|  |  |
| --- | --- |
| **Gerb-BMSTU_01** | **«Московский государственный технический университет  имени Н.Э. Баумана»**  **(МГТУ им. Н.Э. Баумана)** |

**Домашнее задание**

по курсу "Принципы автономного наведения ЛА "

"Построение программной траектории движения ЛА методом динамического программирования"

Вариант № 2

Студент Пашинин С.А.

Группа ИУ2-62

2009 г.

1. **Исходные данные**
2. **Характеристики самолета и траектории**

|  |  |
| --- | --- |
| Самолет | ТУ-134 |
| Масса, “взлетный вес” | 47000 кг |
| Площадь крыла | 127 м2 |
| Кол-во двигателей и тип | 2 двигателя Д-30 |
|  |  |
| Макс. мощность от номинальной | 100% |
| Критерий оптимизации | время |
| Начальная высота | 300м |
| Начальная скорость | 310 км/ч |
| Конечная высота | 6000м |
| Конечная скорость | 700 км/ч |

1. **Уравнения математической модели продольного движения ЛА:**

 (1)

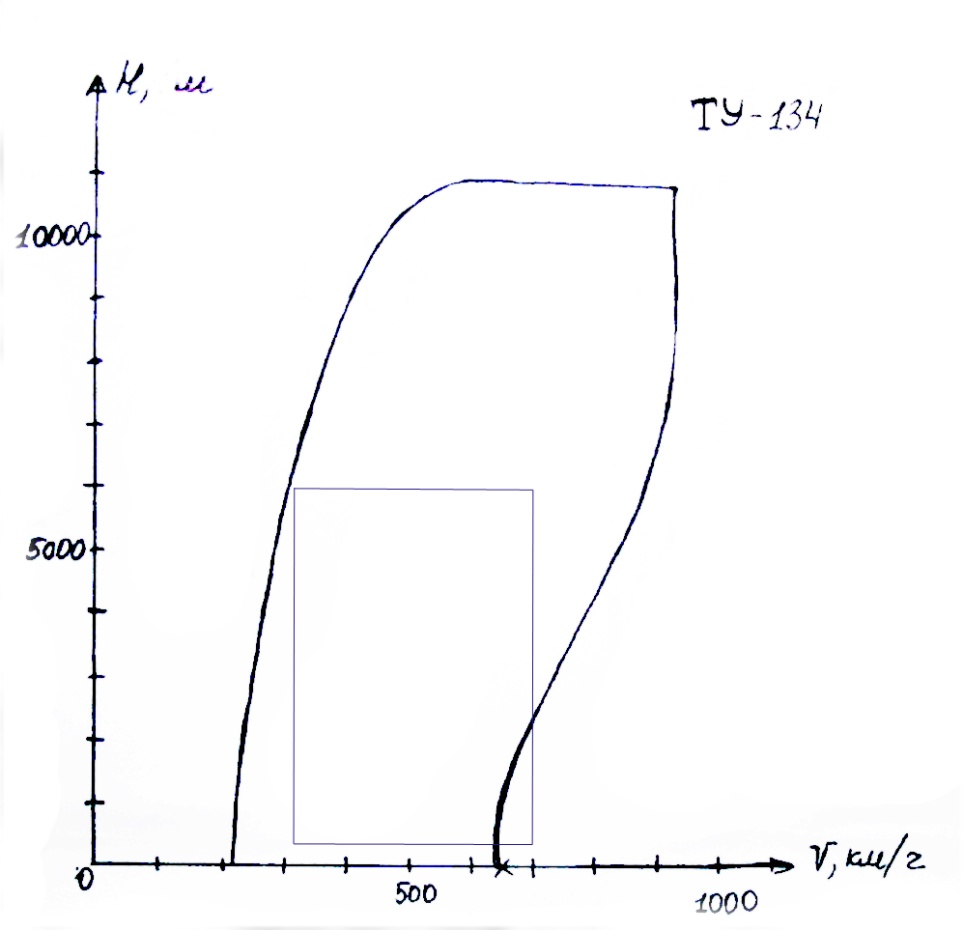
 (2)

 (3)

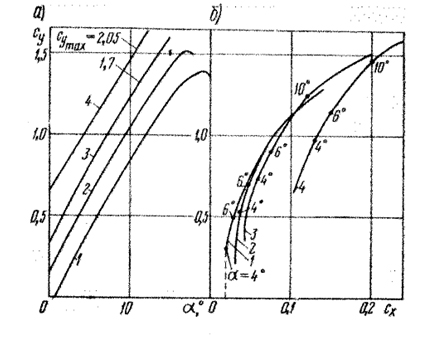
 (4)

 (5)

1. **Аэродинамические характеристики ЛА**



**рис. 1.** Полетная область самолета и область для моделирования полета.



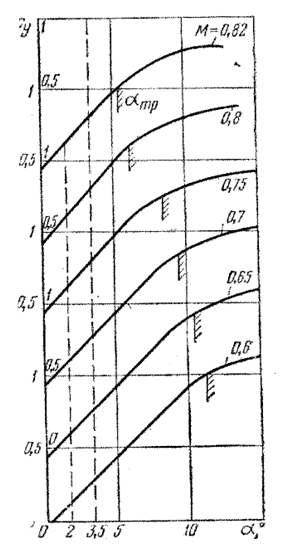
**Рис. 2.** График зависимости коэффициента подъемной силы от угла атаки (а) и поляры самолета с разметкой углов атаки (б) при убранных шасси для чисел M<0.4

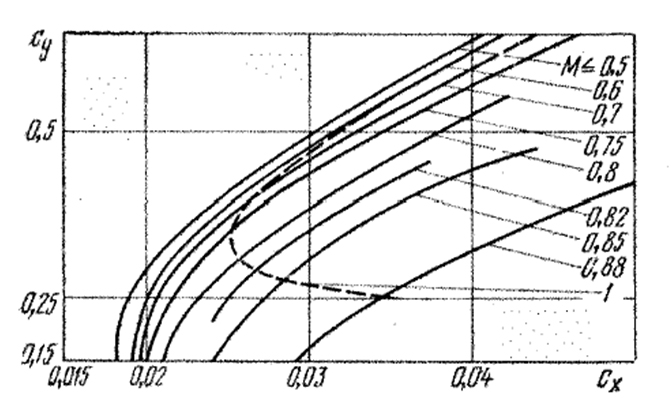
1 – закрылки убраны

2 – закрылки отклонены на 10 ̊

3 – закрылки отклонены на 20 ̊

4 – закрылки отклонены на 38 ̊, щиток убран



 **Рис. 4.** Поляры самолета ТУ-134А для больших чисел M полета

1 – поляра режимов горизонтального полета G=43000кгс; H=10000м.

**Рис. 3.** Влияние числа M на

зависимость



**Рис. 5.** Зависимость тяги двух двигателей на номинальном режиме работы от числа М полета

в стандартных условиях (с учетом потерь) на разных высотах Н = 0÷11км ; tмca .



**Рис. 6.** Зависимость удельного расхода топлива в стандартных условиях от скорости полета

для номинального режима работы двигателя (с учетом потерь) на разных высотах.



**Рис. 7.** Зависимость удельного расхода топлива на номинальном режиме от высоты полета (с

учетом потерь).

1. **Параметры стандартной атмосферы по ГОСТ4401-81.**

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| Геометричес-кая высота h, м | Величины в функции геометрической высоты | | | | | | |
| Геопо-тенциаль ная высота Н, м | Температура | | Давление | | Плотность *кг/м3* | Ускоре-ние сво-бодного падения g, м/с2 |
| Т, К | t°, C | Па | мм рт. ст. |
| 0 | 0 | 288,15 | 15 | 1,01325E+05 | 7,60000E+02 | 1,22500E+00 | 9,8066 |
| 500 | 500 | 284,9 | 11,75 | 9,54613E+04 | 7,16019E+02 | 1,16727E+00 | 9,8051 |
| 1000 | 1000 | 281,651 | 8,501 | 8,98763E+04 | 6,74128E+02 | 1,11166E+00 | 9,8036 |
| 2000 | 2000 | 275,154 | 2,004 | 7,95014E+04 | 5,96310E+02 | 1,00655E+00 | 9,8005 |
| 3000 | 2999 | 268,659 | -4,491 | 7,01212E+04 | 5,25952E+02 | 9,09254E-01 | 9,7974 |
| 4000 | 3997 | 262,166 | -10,984 | 6,16604E+04 | 4,62491E+02 | 8,19347E+01 | 9,7943 |
| 5000 | 4996 | 255,676 | -17,474 | 5,40483E+04 | 4,05359E+02 | 7,36429E-01 | 9,7912 |
| 6000 | 5994 | 249,187 | -23,963 | 4,72176E+04 | 3,54161E+02 | 6,60111E-01 | 9,7882 |
| 7000 | 6992 | 242,7 | -30,45 | 4,11051E+04 | 3,08315E+02 | 5,90018E-01 | 9,7851 |
| 8000 | 7990 | 236,215 | -36,935 | 3,56516E+04 | 2,67409E+02 | 5,26783E-01 | 9,782 |
| 9000 | 8987 | 229,733 | -43,417 | 3,08007E+04 | 2,31024E+02 | 4,67063E-01 | 9,7789 |
| 10000 | 9984 | 223,252 | -49,898 | 2,64999E+04 | 1,98765E+02 | 4,13510E-01 | 9,7759 |
| 11000 | 10981 | 216,774 | -56,376 | 2,26999E+04 | 1,70264E+02 | 3,64801E-01 | 9,7728 |
| 12000 | 11977 | 216,65 | -56,5 | 1,93994E+04 | 1,45507E+02 | 3,11937E-01 | 9,7697 |
| 13000 | 12973 | 216,65 | -56,5 | 1,65796E+04 | 1,24357E+02 | 2,66595E-01 | 9,7667 |
| 14000 | 13969 | 216,65 | -56,5 | 1,41703E+04 | 1,06286E+02 | 2,27855E-01 | 9,7636 |
| 15000 | 14965 | 216,65 | -56,5 | 1,21118E+04 | 9,08459E+01 | 1,94755E-01 | 9,7605 |
| 16000 | 15960 | 216,65 | -56,5 | 1,03528E+04 | 7,76524E+01 | 1,66470E-01 | 9,7575 |
| 17000 | 16955 | 216,65 | -56,5 | 8,84970E+03 | 6,63782E+01 | 1,42301E-01 | 9,7544 |
| 18000 | 17949 | 216,65 | -56,5 | 7,56521E+03 | 5,67437E+01 | 1,21647E-01 | 9,7513 |
| 19000 | 18943 | 216,65 | -56,5 | 6,46747E+03 | 4,85100E+01 | 1,03995E-01 | 9,7483 |
| 20000 | 19937 | 216,65 | -56,5 | 5,52929E+03 | 4,14734E+01 | 8,89097E-02 | 9,7452 |
| 22000 | 21924 | 218,574 | -54,576 | 4,04748E+03 | 3,03586E+01 | 6,45096E-02 | 9,7391 |
| 24000 | 23910 | 220,65 | -52,59 | 2,97174E+03 | 2,22898E+01 | 4,69377E-02 | 9,733 |
| 26000 | 25894 | 220,544 | -50,606 | 2,18837E+03 | 1,64141E+01 | 3,42565E-02 | 9,7269 |
| 28000 | 27977 | 224,527 | -48,623 | 1,61619E+03 | 1,21224E+01 | 2,50762E-02 | 9,7208 |
| 30000 | 29859 | 226,509 | -46,641 | 1,19703E+03 | 8,97843E+00 | 1,84101E-02 | 9,7147 |
| 35000 | 34808 | 236,513 | -36,637 | 5,74592E+02 | 4,30979E+00 | 8,46334E-03 | 9,6995 |
| 40000 | 39750 | 250,35 | -22,8 | 2,87143E+02 | 2,15375E+00 | 3,99566E-03 | 9,6844 |
| 50000 | 49610 | 270,65 | -2,5 | 7,97787E+01 | 5,98389E-01 | 1,02687E-03 | 9,6542 |

D:\baumanka\sem6\ПАН\finish\PNK0016.TIF

Все необходимые аппроксимированные зависимости программа получает автоматически из таблиц с данными (в файлах .txt). Эти зависимости вычисляются, и коэффициенты хранятся в массивах. Вручную аппроксимированные функции в текст программы не прописываются.

Некоторые функции, полученные из таблицы для ГОСТа атмосферы:

* Ускорение свободного падения на высоте Н в метрах: , м/с2
* Плотность воздуха на высоте Н в метрах: , кг/м3
* Скорость звука на высоте Н в метрах: , м/с

1. **Время полета на элементарных участках**

Общие уравнения движения ЛА:

 (1)

 (2)

 (3)

 (4)

 (5)

где ,

, 

*X* – сила лобового сопротивления

*Y* – подъемная сила

Су – коэффициент подъемной силы

Сх – коэффициент лобового сопротивления

– угол установки двигателей, при расчете примем *=0*

 – угол атаки

 – угол наклона траектории

*q* – скоростной напор

Рассмотрим три режима:

* **разгон**

,, 

 => 



Найдем угол атаки, считая его малым:

из (2): ,

т.к. , то 

* **подъем-разгон**

, , , 



 при 

Получим выражение для угла :

 (1)

Будем считать, что , тогда

 (2)

Подставим (2) в выражение (1):



Выразим угол , считая его малым:

, учитывая , 

Значение угла α получим аналогично, как для режима «разгон»: 

* **подъем**

, , 





1. **Алгоритм работы программы**

В работе программы используется стек, где хранятся указатели на текущие точки, для которых нужно вести дальнейший просмотр траектории методом Беллмана.

Стек ВСЕГДА упорядочен таким образом, что в конце стека находится точка, от которой время до последней точки траектории минимально.

Размер стека характеризуется линейными размерами заданной области, поэтому много памяти он не отнимет.

Исходя из этого, алгоритм поиска оптимальной траектории смотри на след. странице.

Примечания:

После завершения работы алгоритма для каждой точки вычисленной траектории, начиная с первой, хранится переход к следующей точке, соответствующий оптимальной траектории (параметр NextID класса Point).

Если в алгоритме просчитывать абсолютно все точки области, то получим поле из точек, причем для каждой будет известна оптимальная траектория до указанной конечной точки.

Указание конечной точки траектории

Начало

*Тело процедуры (алгоритма)*

Просматриваем все точки, из которых можно попасть в указанную

Определяем id точки в массиве и общее время до конечной точки

Точка уже была пройдена?

Время в этот раз меньше?

присвоить новое значение общего времени до конечной точки

указать следующую точку для перехода из данной

обновить положение точки в стеке

Нет

Да

Нет

Да

присвоить значение общего времени до конечной точки

указать следующую точку для перехода из данной

добавить точку в стек

Да

Нет

Есть ли еще траектории?

Одна тр-ия получена и невозможна лучшая

Нет

Да

указать последнюю точку из стека

удалить последнюю точку стека

Конец

**Текст программы**

Point\* point; - указатель на конечную точку траектории.

(Начальная вседа space[0])

void MakeTrajectory(Point\* point)

{

if (point==NULL) return;

for (int i=0; i<point->ConnectionsCount; i++) //Смотрим все точки, из

//которых можно попасть в данную

{

int id\_from = point->Connections[i].Point\_ID; //id точки в массиве

//время до последней точки траектории

double len = point->LengthToEnd + point->Connections[i].time;

if (space[id\_from].NextID != -1)//если добавляем в стек точку для

//которой уже считали общее время полета,

{ // то нужно проверить, будет ли эта траектория лучше

if (len < space[id\_from].LengthToEnd)//И если общее время

получилось меньше, выбираем этот путь

{

space[id\_from].LengthToEnd = len;

space[id\_from].NextID = point->id;

q.add(&space[id\_from], true); //и перемещаем точку в

//стеке, “говоря”, что она стала более оптимальной

} // а если эта траектория хуже, то ничего не делать.

} else //Если точка не была пройдена, просто добавить ее в стек.

{

space[id\_from].LengthToEnd = len;

space[id\_from].NextID = point->id;

q.add(&space[id\_from]);

}

}

// Если есть еще возможные траектории

if (q.A.size()>0)

{

// Остановиться, когда все оставшиеся пути дают время, большее, чем уже

// найденная траектория.

if (space[0].NextID != -1)

if (q.A[q.A.size()-1]->LengthToEnd > space[0].LengthToEnd) return;

// Продолжить алгоритм со следующей наиболее оптимальной точкой.

Point\* pp = q.A.back();

q.A.pop\_back();

MakeTrajectory(pp);

}

};

**Используемые классы:**

class QUEUE // Очередь текущих проверяемых точек

{

public:

vector<Point\*> A;

void QUEU() {A.clear(); A.resize(0);};

void add(Point\* p, bool modify=false)

{

if (modify)

{

for (unsigned int i=0; i<A.size(); i++)

{

if (A[i] == p)

{

A.erase(A.begin()+i);

i--;

}

}

}

vector<Point\*>::iterator it;

for (unsigned int i=0; i<A.size(); i++)

{

if (A[i]->LengthToEnd < p->LengthToEnd)

{

it = A.begin();

A.insert ( it+i , p );

return;

}

}

it = A.end();

A.insert ( it , p );

};

};

class Point // Точка в области моделирования полета

{

public:

Point() {Connections = NULL; ConnectionsCount = 0; LengthToEnd=0;

NextID=-1; OutOfRange=false;};

bool OutOfRange;

double V, H;

Connection\* Connections;

int id, NextID;

int ConnectionsCount;

double LengthToEnd; // Просчитанное время от

//этой точки до конечной точки траектории

void SetConnections(int num)

{

ConnectionsCount = num;

Connections = new Connection [num];

}

};

class Connection

{public:

Connection() {time = 0;};

int Point\_ID; // Связь с точкой Point\_ID

double time; // время полета

};

График оптимальной траектории (набор высоты(м) в зависимости от скорости(км/ч))

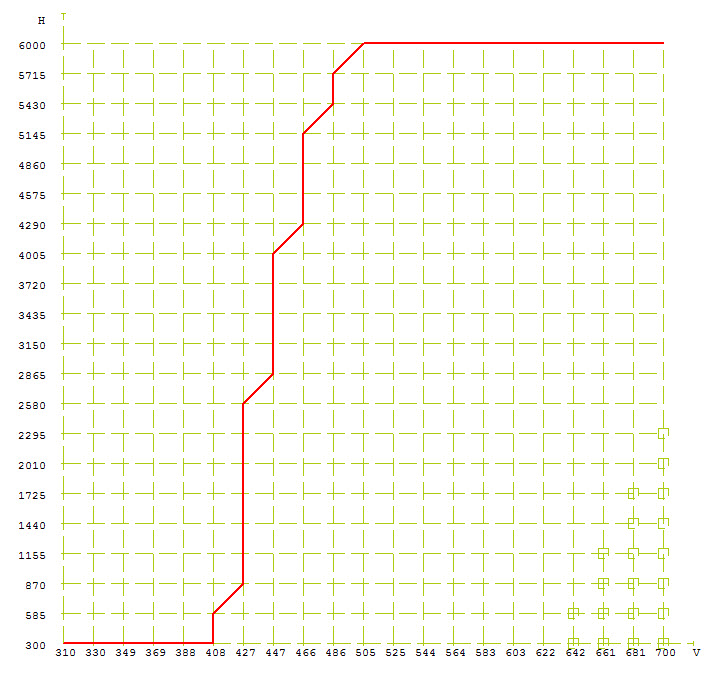
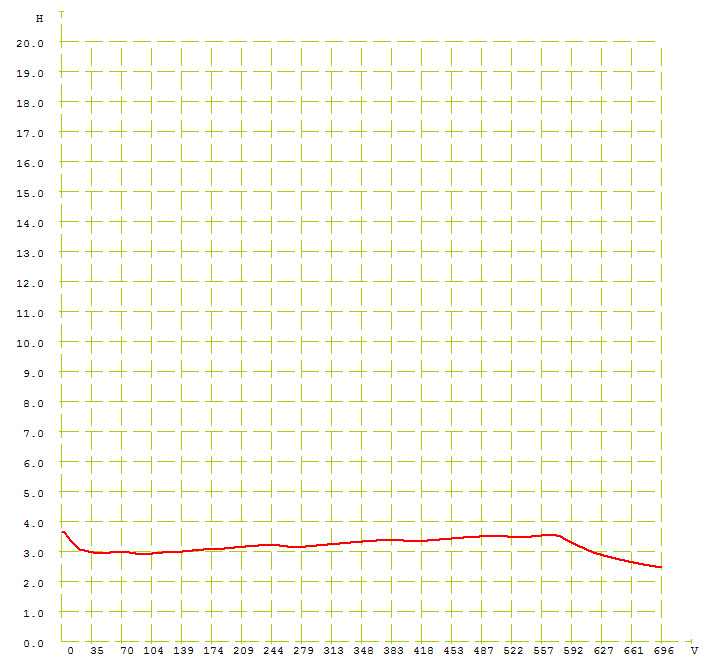
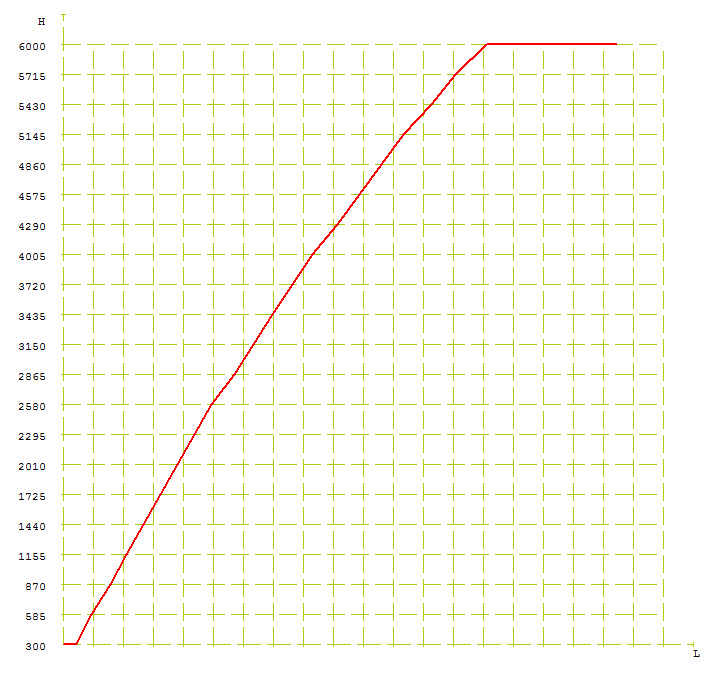


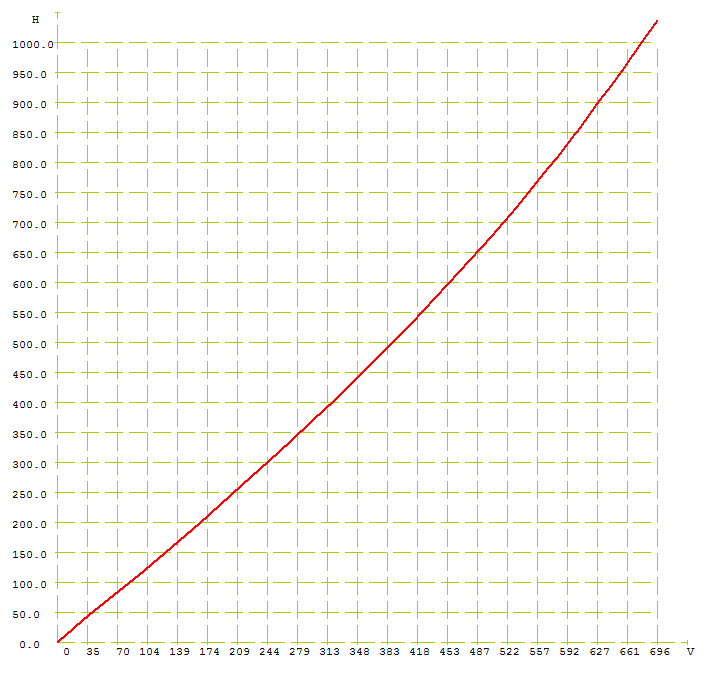
График угла атаки( ̊) в зависимости от времени(c):



Набор высоты от дальности полета (качественно):



Расход топлива (кг) от времени полета (с):



Общее время полета: 696 с

Расход топлива: 1038 кг за время данного полета

// Время на участке "Разгон". V[км/ч]; H[м]

double timeX(double V1, double V2, double H)

{

V1 = V1/3.6; //перевод в м/с

V2 = V2/3.6;

double P\_ = (P\_Force(V2,H) + P\_Force(V1,H)) / 2;

//Среднее значение тяги двигателей на участке

double alpha\_ = (alpha(V2,H) + alpha(V1,H)) / 2;

//Среднее значение угла атаки на участке

double X\_ = (X(V2,H) + X(V1,H)) / 2;

//Среднее значение силы сопротивления на участке

return m\*(V2-V1) / (P\_\*cos(alpha\_/57.3) - X\_);

}

// Время на участке "Подъем". V[км/ч]; H[м]

double timeY(double V, double H1, double H2)

{

V = V/3.6; //перевод в м/с

double P\_ = (P\_Force(V,H1) + P\_Force(V,H2)) / 2;

//Среднее значение тяги двигателей на участке

double alpha\_ = (alpha(V,H1) + alpha(V,H2)) / 2;

//Среднее значение угла атаки на участке

double X\_ = (X(V,H1) + X(V,H2)) / 2;

//Среднее значение силы сопротивления на участке

double theta\_ = (thetaY(V,H1) + thetaY(V,H2)) / 2;

//Среднее зн. угла

return (H2-H1) / (V\*sin(theta\_));

}

// Время на участке "Подъем-Разгон". V[км/ч]; H[м]

double timeXY(double V1, double V2, double H1, double H2)

{

V1 = V1/3.6;

V2 = V2/3.6;

double V\_ = (V1+V2)/2;

double H\_ = (H1+H2)/2;

double C = (V2-V1)/(H2-H1);

double theta\_ = thetaXY(V\_,H\_,C);

return 1/(C\*sin(theta\_)) \* log(V2/V1);

}