§2 Второй отборочный этап

Второй отборочный этап проводится в командном формате в сети интернет. Продолжительность второго отборочного этапа — 4 недели. Работы оцениваются автоматически средствами stepik. Задачи носят междисциплинарный характер и в упрощенной форме воссоздают инженерную задачу заключительного этапа, участники должны были писать программы на языке C++ и Python.

Разработана автоматическая система генерации уникального условия конкретной задачи для каждого участника, а также автоматизирован процесс проверки результата решения задачи.

Задача 1 (2 балла). Фильтрация сигналов.

Один из наиболее часто встречающихся методов фильтрации сигналов – метод скользящего среднего. В таком случае из общей выборки размером N выбирается локальная, состоящая из n элементов и k-тый элемент рассчитывается как среднее арифметическое k-той выборки (элементы основной выборки от k-n до k). Затем k увеличивается на 1 и процедура повторяется, при этом n неизменным. Если k-n < 0, то элемент считается равным нулю. Пусть дана выборка зашумленного синусоидального сигнала $x(t) = A\sin(\omega t) + w(t)$, где A - амплитуда, ω - круговая частота, w - аддитивная помеха. Напишите программу на языке C, определяющую дисперсию исходного сигнала, а затем дисперсию сигнала, прошедшего процедуру фильтрации по n элементам.

Дисперсия D мера разброса случайной величины, относительно ее математического ожидания (среднего сигнала).

$$D = \frac{\sum_{i=1}^{N} \left(x(t_i) - \bar{x}(t_i) \right)^2}{N-1}$$

За среднее значение сигнала принять $\bar{x}(t) = A\sin(\omega t)$.

Формат входных данных:

Первая строка: параметры сигнала число точек $N(5 \le N \le 10000)$ общей выборки и заданное число $n(1 \le n \le 100)$ локальной выборки, значение амплитуды $A(-1000.0 \le A \le 1000.0)$, значение круговой частоты $\omega(-\pi \le \omega \le \pi)$ Далее - два столбца, разделенных табуляцией. Первый столбец – время, второй – значения сигнала $x(t) = A\sin(\omega t) + w(t)$.

Формат выходных данных:

два числа с точностью 6 знаков после запятой, каждое в новой строке: дисперсия исходного сигнала и дисперсия фильтрованного сигнала.

Пример входных данных:

```
15 5 1.0 1.0
0.0 -0.41456659580494093
0.01 0.21464650810642005
0.02 0.20756670133274546
0.03 -0.020825303889748015
0.04 -0.3658466869439202
0.05 -0.38824196435413005
0.06 0.3187930234482115
0.07 -0.3202780262946304
0.08 0.1801071150373843
0.09 -0.19774814552844183
0.1 0.5640858833751679
0.11 0.2049294648895833
0.12 -0.162058299298603
0.13 0.023129988527424317
0.14 0.21563227823140296
```

Пример выходных данных:

```
0.088671
0.012303
```

Решение:

Текст программы:

```
#include <iostream>
#include <algorithm>
#include <vector>
#include <cmath>
#define ll long long
#define ld long double

ll N, n, k;
ld A, o, w, d1=0, d2=0, s=0;

ld sx(ld ti){
   return A*(sin(o*ti));   }

using namespace std;
int main() {

   cin>>N>>n>>A>>o;
   vector<ld>t(N), x(N);
   for(ll i=0; i<N; i++) {
      cin>>t[i]>>x[i];
      d1+=(x[i]-sx(t[i]))*(x[i]-sx(t[i]));
   }

   for(ll i=0; i<n; i++) {
      s+=x[i];
```

```
        d2+=((s/n)-sx(t[i]))*((s/n)-sx(t[i]));
    }

    for(ll k=n; k<N; k++){
        s=s-x[k-n]+x[k];
        d2+=((s/n)-sx(t[k]))*((s/n)-sx(t[k]));
    }

    cout.precision(6);
    cout<<fixed<<d1/(N-1);
    cout<<endl;
    cout<<fixed<<d2/(N-1);

}
```

Задача 2 (3 балла). Распознавание образов.

Решите следующую задачу на языке **Python**:

Имеется некоторая картинка размером n x m. Каждый пиксель имеет свой цвет, характеризующийся тремя компонентами: красным, зеленым и синим (RGB-модель). В каком-то месте этой картинки нарисован закрашенный фиолетовый(#5A009D) круг с некоторым целочисленным радиусом. Определите координаты центра круга и его радиус. Гарантируется, что на картинке изображен единственный круг соответствующего оттенка, полностью помещающийся в картинку, и фон не содержит пикселей данного цвета.

Входные данные

В первой строке записываются $n(1\le n\le 3000)$ количество "строк" в картинке и $m(1\le m\le 3000)$ количество "столбцов". В следующих n x m строках записывается по 33 числа от 0 до 255 включительно, характеризующие, соответственно, красную, зеленую и синюю компоненты цвета для каждого пикселя в порядке построчного обхода (сначала описываются mm элементов первой строки, затем второй, затем третей, ..., затем n-ной).

Выходные данные

Выведите через пробел три целых числа: координата строки центра окружности, координата столбца центра окружности и радиус окружности.

Примечание

Обратите внимание, что окружность отображаемая одним пикселем имеет радиус 0, крестиком 1 и т.д., то есть радиус соответствует наибольшему расстоянию между центрами пикселей и центром пикселя, являющимся центром окружности

Пример входных данных:

```
1 7
39 65 43
12 62 27
252 245 197
184 189 125
225 126 145
```

```
177 22 131
90 0 157
```

Пример выходных данных:

```
0 6 0
```

Решение:

Текст программы:

```
n, m = map(int, input().split())

image = [[tuple(map(int, input().split())) for u in range(m)] for i in range(n)]

binary_image = [[image[i][u] == (90, 0, 157) for u in range(m)] for i in range(n)]

crow = 0
crowc = 0

for i in range(n):
    if binary_image[i].count(True) > crowc:
        crow = i
        crowc = binary_image[i].count(True)

rot_binary_image = list(zip(*binary_image[::-1]))

ccol = 0
ccolc = 0

for i in range(m):
    if rot_binary_image[i].count(True) > ccolc:
        ccol = i
        ccolc = rot_binary_image[i].count(True)

r = (crowc-1)/2

print(crow, ccol, int(r))
```

Задача 3 (5 баллов). Навигация.

Путевая скорость самолета (скорость относительно земли) есть сумма векторов воздушной скорости самолета и скорости ветра: $\overline{V_k} = \bar{V} + \bar{W}$ где $\bar{V}_k$ - вектор путевой скорости, $\bar{V}$ - вектор воздушной скорости, $\bar{W}$ - вектор скорости ветра. Вектор скорости ветра задается модулем скорости ветра в м/с и углом ветра. Самолету требуется перелететь из точки 1 с координатами (φ1, λ1) в точку 2 с координатами (φ2, λ2), где φ - широта, λ - долгота. Выполнить полет требуется за заданное наперед время t. Определите, какова длина маршрута, какой должен быть угол пути, какой величины должна быть воздушная скорость самолета и каким должен быть его курс, чтобы он выполнил полётное задание. Считать, что вектор воздушной скорости совпадает с продольной осью самолета,

карта плоская, маршрут прямолинейный, в 1 градусе широты 111111 м, в 1 градусе долготы 60515 м.

Угол курса – угол между продольной осью самолета и направлением на север. *Угол ветра* – угол между направлением вектора скорости ветра и направлением на север *Угол пути* – угол между направлением на север и вектором путевой скорости. Диапазоны угла курса, угла ветра и угла пути $0 \le \psi < 3600 \le \psi < 360$ градусов, положительное направление – по часовой стрелке.

Формат входных данных:

В первой строке три целых числа – широта точки старта в формате градусы, минуты, секунды

Во второй строке три целых числа – долгота точки старта в формате градусы, минуты, секунды.

В третьей строке три целых числа – широта точки цели в формате градусы, минуты, секунды.

В четвертой строке три целых числа – долгота точки цели в формате градусы, минуты, секунды.

В пятой строке три целых числа – заданное время полета в минутах, модуль скорости ветра в м/с, угол ветра в градусах.

Формат выходных данных:

В строке 4 числа, записанные через пробел: длина маршрута в метрах, требуемый угол пути в градусах, модуль воздушной скорости в м/с, угол курса в градусах.

Комментарий к тесту:

Для полёта из Москвы в некоторую заданную точку за 33 минуты при ветре со скоростью 9 м/с и углом 299 градусов, длина маршрута составит 204 км 435 м, требуемый угол пути будет равен 139.9 градуса, модуль воздушной скорости -- 111.7 м/с и угол курса -- 138.2 градуса.

Пример входных данных:

```
57 45 13
37 36 93
56 20 47
39 48 8
33 9 299
```

Пример выходных данных:

```
204435.25218594205 139.89172008751964 111.7045466909329
138.2453061138902
```

Решение:

Текст программы:

```
# put your python code here
from math import sqrt,atan,degrees,radians,sin,cos
a=input().split()
shs=int(a[0])*60*60+int(a[1])*60+int(a[2])
a=input().split()
ds=int(a[0])*60*60+int(a[1])*60+int(a[2])
a=input().split()
shf=int(a[0])*60*60+int(a[1])*60+int(a[2])
a=input().split()
df=int(a[0])*60*60+int(a[1])*60+int(a[2])
a=list(map(int,input().split()))
t=60*a[0]; mv=a[1]; uv=radians(a[2])


rsh=abs(shf-shs)*111111/3600
arsh=(shf-shs)*111111/3600
rd=abs(df-ds)*60515/3600
ard=(df-ds)*60515/3600


dl=sqrt(rsh**2+rd**2)


if shf>shs and df>ds: um=degrees(atan(rd/rsh))
elif shf<shs and df>ds: um=degrees(atan(rsh/rd))+90
elif shf<shs and df<ds: um=degrees(atan(rd/rsh))+180
else: um=degrees(atan(rsh/rd))+270


dv=mv*t
rsh=abs(arsh-(dv*cos(uv)))
rd=abs(ard-(dv*sin(uv)))



dll=sqrt(rsh**2+rd**2)
u=dll/t


shf=shf-((dv*cos(uv))*3600/111111)
df=df-((dv*sin(uv))*3600/60515)


if shf>shs and df>ds: us=degrees(atan(rd/rsh))
elif shf<shs and df>ds: us=degrees(atan(rsh/rd))+90
elif shf<shs and df<ds: us=degrees(atan(rd/rsh))+180
else: us=degrees(atan(rsh/rd))+270

print(dl,um,u,us)
```

Задача 4 (на время). Движение в матрице.

Напишите программу, умеющую обходить прямоугольное поле по спирали по часовой стрелке, начиная с левого верхнего угла. Каждое прямоугольное поле представляет из себя матрицу, заполненную некоторыми словами, состоящими из латинских букв, цифр, знаков препинания и других знаков ASCII-таблицы в диапазоне [21, 7E]. Все слова между собой разделены не менее чем одним разделителем из следующих 3-х: пробел, перенос строки и табуляция.

ASCII Code Chart

	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	NUL	SOH	STX	ETX	EOT	ENQ	ACK	BEL	BS	HT	LF	VT	FF	CR	SO	SI
1	DLE	DC1	DC2	DC3	DC4	NAK	SYN	ETB	CAN	EM	SUB	ESC	FS	GS	RS	US
2		!	"	#	$	%	&	'	(	)	*	+	,	-	.	/
3	0	1	2	3	4	5	6	7	8	9	:	;	<	=	>	?
4	@	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O
5	P	Q	R	S	T	U	V	W	X	Y	Z	[	\	]	^	_
6	`	a	b	c	d	e	f	g	h	i	j	k	l	m	n	o
7	p	q	r	s	t	u	v	w	x	y	z	{	|	}	~	DEL

Формат входных данных:

В первой строке вводится целое число t ($0 \le t \le 50$) - количество тестов и целое число m ($1 \le m \le 1000$) - максимальный размер стороны матрицы. Далее для каждого из t тестов в отдельной строке вводятся два целых числа r ($1 \le r \le m$) и c ($1 \le c \le m$) - количество строк и столбцов входной матрицы, соответственно. Далее с новой строки перечисляются r x c - слов, разделенных разделителями.

Формат выходных данных:

Для каждого теста в отдельной строке выведите слова в требуемом порядке, разделенные пробелом.

Пример входных данных:

```
2 4
4 4
 1  2  3  4
12 13 14  5
11 16 15  6
10  9  8  7
2 3
I'd rather be
good. than lucky
```

Пример выходных данных:

```
1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16
I'd rather be lucky than good.
```

Решение:

Текст программы:

```
/*
  1  2  3  4
 12 13 14  5
 11 16 15  6
 10  9  8  7
*/
#include <cstdio>
#include <string>
#include <iostream>
#include <algorithm>
#include <vector>

using namespace std;

int main(void)
{
   int tests, max_size, row_size, col_size;
   scanf("%d%d", &tests, &max_size);

   int row, column;

   vector<vector<string> > mas(max_size, vector<string>(max_size));

   int way[4] = {1, 0, -1, 0};

   for (int test = 0; test < tests; ++test) {
      scanf("%d %d", &row_size, &col_size);
      for (int i = 0; i < row_size; ++i) {
         for (int j = 0; j < col_size; ++j) {
            cin >> mas[i][j];
         }
      }

      row = 0;
      column = -1;

      int row_len = col_size;
      int col_len = row_size;

      for (int lines = 0; lines < 2 * min(row_size, col_size); ++lines) {
         row_len -= lines % 2;
         col_len -= lines % 2;

         int cnt_elems;

         if (lines % 2 == 0) {
            cnt_elems = row_len;
         } else {
            cnt_elems = col_len;
         }

         for (int i = 0; i < cnt_elems; ++i) {
            row += way[(lines + 3) % 4];
            column += way[lines % 4];
            cout << mas[row][column] << ' ';
         }
      }
      printf("\n");
```

```
    }

    return 0;
}
```

Задача 5 (на время). Моделирование полета.

Введение

Рассматривается следующая ситуация. Беспилотный летательный аппарат самолетной схемы (БПЛА) движется в горизонтальной плоскости в окрестности прямолинейной линии заданного пути (ЛЗП) (рис. 1).

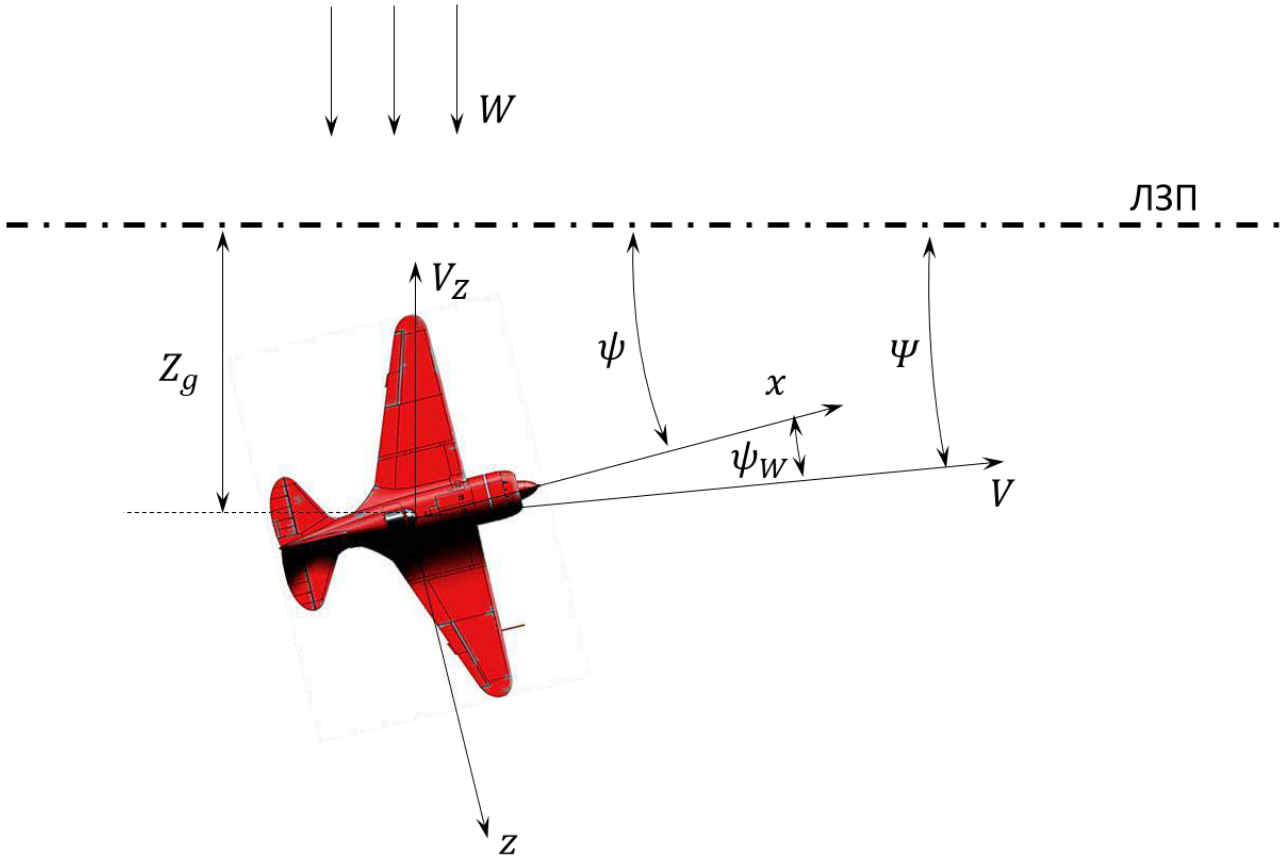

Рис. 1. Движение БПЛА в горизонтальной плоскости.

Его движение описывается следующими уравнениями:

$$\frac{dZ_g}{dt} = V_Z, \tag{1}$$

$$V_Z = -\frac{V}{57.3}\Psi, \tag{2}$$

$$\Psi = \psi + \psi_W, \tag{3}$$

$$\frac{d\psi}{dt} = -\frac{g}{V}\gamma, \tag{4}$$

$$\frac{d\gamma}{dt} = \omega_x, \tag{5}$$

$$\frac{d\omega_x}{dt} = \bar{M}_x^{\omega_x}\omega_x + \bar{M}_x^{\delta_э}\delta_э, \tag{6}$$

где

Z_g – нормальное боковое отклонение от ЛЗП [м],

V_Z – скорость бокового движения [м/с],

Ψ – угол пути [градусы],

ψ – угол курса [градусы],

ψ_W – приведенное к углу пути ветровое возмущение [градусы],

γ – угол крена [градусы],

ω_x – угловая скорость вокруг продольной оси БПЛА [градусы/с],

$\delta_э$ – угол отклонения элеронов [градусы],

V – воздушная скорость [м/с],

$\bar{M}_x^{\omega_x}$ – коэффициент собственного демпфирования [1/с],

$\bar{M}_x^{\delta_э}$ – коэффициент эффективности элеронов [1/с^2],

$g = 9.81$ – ускорение свободного падения [м/с^2],

$57.3 \approx 180/\pi$ – коэффициент перевода градусов в радианы.

Для обеспечения стабилизации ЛЗП используются три закона управления:

$$\psi^{зад} = K_{Zg}Z_g, \tag{7}$$

$$\gamma^{зад} = K_\psi\left(\psi - \psi^{зад}\right), \tag{8}$$

$$\delta_э = K_\gamma\left(\gamma - \gamma^{зад}\right) + K_{\omega_x}\omega_x, \tag{9}$$

где

$\psi^{зад}$ – заданный угол курса,

$\gamma^{зад}$ – заданный угол крена,

$\delta_э^{зад}$ – заданный угол отклонения элеронов;

K_{Zg}, K_ψ, K_γ, K_{ω_x} – постоянные коэффициенты законов управления.

Коэффициент K_{Zg} определяет величину требуемого курса в зависимости от бокового отклонения.

Коэффициент K_ψ определяет, насколько самолет должен накрениться, чтобы сравнять курс фактический и требуемый.

Коэффициенты K_γ, K_{ω_x} отвечают за стабилизацию угла крена.

Требуется:

1. Для заданных параметров самолета $\bar{M}_x^{\omega_x}$, $\bar{M}_x^{\delta_э}$, равных -3.0, -12.0 соответственно подобрать коэффициенты закона управления (9) K_γ, K_{ω_x} так, чтобы переходный процесс по углу крена $\gamma(t)\big|_{\gamma^{зад}=1}$ имел время $t_{пп}=1.3...1.5$ сек и перерегулирование $\sigma \le 5\%$.

2. Для заданной скорости полета V, заданного постоянного приведенного ветрового возмущения ψ_W, заданных коэффициентов K_{Zg}, K_ψ, коэффициентов K_γ и K_{ω_x} найденных в п.1 и нулевых начальных условий $Z(t)\big|_{t=0}=0$, $V_Z(t)\big|_{t=0}=0$, $\psi(t)\big|_{t=0}=0$, $\gamma(t)\big|_{t=0}=0$, $\omega_x(t)\big|_{t=0}=0$ решением разностных уравнений (10-18) (написание программы на выбранном вами языке программирования) получить значение ошибки стабилизации ЛЗП $\varepsilon_Z = Z_g(t)\big|_{t=\infty}$ в установившемся режиме (время моделирования устанавливать более 100 сек., при этом шаг моделирования равен 0.01).

ПРИЛОЖЕНИЕ

Малые углы

Если аргумент функций синус, косинус, тангенс и арктангенс мал (до 20-25 градусов), то с погрешностью до 10% допустимо выполнить следующую замену:

$$\sin(x) \approx x,$$
$$\cos(x) \approx 1,$$
$$\tan(x) \approx x,$$
$$\arctan(x) \approx x,$$

в чем можно убедиться с помощью калькулятора.

Вывод уравнений

Уравнение (1). По определению, $\Delta Z_g = V_Z \Delta t$. Разделим на Δt и перейдем к пределу: $\lim_{\Delta t \to 0} \frac{\Delta Z_g}{\Delta t} = \frac{dZ_g}{dt} = V_Z$.

Уравнение (2). Из рис. 2 получаем: $V_Z = V\sin(-\Psi) = -V\sin(\Psi)$. Считая Ψ малым углом и переходя к градусам получим: $-V_Z = \dfrac{V}{57.3}\Psi$.

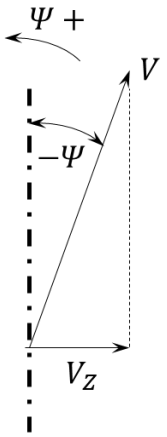

Рис. 2. Связь боковой и путевой скоростей.

Уравнение (3). Из рис. 3 получаем: $\psi_W = \arctan\left(\frac{W}{V}\right) \approx \frac{W}{V}$. Также, предполагая, что углы малые $V = V_k \cos(\psi_W) \approx V_k$. Т.е. путевая и воздушная скорости приблизительно равны.

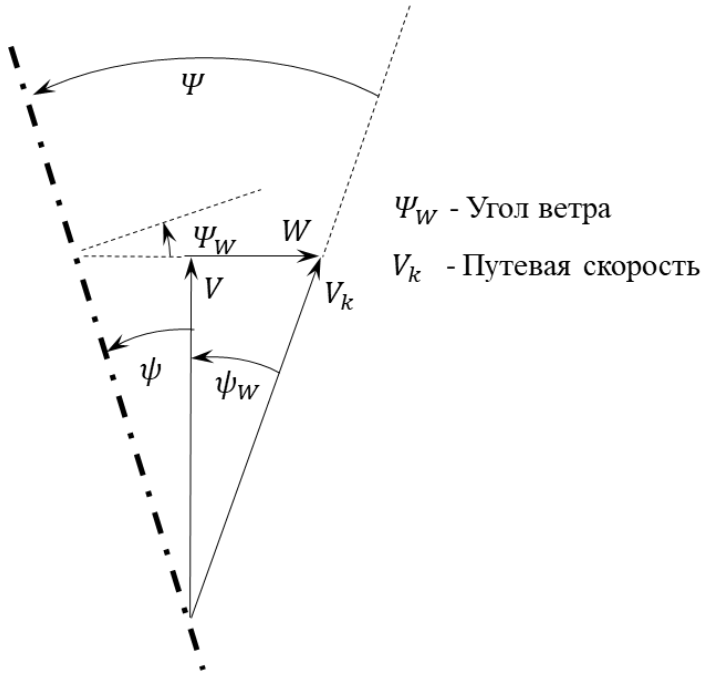

Рис. 3. Связь скорости ветра с приведенным ветровым возмущением.

Уравнение (4). При накренении самолета (рис. 4) возникает боковая проекция Z подъемной силы Y (направлена перпендикулярно плоскости крыльев). Поскольку угол крена принимается малым, верны следующие утверждения: $G = Y\cos\gamma \approx Y$, $Z = Y\sin\gamma = \frac{Y\gamma}{57.3} = \frac{G\gamma}{57.3}$. Разделив последнее уравнение на массу перейдем к боковому ускорению: $a_Z = \frac{g}{57.3}\gamma$. Движение с постоянной скоростью и центростремительным

ускорением есть движение по окружности. С одной стороны, центростремительное ускорение может быть найдено как $a_Z = \frac{V^2}{R_{вир}}$, а с другой $a_Z = \left(\frac{d\psi}{dt}\right)^2 R_{вир}$. Приравнивая правые части двух полученных уравнений и учитывая, что положительное направление вращения против часовой стрелки, получим: $\frac{d\psi}{dt} = -\frac{g}{57.3V}\gamma$, курс измеряется в радианах.

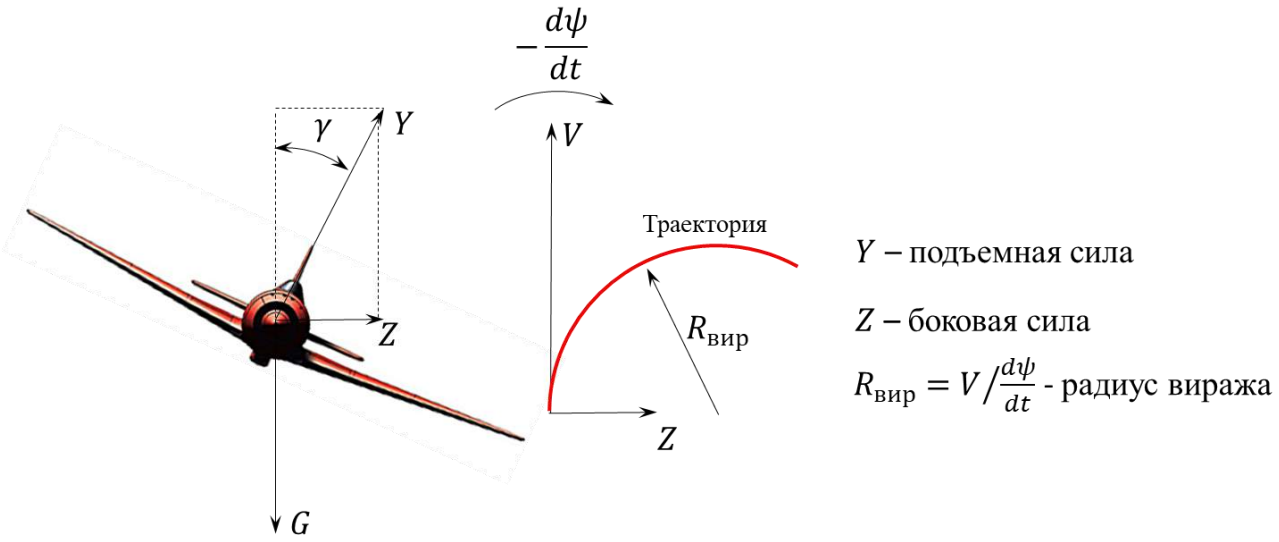

Рис. 4. Связь изменения курса и угла крена.

Уравнение (5). Вывод аналогичен уравнению 1.

Уравнение (6). Основным фактором изменения угловой скорости БПЛА является отклонение элеронов $\delta_э$. За то, насколько изменится угловая скорость БПЛА вокруг продольной оси при отклонении элеронов на 1 градус отвечает коэффициент эффективности элеронов $\bar{M}_x^{\delta_э} < 0$, определяемый экспериментально в ходе аэродинамических продувов и расчетов. Чем больше этот коэффициент по модулю, тем энергичнее реагирует самолет на управление. Однако наличие угловой скорости препятствует процессу управления. Данное обстоятельство вызвано тем, что ближе к концам крыла происходит изменение угла атаки за счет скорости вращательного движения, что приводит к росту сопротивления воздушной среды пропорционально угловой скорости ω_x и торможению вращения. Данное явление носит название собственного демпфирования и характеризуется коэффициентом $\bar{M}_x^{\omega_x} < 0$, показывающим, в каком темпе будет падать угловая скорость вращения при ее текущем значении.

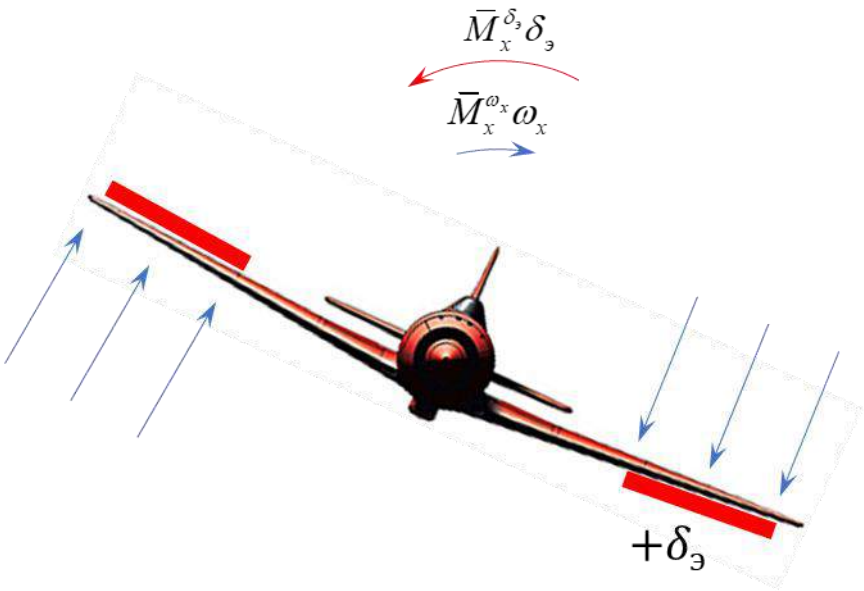

Рис. 5. Приведенные управляющий и демпфирующий моменты самолета.

Переход от дифференциальных уравнений к разностным

Заменяя производные в уравнениях (1 - 6) конечными разностями по схеме $\frac{dx}{dt} = \lim_{\Delta t \to 0} \frac{\Delta x}{\Delta t} \approx \frac{\Delta x}{\Delta t} = \frac{x[k+1] - x[k]}{\Delta t}$ (метод Эйлера), получим следующую систему разностных уравнений, пригодную для моделирования на ЭВМ:

$$Z_g[k+1] = Z_g[k] + V_Z[k]\Delta t, \tag{10}$$

$$V_Z[k] = -\frac{V}{57.3}\Psi[k], \tag{11}$$

$$\Psi[k] = \psi[k] + \psi_W[k], \tag{12}$$

$$\psi[k+1] = \psi[k] - \frac{g}{V}\gamma[k]\Delta t\ , \tag{13}$$

$$\gamma[k+1] = \gamma[k] + \omega_x[k]\Delta t, \tag{14}$$

$$\omega_x[k+1] = \omega_x[k] + M_x^{\omega_x}\Delta t\,\omega_x[k] + M_x^{\delta_э}\Delta t\,\delta_э[k], \tag{15}$$

$$\delta_э = K_\gamma\left(\gamma[k] - \gamma^{зад}[k]\right) + K_{\omega_x}\omega_x[k], \tag{16}$$

$$\gamma^{зад}[k] = K_\psi\left(\psi[k] - \psi^{зад}[k]\right), \tag{17}$$

$$\psi^{зад}[k] = K_{Zg}Z_g[k], \tag{18}$$

Качество переходных процессов

Время нарастания – время $t_н$ за которое система в первый раз входит в 5%

окрестность от установившегося значения (5% трубка) $x_{уст}$, т.е. $x(t) \in \{0.95x_{уст}, 1.05x_{уст}\}$.

Время переходного процесса – время $t_{пп}$ вхождения в 5% трубку установившегося значения, после которого система не выходит за пределы 5% трубки.

Перерегулирование – отношение разности максимального и установившегося значения к установившемуся в процентах: $\sigma = \dfrac{x_{max} - x_{уст}}{x_{уст}} 100\%$

Установившееся значение – значение $x_{уст}$ при $t \to \infty$.

Установившаяся ошибка – разница между заданным сигналом и установившимся значением: $\varepsilon = u - x_{уст}$.

Рассмотрим пример. Система описывается уравнением:

$$\frac{d^2x}{dt^2} + a_1 \frac{dx}{dt} + a_0 x = bu,$$

где $a_1 = 1.1$, $a_0 = 1.2$, $b = 1$. Такая система обладает переходным процессом (при единичном ступенчатом воздействии $u = 1$) со следующими показателями качества (рис. 6):

Время нарастания: 2.1 сек

Время переходного процесса: 4.8 сек

Перерегулирование: 18%

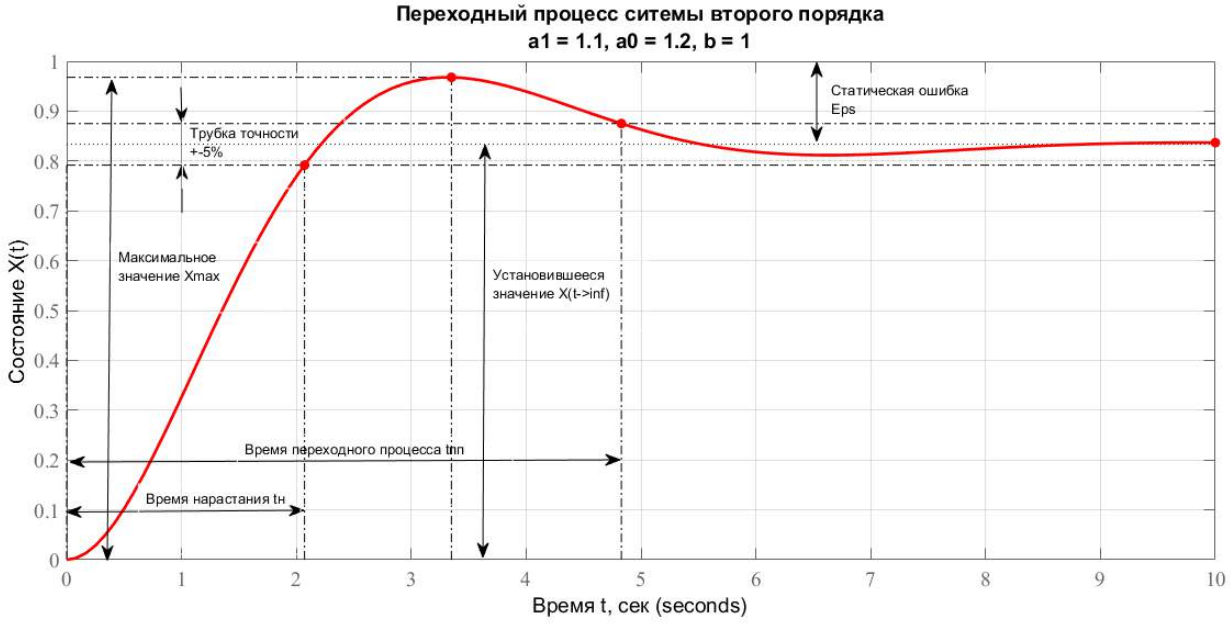

Рис. 6. Пример переходного процесса при единичном ступенчатом воздействии и его основные характеристики.

Пошаговая инструкция для разработки программы

Первая часть задачи (решается отдельно на личном ПК, отправлять на Stepik не требуется)

1. Система управления углом крена описывается уравнениями (5), (6), (9). Для нее требуется подобрать коэффициенты закона управления (9) K_γ, K_{ω_x}. Для программирования используются разностные аналоги этих уравнений (14), (15), (16). При этом шаг Δt принять не менее 10^{-3}, $\gamma^{зад} = 1$.

2. Используя эти уравнения вычислить $\gamma(t)$ при $t \in (0...10)$ с шагом $\Delta t \le 10^{-3}$. При этом K_γ и K_{ω_x} задаются самостоятельно и подлежат определению с точки зрения требований к переходному процессу $\gamma(t)$.

3. После получения $\gamma(t), t \in (0...10)$ и заданных K_γ и K_{ω_x} определить время переходного процесса t_{nn} и перерегулирование σ (см. условие задачи п.1).

4. Если время переходного процесса t_{nn} и перерегулирование σ соответствуют требованиям в п.1. условии задачи, то процесс поиска коэффициентов считается завершенным. В противном случае, выбрать другие коэффициенты K_γ, K_{ω_x} и повторить процедуру 2 – 4.

Вторая часть задачи (решение отсылается на Stepik)

1. Уравнения (1-9) описывают динамику системы автоматической стабилизации ЛЗП. Требуется определить ошибку стабилизации ЛЗП в установившемся режиме при постоянном заданном ветровом возмущении ψ_W. Для этого программируется решение разностных уравнений (10-18) с целью получения зависимости $Z_g(t)$, при $t \in (0...50)$, $\Delta t \le 10^{-2}$.

2. По имеющемуся переходному процессу $Z_g(t)$ определить установившуюся ошибку относительно нулевого отклонения. Это и будет ответом в задаче.

Формат входных данных:

Четыре вещественных числа -- параметры самолета и данные о режиме полета:

V $(80 \le V \le 120)$,
ψ_W $(0.1 \le \psi_W \le 5.0)$,
K_{Z_g} $(0.1 \le K_{Z_g} \le 0.15)$,
K_ψ $(4.0 \le K_\psi \le 7.0)$.

Формат выходных данных:

Значение ошибки стабилизации ЛЗП ε_Z с погрешностью в пределах до 11.0 метра.

Пример входных данных:

```
98.3 0.14 0.13 4.48
```

Пример выходных данных:

```
-1.1326111236689242
```

Решение:

Текст программы:

```
#include <iostream>

using namespace std;

int main()
{

   double M_x_omega = -3.0, M_x_delta = -12.0;

   double gamma_zad = 0.0;

   double t = 0.0;
   double step = 10000.0 / 100000;

   double k_gamma = 0.45, k_omega = 0.035;

   double Z_g[100001], V_z[100001], psi[100001], \
   omega[100001], delta[100001], gamma[100001];

   double V, psi_W, K_Z_g, K_psi;
   cin >> V >> psi_W >> K_Z_g >> K_psi;
   //V = 120;
   //psi_W = 3.4;
   //K_Z_g = 0.13;
   //K_psi = 5;

   gamma[0] = 0;
   omega[0] = 0;
   delta[0] = 0;
   V_z[0] = 0;
   Z_g[0] = 0;

   double g = 9.81;
   double psi_zad;

   for (int k = 0; k < 100000; ++k) {

      gamma[k + 1] = gamma[k] + omega[k] * step;
      omega[k + 1] = omega[k] + (M_x_omega * omega[k] + M_x_delta * delta[k]) * step;
      delta[k + 1] = k_gamma * (gamma[k] - gamma_zad) + k_omega * omega[k];
      V_z[k] = -V * (psi[k] + psi_W) / 57.3;
      Z_g[k + 1] = Z_g[k] + V_z[k] * step;
      psi[k + 1] = psi[k] - g * gamma[k] * step / V ;
      psi_zad = K_Z_g * Z_g[k];
      gamma_zad = K_psi * (psi[k] - psi_zad);
   }
```

```
    cout << Z_g[10000] << endl;

    return 0;
}
```