02 | -2.09417714e-02 |
| … | … |
| -1.36968540e+01 | 1.86412354e-01 |
| -1.36469473e+01 | -1.18329013e+00 |
| -1.34604077e+01 | -2.54114858e+00 |

Построим график чувствительности

*fig.clear()*

*time = linspace(0.0, 100.0, N)*

*plot(time, C[:, 0][:])*

*xlabel('t')*

*ylabel('C1')*

*savefig("2.4.png")*

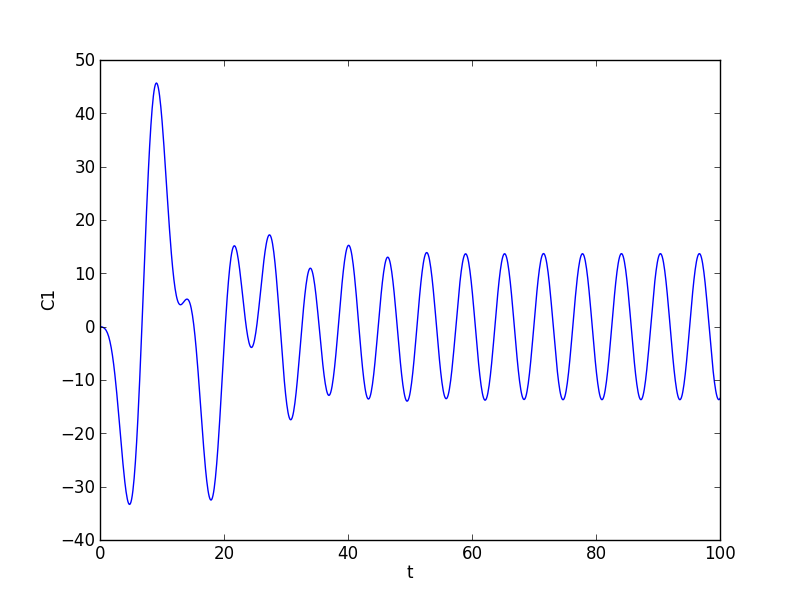
**

Рис. 12. График чувствительности

Построим график чувствительности

*fig.clear()*

*time = linspace(0.0, 100.0, N)*

*plot(time, C[:, 1][:])*

*xlabel('t')*

*ylabel('C2')*

*savefig("1.5.png")*

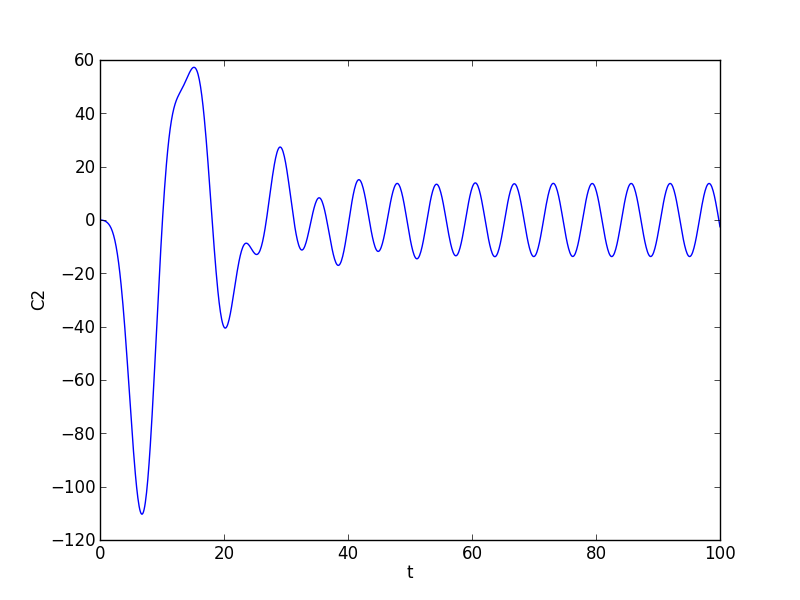
**

Рис. 13. График чувствительности

### 3.2.4. Реализация МНК-алгоритма

Определим требуемые оценки параметров:

Q = CT \* C:

*Q = np.matrix(C.T.dot(C))*

A = Q-1:

*A = Q.I*

b = CT \* Z:

*b = C.T.dot(Z)*

teta\_mnk = A \* b:

*teta\_mnk = A.dot(b)*

θ1 = , θ2 =

### 3.2.5. Построение эллипса рассеяния

det(lambda\*E - A) = 0

(lamda - 5.52396113e-06)\*(lambda - 1.49058867e-06) - 1.91489016e-08 \* 1.91489016e-08 = lambda^2 - 1.49058867e-06\*lambda - 5.52396113e-06\*lambda + 8.233953873898396e-12 - 3.6668043248648257e-16 = lambda^2 - 7.014549800000001e-06\*lambda + 8.23358719346591e-12

Подсчет и вывод корней уравнения:

*def equation\_solve(a,b,c):*

*D=b\*\*2-4\*a\*c*

*if a:*

*if D>0:*

*x1=(-b+D\*\*0.5)/(2\*a)*

*x2=(-b-D\*\*0.5)/(2\*a)*

*print "Корни уравнения:\n","x1 =",x1,"\nx2 =",x2*

*return [x1, x2]*

*if D==0:*

*x1=(-b)/(2\*a)*

*print "Корень уравнения:\n","x1 = x2 =",x1*

*if D<0:*

*print "Корни уравнения:"*

*print 'x1 = '-b/2\*a+math.sqrt(D)/2\*abs(a)*

*print 'x2 = '-b/2\*a-math.sqrt(D)/2\*abs(a)*

*elif b:*

*x1=-c/b*

*print'Корень уравнения:\n','x =',x1*

*elif c:*

*print'Уравнение неверно'*

*else:*

*print'Уравнение верно'*

Находим длины полуосей:

*lambda1, lambda2 = equation\_solve(1, - 7.014549800000001e-06, 8.23358719346591e-12)*

*a1 = math.sqrt(lambda1)*

*a2 = math.sqrt(lambda2)*

a1 = 0.00176529141023

a2 = 0.000126357531956

Находим направления полуосей:

Для lambda1 = 5.52405203957e-06:

f1 = 1

f2 = 210.63680754406928

Для lambda2 = *1.49049776043e-06*:

f1 = 1

f2 = - 0.004747508492197222

Строим эллипс рассеяния:

npts = 250

theta = np.arange(npts)\*2.0\*math.pi/(npts-1)

angle = np.arctan(210.63680754406928)

x = 0 + 0.00235033019799\*np.cos(theta)\*np.cos(angle) - 0.00122085943516\*np.sin(theta)\*np.sin(angle)

y = 0 + 0.00235033019799\*np.cos(theta)\*np.sin(angle) - 0.00122085943516\*np.sin(theta)\*np.cos(angle)

clf()

plot(x,y,color="r")

axis([-0.003, 0.003, -0.003, 0.003])

grid(True)

savefig("2.ellipse.png")

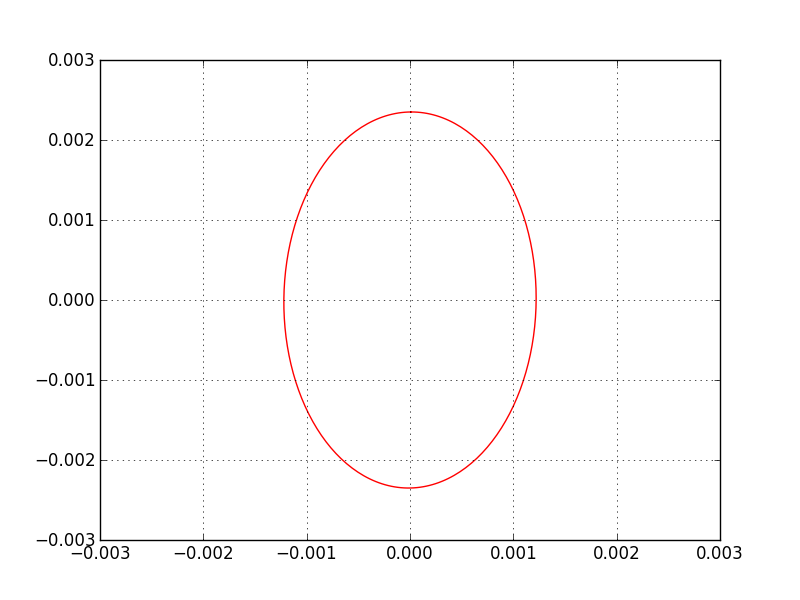
**

Рис. 14. Расположение эллипса рассеяния