Тема № 4.
Математическое моделирование процессов функционирования интегрированных бортовых систем беспилотных маневренных летательных аппаратов

Групповое занятие № 6.
Функциональная схема объектно-ориентированного программно математического обеспечения для математического моделирования интегрированных систем навигации и наведения беспилотных маневренных летательных аппаратов

Учебные цели занятия

Знать:

- **моделирование неуправляемого движения летательных аппаратов;**
- **моделирование бортового измерительного комплекса;**
- **моделирование блока навигации и управления;**
- **полную объектную структуру моделирования.**

Отводимое время на занятие 90 минут

Учебные вопросы занятия

1. Моделирование неуправляемого движения летательных аппаратов.

2. Моделирование бортового измерительного комплекса.

3. Моделирование блока навигации и управления.

4. Полная объектная структура моделирования.

Литература на самоподготовку

1. Управление и наведение беспилотных маневренных летательных аппаратов на основе современных информационных технологий. Под ред. М. Н. Красильщикова и Г. Г. Себрякова. – М.: Физматлит, 2003. Стр. 204 – 247.

2. Конспект занятий.

В настоящее время наиболее эффективным подходом к разработке программно-математического обеспечения для математического моделирования интегрированных систем навигации и наведения высокоманевренных ЛА является объектно-ориентированный подход (ООП).

При использовании ООП возникает проблема определения так называемых базовых классов, т. е. выделение самых общих, типовых черт исследуемых процессов и построения соответствующей иерархии этих классов.

Рекомендации по формированию объектной схемы программно - математического обеспечения.

1. Все реально существующие объекты исследования, такие, как ЛА, система управления, блок измерителей и т.п., представляющие собой системы с конечным количеством входов и выходов, должны быть представлены своими аналогами-классами.

С точки зрения программиста соответствующий класс должен являться «черным ящиком» с некоторым количеством свойств, но со скрытым механизмом функционирования.

2. Построение иерархической цепочки таких классов целесообразно начинать с самого общего, абстрактного класса, у которого определены лишь наиболее общие, типовые для всей предполагаемой цепочки, поля, а методы объявлены как виртуальные и абстрактные.

В таких классах объявлены только структуры полей и шаблоны методов, а сами тела методов отсутствуют, что требует их перекрытия в классах-потомках.

3. Если при моделировании используются сложные численные алгоритмы, требующие большого числа настроек и дополнительных процедур, то необходимо построить библиотеки соответствующих классов, реализующие упомянутые алгоритмы.

При этом метод, детализирующий исходную математическую задачу, должен быть объявлен абстрактным с целью дальнейшего перекрытия в потомке уже непосредственно в рамках данного проекта (например функция вычисления правых частей системы обыкновенных дифференциальных уравнений).

4. Вспомогательные процедуры и простые алгоритмы целесообразно оформить в виде отдельных модулей, не облекая их в классы, с целью упрощения общей структуры, для повышения быстродействия программы.

Так, например, функции и процедуры матричной алгебры, алгебры комплексных чисел, кватернионов, тензоров и т. п. лучше всего заключить в отдельные модули, описав предварительно соответствующие типы (матрица, комплексное число, кватернион, тензор и т.п.).

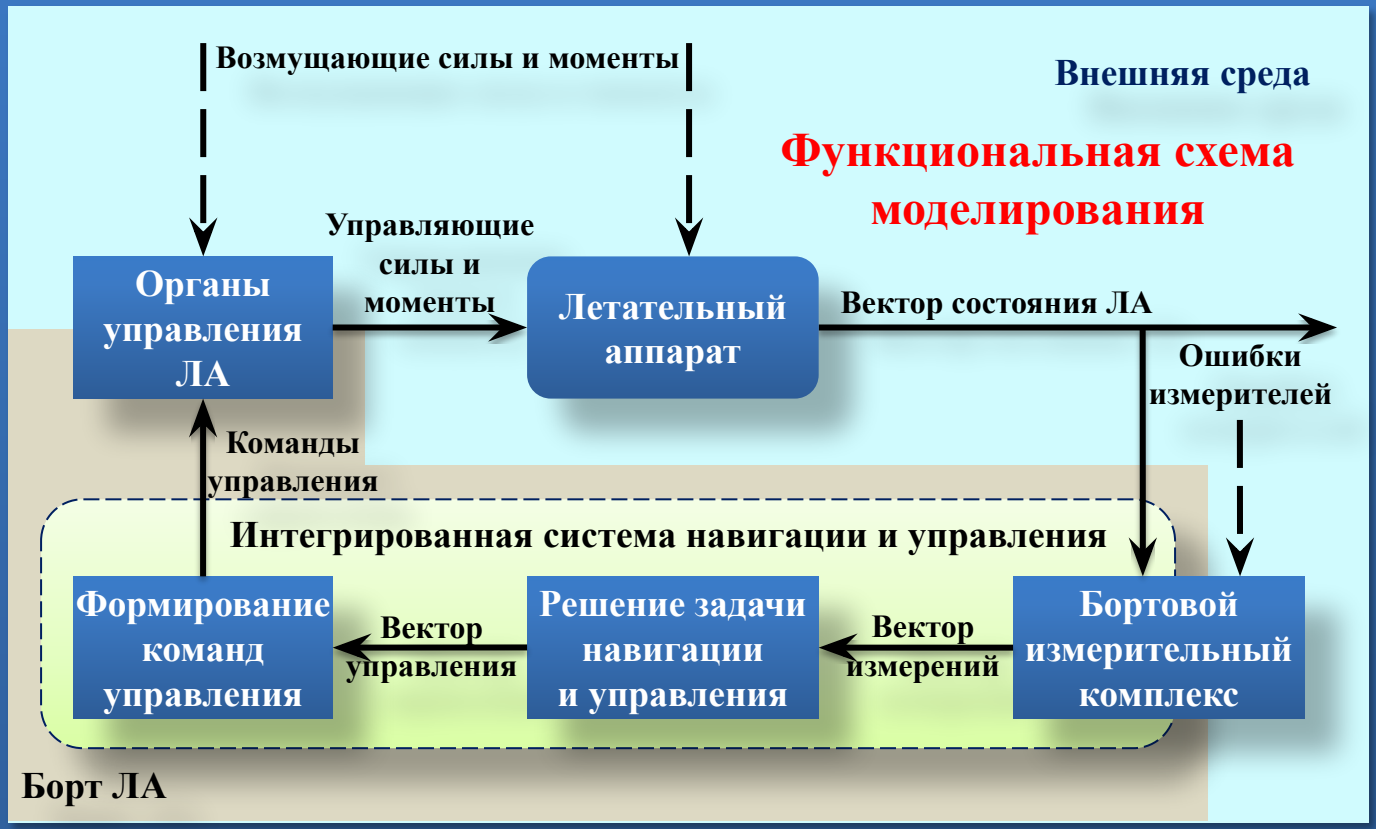
5. Если исследуемые процессы обладают вложенностью, (один процесс связан непосредственно или косвенно с несколькими другими) то в классе, реализующем данный процесс, необходимо предусмотреть соответствующее поле под объект класса, реализующего вложенный процесс.

Такие вложенные объекты должны быть созданы извне (в вызывающей программе), а в рассматриваемый класс передавался уже созданный объект.

Это необходимо чтобы разными классами использовался только один экземпляр такого класса и все обращения к его данным были бы согласованы.

Таким образом, при инициализации всей структуры сначала должны быть созданы самые независимые, простые объекты, а уже затем сложные, составные объекты.

Для построения функциональной схемы ПМО, предназначенного для математического моделирования интегрированных бортовых систем навигации и наведения высокоманевренных ЛА, необходимо составить функциональную схему моделирования, содержащую все объекты моделирования с указанием их назначения и взаимодействия с другими объектами и внешней средой. Такая функциональная схема приведена на рис



Рассмотрим подробнее каждый из элементов этой функциональной схемы с целью определения состава и функционального назначения классов, определяющих объектную структуру ПМО

ВОПРОС 1

Моделирование неуправляемого движения летательных аппаратов

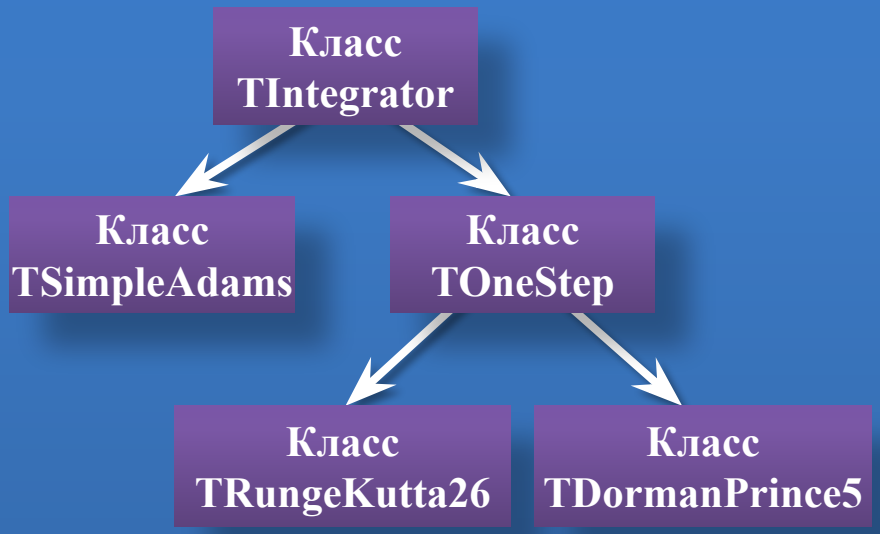
Блок "Летательный Аппарат" описывает динамику высокоманевренного беспилотного ЛА (как центра масс, так и углового движения) под воздействием сил и моментов, обусловленных влиянием внешней среды (неконтролируемых факторов: детерминированных, стохастических, неопределенных и нечетких) и отклонением управляющих органов.

Для того чтобы определить базовые классы и соответствующие цепочки классов-наследников, реализующих обсуждаемый элемент, определим необходимый состав моделей и алгоритмов, реализующих процесс моделирования динамики неуправляемого ЛА с указанием необходимых исходных данных.

Обсуждаемая проблема является задачей интегрирования системы обыкновенных дифференциальных уравнений (ОДУ) первого порядка, описанной ранее.

Таким образом, с точки зрения программной реализации данный блок состоит из двух классов – класса, реализующего численный метод интегрирования систем ОДУ, и класса, описывающего модель неуправляемого движения центра масс и углового движения ЛА.

Рассмотрим цепочку классов, реализующих библиотеку методов численного интегрирования систем обыкновенных дифференциальных уравнений.



Библиотека методов численного интегрирования

В данной библиотеке размещены следующие методы:

- метод "классический" Рунге-Кутты;
- метод вложенный Дормана-Принса;
- метод прогноза-коррекции Адамса-Мултона-Башфорта.

Все методы, кроме вложенного, используют постоянный шаг интегрирования и не используют оценку локальной погрешности на шаге.

Метод прогноза-коррекции является итерационным, причем завершение итераций определяется либо по достижению заданной точности невязки двух последних решений, либо по достижению заданного количества итераций.

Основным классом-предком для всей цепочки будет являться абстрактный класс *TIntegrator*, объединяющий в себе самые общие черты данного объектного дерева, однако содержащего лишь объявления полей и шаблоны методов. Ниже в таблицах приведены названия и описания полей и методов данного класса.

Свойства класса	Тип	Описание
Dim	целый	Размерность задачи (число компонент фазового вектора)
T0	веществен.	Начало интервала прогнозирования
Tк	веществен.	Конец интервала прогнозирования
CurT	веществен.	Текущее значение переменной интегрирования
dtRep	веществен.	Шаг вывода результатов интегрирования
RepFileName	строка	Имя файла результатов
Tol	веществен.	Локальная точность интегрирования
h	сеществен.	Шаг интегрирования
Out Put Mode	TOut Put Mode	Тип вывода (отсутствует, на каждом шаге, плотная выдача)
Out Dest	TOut Dest	Источник вывода результатов (файл или консоль)
PCurY	массив веществ, чисел	Текущий фазовый вектор
PW	массив веществ, чисел	Вектор весовых коэффициентов
OfPauseRun	логический	Флаг остановки вычислений
Owner	указатель	Ссылка на объект-владелец
CmUpdate	целый	Идентификатор сообщения

Метод	Тип	Действие
constructor Create (const anOwer: Pointer; const acmUpdate: Byte);	Конструктор класса	Создаёт объект и присваивает значения полям указателя на владельца и идентификатора сообщения
Destructor Destroy; override;	деструктор класса	Уничтожает объект
procedure InformationToOwer; virtual;	виртуальный	Передаёт управление ОС путём вызова сообщения с идентификатором сообщения
procedure SetIniData (aDim: Integer; aT0, aTk: Float; ah, ahMax: Float; aTol: Afloat; aOPMode: TOutPutMode; aOPDest: TOutDest; adtRep: Float; aY: array of Float; aW: array of Float; aRepFName: String); virtual;	виртуальный	Инициализирует начальные значения полей объекта
procedure GetIniDataFromFile (aFileName: String); virtual; abstract;	виртуальный, абстрактный	Считает исходные данные из файла с именем aFileName
procedure SetNewState (const Ti: Float; const aY: array of Float); virtual;	виртуальный	Устанавливает новые значения фазового вектора и переменной интегрирования
procedure Run(var aY: array of Float); virtual; abstract;	виртуальный, абстрактный	Выполняет численное интегрирование на всём интервале интегрирования; результат сохраняется в массиве aY
procedure RunTo(const ti: Float; var aY: array of Float); virtual; abstract;	виртуальный, абстрактный	Выполняет численное интегрирование до значения переменной интегрированияравного t_i; результат сохраняется в массиве aY
procedure Funcs(const Ti: Float; const aY: array of Float; var dY: array of Float); virtual; abstract;	виртуальный, абстрактный	Вычисляет значения вектора производных dY для значения переменной интегрирования t_i и фазового вектора aY
Procedure Report(const Ti: Float; const aY: array of Float; var dY: array of Float); virtual;	виртуальный	Выводит результаты интегрирования на момент t_i

Таким образом в приведенном абстрактном классе представлены основные "обобщенные" черты и методы всей иерархии классов, реализующих численные методы интегрирования систем ОДУ.

Из приведенных таблиц, в данном классе отсутствует непосредственно численный алгоритм (методы *Run* и *RunTo* – абстрактные).

Дальнейшим развитием базового класса является класс-потомок *TOneStep*, реализующий основные свойства группы одношаговых методов.

В таблицах приведены названия и описания новых или перекрытых по отношению к родительскому полей и методов данного класса.

Свойства класса	Тип	Описание
RejCount	целый	Счетчик отброшенных шагов
AccCount	целый	Счетчик принятых шагов
FcnCount	целый	Счетчик сделанных вызовов функции правых частей

Метод	Тип	Действие
Constructor Create(const anOwner: Pointer; const acmUpdate: Byte);	конструктор класса	Создает объект и присваивает значения полям указателя на владельца и идентификатора сообщения
procedure InitialStep; virtual;	виртуальный	Устанавливает значение начального шага интегрирования
procedure ResetCalculatlon; virtual;	виртуальный	Процедура «сброса» вычислений
Procedure SetInIData(aDIm: Integer; aTO, aTk: Float; ah, ahMax: Float; aTol: Float; aOPMode: TOutPutMode; aOPDest: TOutDest; adtRep: Float; aY: array of Float; aW: array of Float; aRepFName: String); override;	виртуальный	Инициализирует начальные значения полей объекта

Кроме того, непосредственно от класса *TIntegrator* образован класс *TSimple Adams*, реализующий простейший метод прогноза-коррекции Адамса-Мултона-Башфорта первого порядка.

В данном методе прогноз осуществляется методом Эйлера (явный метод Адамса), а коррекция – методом трапеций (неявный метод Адамса).

Одним из ключевых свойств данного класса является логическое поле *ofCorrection*, позволяющее пользователю определять, будет ли использоваться коррекция, или вычисления будут происходить только по методу Эйлера.

Часто, особенно в системах ОДУ, в правые части которых входят результаты навигационных измерений, метод прогноза-коррекции является единственным методом, не дающим эффекта "запаздывания" численного решения вследствие использования неявной коррекции.

В таблицах приведены названия и описания новых или перекрытых по отношению к родительскому полей и методов данного класса.

Свойства класса	Тип	Описание
POldY	массив веществ, чисел	Массив, содержащий фазовый вектор на предыдущей итерации
PCurF	массив веществ, чисел	Массив, содержащий значения функции правых частей на текущей итерации
POldF	массив веществ, чисел	Массив, содержащий значения функции правых частей на предыдущей итерации
ofCorrection	логический	Флаг использования коррекций
Maxlteratlon	целый	Максимальное число итераций

Метод	Тип	Действие
Constructor Create(const anOwner: Pointer; const acmUpdate: Byte);	конструктор класса	Создает объект и присваивает значения полям указателя на владельца и идентификатора сообщения
Procedure SetInIData(aDIm: Integer; aTO, aTk: Float; ah, ahMax: Float; aTol: Float; aOPMode: TOutPutMode; aOPDest: TOutDest; adtRep: Float; aY: array of Float; aW: array of Float; aRepFName: String); override;	виртуальный	Инициализирует начальные значения полей объекта
procedure Predictor; virtual;	виртуальный	Выполняет прогноз численного решения на шаге
procedure Corrector; virtual;	виртуальный	Выполняет коррекцию численного решения на шаге
procedure Step; override;	виртуальный	Организует вычисление в пределах одного шага интегрирования
procedure Run(var aY: array of Float); override;	виртуальный	Выполняет численное интегрирование на всем интервале интегрирования; результат сохраняется в массиве aY
procedure RunTo(const ti: Float; var aY: array of Float); override;	виртуальный	Выполняет численное интегрирование до значения переменной интегрирования равного t_i; результат сохраняется в массиве aY
procedure Extract (const ti: Float; var aY: array of Float); virtual;	виртуальный	Возвращает фазовый вектор для значения переменной интегрирования равного t_i; при этом t_j принадлежит текущему шагу интегрирования (интерполяция результата интегрирования)

От класса *TOneSiep* образованы два класса-потомка, реализующих современные одношаговые методы интегрирования семейства Рунге-Кутты. Наиболее простой из них – классический метод Рунге-Кутты 4 порядка с постоянным шагом интегрирования представлен классом *TRungeKutta*26.

Ниже, в таблицах приведены названия и описания новых или перекрытых по отношению к родительскому полей и методов данного класса.

Свойства класса	Тип	Описание
POldY	массив веществ, чисел	Массив, содержащий фазовый вектор на начало шага интегрирования
POldF	массив веществ, чисел	Массив, содержащий значения функции правых частей на начало шага интегрирования итерации

Метод	Тип	Действие
Constructor Create(const anOwner: Pointer; const acmUpdate: Byte);	конструктор класса	Создает объект и присваивает значения полям указателя на владельца и идентификатора сообщения
destructor Destroy; override;	деструктор класса	Уничтожает объект
Procedure SetInIData(aDIm: Integer; aTO, aTk: Float; ah, ahMax: Float; aTol: Float; aOPMode: TOutPutMode; aOPDest: TOutDest; adtRep: Float; aY: array of Float; aW: array of Float; aRepFName: String); override;	виртуальный	Инициализирует начальные значения полей объекта
procedure Step; override;	виртуальный	Организует вычисление в пределах одного шага интегрирования
procedure Run(var aY: array of Float); override;	виртуальный	Выполняет численное интегрирование на всем интервале интегрирования; результат сохраняется в массиве aY
procedure RunTo(const ti: Float; var aY: array of Float); override;	виртуальный	Выполняет численное интегрирование до значения переменной интегрирования равного t_i; результат сохраняется в массиве aY
procedure Extract (const ti: Float; var aY: array of Float); virtual;	виртуальный	Возвращает фазовый век-тор для значения перемен-ной интегрирования равного t_i; при этом t_i при-надлежит текущему шагу интегрирования (интерполяция результата интегрирования)

Класс *TDormanPrince*5 реализует современный вложенный метод численного интегрирования, позволяющий получать на одном и том же разбиении шага интегрирования два численных решения 5 и 4 порядка, используя их для вычисления локальной погрешности и определения длины нового шага интегрирования.

Кроме того, для данного метода получены так называемые непрерывные формулы, позволяющие использовать полученные решения для интерполяции решения в пределах одного шага интегрирования с 4-м порядком точности, что существенно лучше традиционной сплайн-интерполяции, используемой для других методов.

Механизм реализован в специальном методе *Densit*. Ниже, в таблице приведены названия и описания новых или перекрытых по отношению к родительскому методов данного класса.

Метод	Тип	Действие
Constructor Create(const anOwner: Pointer; const acmUpdate: Byte);	конструктор класса	Создает объект и присваивает значения полям указателя на владельца и идентификатора сообщения
destructor Destroy; override;	деструктор класса	Уничтожает объект
Procedure SetInIData(aDIm: Integer; aTO, aTk: Float; ah, ahMax: Float; aTol: Float; aOPMode: TOutPutMode; aOPDest: TOutDest; adtRep: Float; aY: array of Float; aW: array of Float; aRepFName: String); override;	виртуальный	Инициализирует начальные значения полей объекта
procedure Step; override;	виртуальный	Организует вычисление в пределах одного шага интегрирования
procedure Densit(const Ti: Float); virtual;	виртуальный	Организует процедуру интерполяции по «непрерывным» формулам в пределах одного шага интегрирования
procedure Run(var aY: array of Float); override;	виртуальный	Выполняет численное интегрирование на всем интервале интегрирования; результат сохраняется в массиве aY
procedure RunTo(const ti: Float; var aY: array of Float); override;	виртуальный	Выполняет численное интегрирование до значения переменной интегрирования равного t_i; результат сохраняется в массиве aY
procedure Extract (const ti: Float; var aY: array of Float); virtual;	виртуальный	Возвращает фазовый вектор для значения переменной интегрирования равного t_i; при этом t_i принадлежит текущему шагу интегрирования (интерполяция результата интегрирования)

Таким образом, формируется библиотека численных методов интегрирования, реализующая наиболее распространенные алгоритмы данного класса задач и позволяющая создавать классы-наследники для конкретных задач исследования динамики процессов и систем.

Формализация конкретной задачи состоит в создании потомка от одного из описанных классов с перекрытием методов *Funcs* и *Report* детализирующих процесс вычисления правых частей системы ОДУ (модели эволюции или движения системы) и вывод результатов интегрирования.

Для реализации обсуждаемого класса обратимся к основным закономерностям динамики высокоманевренного беспилотного ЛА.

Известно, что в модели движения центра масс и углового движения ЛА в правые части дифференциальных уравнений входят ускорения (продольное движение) и моменты (угловое движение), обусловленные влиянием внешней среды (возмущающими факторами) и отклонением управляющих органов.

В этой связи предварительно необходимо создать классы, описывающие изучаемый ЛА (массово-инерционные и геометрические характеристики, реакции на воздействия факторов различной природы), и модель внешней среды (модели силовых и возмущающих факторов).

Для рассматриваемого ЛА будем полагать, что все массово-инерционные и аэродинамические характеристики получены на основе экспериментальных данных и содержатся в соответствующих таблицах.

Кроме того, если известны случайные вариации этих параметров, то они могут быть использованы при моделировании неуправляемого движения ЛА совместно с номинальными значениями.

Классы, реализующие модель внешней среды:

- класс *TStatAtmo*, реализующий модель атмосферы (включая случайные вариации плотности атмосферы);
- класс *TGraviModel*, реализующий модель гравитационного потенциала Земли;
- класс *TWindModel*, реализующий модель возмущений, обусловленных влиянием ветра.

Модель атмосферы для рассматриваемого типа объектов обычно задается в виде табличных зависимостей, характеризующих эволюцию параметров атмосферы (температура, давление, плотность и скорость звука) по высоте над земной поверхностью.

В этой связи соответствующий класс *TStatAtmo* является потомком класса *TPower PolyAppro*, реализующего алгоритм полиномиальной аппроксимации табличных функций в соответствии с заданными пользователем параметрами аппроксимации (степень полинома и количество узловых точек аппроксимации).

В данном классе в зависимости от заданных параметров автоматически используется либо метод наименьших квадратов (количество узловых точек больше чем порядок полинома +1) либо классический метод построения степенных полиномов (количество узловых точек равно порядку полинома +1).

В таблицах приведены названия и описания полей и методов класса *TPower Poly Appro*.

Свойства класса	Тип	Описание
PolyPower	Целый	Степень аппроксимирующего полинома
NApproUnlts	Целый	Количество узловых точек
SourceTable	Строка	Имя ASCII-файла, содержащего аппроксимируемую таблицу
SelCols	массив целых	Массив, содержащий номера колонок в таблице, по которым будет произведена аппроксимация
ArgCol	чисел	Номер колонки в таблице, являющейся аргументом аппроксимации
UnitsShift	Целый	Признак выхода за диапазон аргумен-та (экстраполяция)

Метод	Тип	Действие
Constructor Create;	Конструктор класса	Создает объект
Destructor Destroy; override;	Деструктор класса	Уничтожает объект
Procedure Set Range (Lower, Upper: Float); virtual;	Виртуальный	Устанавливает допустимый диапазон значений аргумента
procedure Extract(Arg: Float; var Buffer: array of Float); virtual;	Виртуальный	Возвращает массив значений аппроксимируемых функций Buffer для текущего значения аргумента Arg

Для класса *TStatAtmo*, потомка класса *TPowerPolyAppro*, введено дополнительное поле *AtmoInfo*, содержащее параметры атмосферы, и метод *Get*, возвращающий текущее значение параметров атмосферы для заданной высоты полета ЛА.

```
type
  TAtmoInfo = record
    Te,       {Temperature, K}
    Ro,       {Ballistic Density, kg/m^3}
    P,        {Atmospheric Pressure, kg/(m*s^2)}
    a: Float; {Acoustic Velocity, m/s}
  end;

  {Atmospheric Parameters Approximation}
  TStatAtmo = class(TPowerPolyAppro)
    constructor Create(aFileName: String);
  protected
    FAtmoInfo: TAtmoInfo;
  public
    procedure Get(Alt: Real); virtual;
   property AtmoInfo: TAtmoInfo read FAtmoInfo;
  end;
```

Для учета случайных вариаций параметров атмосферы используется аналогичный описанному класс *TStatAtmoDeriv*, основной особенностью которого является использование такого же ASCII файла, содержащего эволюцию дисперсий случайных вариаций параметров атмосферы по высоте.

На основе аппроксимированных значений для данной высоты реализуется соответствующая случайная комбинация вариаций параметров атмосферы, учитываемая совместно с номинальными значениями.

При моделировании процессов функционирования интегрированной бортовой системы навигации и наведения беспилотного высокоманевренного ЛА на разных этапах могут использоваться несколько моделей гравитационного поля Земли, отличающиеся допущениями относительно формы и распределения масс в теле Земли.

В этой связи в ПМО реализована иерархическая цепочка классов, реализующая необходимые при моделировании модели геопотенциала.

Базовым классом в данной иерархии является абстрактный класс TGraviModel, содержащий только лишь объявление единственного абстрактного метода Extract, возвращающего значения компонент ускорения, обусловленного гравитационным притяжением Земли в зависимости от текущих координат точки.

```
TGraviModel = class
    function Extract(const Loc: TLoc): TLoc; virtual; abstract;
end;
```

От базового класса образовано два класса-потомка: *TSimpieGravi* и *TIERSGravi*

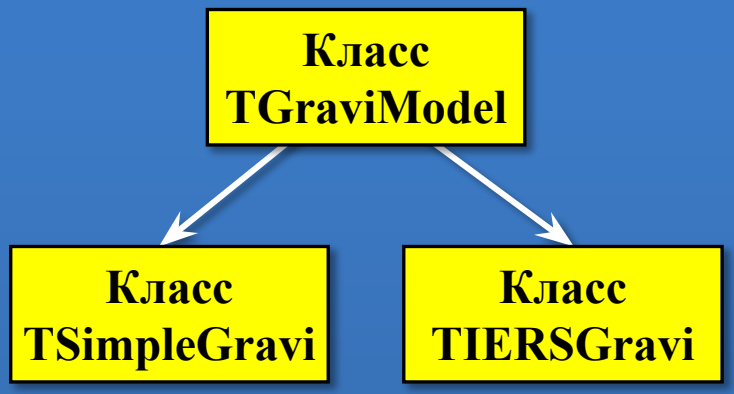


Схема преобразования базового класса в классы-потомки

Первый из этих классов реализует простейшую модель расчета ускорений, обусловленных притяжением Земли, основанную на представлении о сферической Земле и центральном сферическом геопотенциале.

```
TSimpleGravi = class(TGraviModel)
    constructor Create (aGO, aRO: Float);
  protected
    Go,
    Ro: Float;
  public
    function Extract(const Loc: TLoc): TLoc; override;
    property GrAccSurf: Float read Go write Go;
    property RadSurf: Float read Ro write Ro;
  end;
```

Класс *TIERSGravi* реализует современную модель представления геопотенциала, принятую International Earth Rotation Services (IERS) и основанную на разложении геопотенциала в ряд по сферическим функциям до 70 степени и порядка включительно.

Необходимо отметить, что в данном классе реализован алгоритм, предложенный Каннингхемом и позволяющий вычислять компоненты ускорения в декартовой системе координат, что существенно сокращает вычислительные ресурсы по сравнению с традиционными алгоритмами, определяющими проекции ускорения в сферической системе координат.

В таблицах приведены названия и описания полей и методов класса *TIERSGravi.*

Свойства класса	Тип	Описание
Degree	целый	Степень и порядок разложения геопотенциала
FIleName	строка	Имя ASCII-файла, содержащего коэффициенты гармоник геопотенциала (модель Земли)

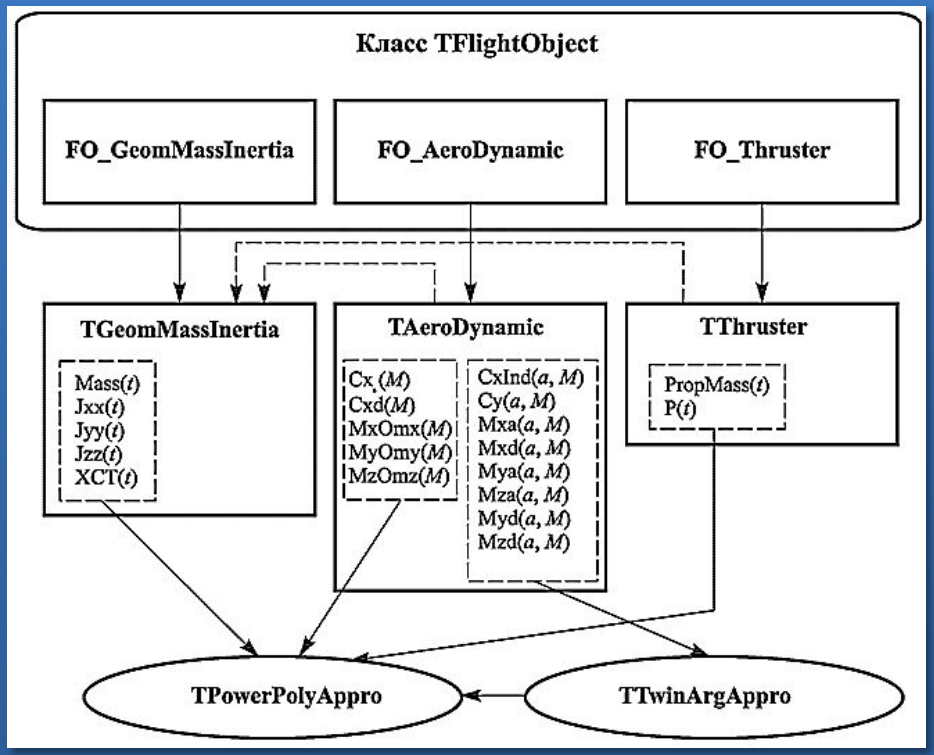
Метод	Тип	Действие
Constructor Create;	конструктор класса	Создает объект
Destructor Destroy; override;	конструктор класса	Уничтожает объект
procedure LoadFrom(const aFName: String); virtual;	виртуальный	Загружает набор коэффициентов гармоник геопотенциала из файла с именем aFName
function Extract (const Loc: TLoc): TLoc; override;	виртуальный	Возвращает запись, содержащую значения ускорения в проекциях на оси земной связанной СК для точки с координатами в земной связанной СК, содержащимися в записи Loc

Для беспилотных высокоманевренных ЛА одним из основных возмущающих факторов внешней среды является воздействие ветра. Данное возмущение формализуется в моделировании вектора скорости ветра и последующего его учета при расчете воздушной скорости ЛА и его основных аэродинамических характеристик. Модель возмущений, обусловленных влиянием ветра, реализована в классе *TWindModel*.

```
TWindModel = class
    constructor Create(aIniFName: String);
    destructor Destroy; override;
  protected
   FEta, FPCW: Float;
   FMean, FRMS: Float;
   FVWind: TLoc;
   kb0: array[0..5] of Float;
public
   procedure GetWind(CurAlt: Float; CurEA: TLoc; BFtoIF: Tensor);
   property Eta: Float read FEta;
   property PCW: Float read FPCW;
   property VWind: TLoc read FVWind;
end;
```

Основным свойством данного класса является свойство *VWind*, вычисляемое методом *Get Wind* для текущей высоты ЛА *Cur Alt* в связанной системе координат. Изначально вектор скорости ветра определяется в земной связанной СК, а затем пересчитывается в связанную с ЛА СК с использованием матрицы перехода *BFToIF*.

Реализовав объектную структуру модели внешней среды, перейдем к формированию классов, описывающих изучаемый ЛА (массово-инерционные и геометрические характеристики, реакции на воздействия факторов различной природы). На рисунке приведена объектная структура непосредственно модели ЛА как неуправляемого материального объекта.



Класс *TFlightObject*, реализующий модель летательного аппарата, функционально содержит в себе три объекта:

- ***FO_GeomMassInertia*, являющийся переменной класса *TGeomM ass Inertia* и содержащий свойства и методы, описывающие эволюцию массово-инерционных и геометрических характеристик по времени полета ЛА;**
- ***FO_AeroDynamic*, являющийся переменной класса *TAeroDynamic* и реализующую модель учета аэродинамических воздействий внешней среды на ЛА (расчет аэродинамических сил и моментов);**
- ***FO_Thruster*, являющийся переменной класса *TThruster* и реализующий модель учета реактивных воздействий ДУ на ЛА (расчет реактивных сил и моментов).**

Класс *TGeomMassInertia* содержит внутренние объекты, являющиеся переменными от класса *TPowerPolyAppro* и предназначенные для аппроксимации данных об эволюции массово-инерциальных и геометрических характеристиках ЛА по времени полета ЛА.

Текущая информация сохраняется в поле *GMIInfo*, представляющее собой запись следующего вида:

```
TGMIInfo = record
    CrntTime: Float;                          {текущее время полета ЛА}
    Mass: Float;                              {текущая полная масса ЛА}
    Jxx, Jyy, Jzz: Float;          {текущие центральные моменты инерции ЛА}
    Jxy, Jxz, Jyz: Float;        {текущие перекрестные моменты инерции ЛА}}
    XCT: Float;                  {текущее положение ц.м. ЛА в связанной СК}

    LMissile: Float;                                          {длина ЛА}
    LWing: Float;                                        {размах крыла ЛА}
    Ba: Float;                              {характерный размер ЛА (САХ)}
    Middle: Float;                      {площадь миделевого сечения ЛА}
    SWing: Float;                                      {площадь крыла ЛА}
  end;
```

Тогда объявление класса *TGeomMassInertia* выглядит следующим образом:

```
TGeomMassInertia = class(TObject)
    constructor Create(const aIniFileName: String);
    destructor Destroy; override;

  protected
    FMass: TPowerPolyAppro;
    FJxx: TPowerPolyAppro;
    FJyy: TPowerPolyAppro;
    FJzz: TPowerPolyAppro;
    FXCT: TPowerPolyAppro;
    FGMIInfo: TGMIInfo;
  public
    procedure GetInfo(const ti: Float; const Y: array of Float); virtual;
    property GMIInfo: TGMIInfo read FGMIInfo;
  end;
```

Метод *GetInfo* вычисляет для текущего момента времени t_i текущие массовоинерционные и геометрические характеристики ЛА и сохраняет их в поле *GMIInfo*.

Для общности в состав формальных параметров метода введен произвольный вектор *Y*, который может быть использован при реализации классов-потомков.

Необходимо отметить, что все рассматриваемые в данном ПМО классы получают исходные данные из файла, представляющего собой стандартный Windows-ini файл, основной особенностью которого является структуризации записей в следующем формате:

```
[Название секции]
ИМЯ ПЕРЕМЕННОЙ 1 = ЗНАЧЕНИЕ
ИМЯ ПЕРЕМЕННОЙ 2 = ЗНАЧЕНИЕ
```

С использованием такого механизма удается избежать жесткой связи объектной структуры приложения, реализующего непосредственно алгоритмическую часть задачи, с интерфейсом пользователя, поскольку файл исходных данных является в этом случае промежуточным буфером.

Каждый объект, которому необходимо проинициализировать свои исходные данные, обращается к единому ini-файлу и считывает значения необходимых переменных позиционируя соответствующую секцию файла. Так, для класса *TGeomMassInertia* существует секция [*GeomMassInertia*].

```
[GeomMassInertia]
LMissile = 2.9
LWing = 0.5
Ba = 0.69
Middle = 0.0228
Swing = 0.233
Mass = c:\dss\data\mass.dat
Jxx = c:\dss\data\jxx.dat
Jyy = c:\dss\data\jyy.dat
Jzz = c:\dss\data\jzz.dat
XCT = c:\dss\data\xct.dat
```

Класс *TAeroDynamic* содержит внутренние объекты, являющиеся переменными от класса *TPowerPolyAppro* и *TTwinArgPoly* и предназначенные для аппроксимации данных об изменении коэффициентов аэродинамических сил и моментов.

Причем, если коэффициент зависит только от одного параметра (например числа Маха M), то соответствующий объект является переменной от *TPowerPolyAppro*, если же зависимость двухпараметрическая (например от числа Маха M и угла атаки), то соответствующий объект является переменной от *TTwinArgAppro*.

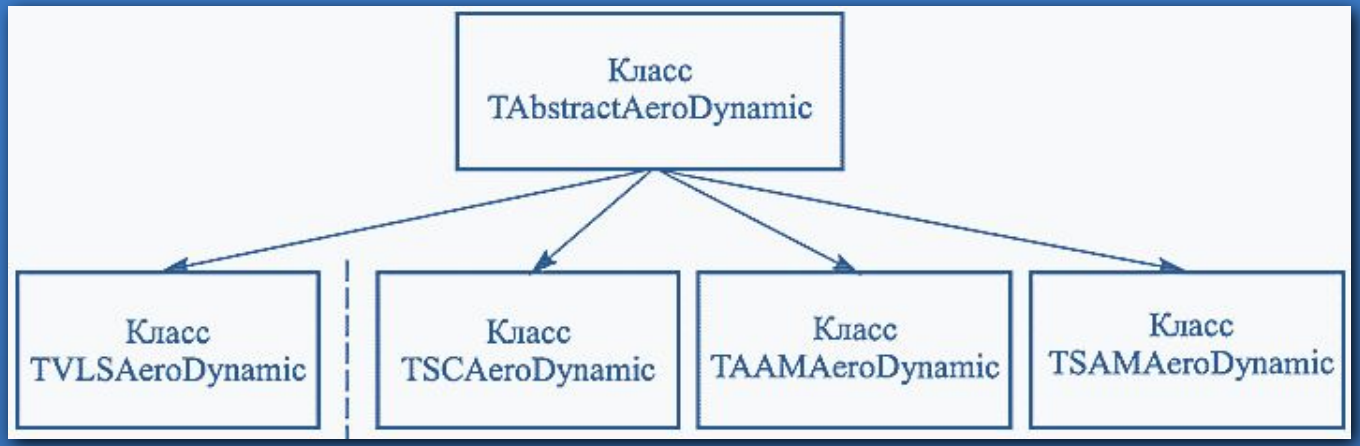
Класс *TTwinArgAppro* является потомком от класса *TPowerPolyAppro* и реализует алгоритм линейной аппроксимации данных по рядам таблицы, полученных как результат полиномиальной аппроксимации по столбцам исходной таблицы.

Первая строка и столбец таблицы трактуются как значения аргументов аппроксимации.

Методы и поля данного класса перекрыты по сравнению с предком и имеют тот же тип и список формальных параметров.

Единственная особенность работы с этим классом состоит в том, что результат аппроксимации всегда скалярный.

Для реализации универсальной модели аэродинамики ЛА создана цепочка наследования классов, отображенная на рисунке.



Объектная модель аэродинамики ЛА

Базовым классом для данной цепочки является класс *TAbstractAeroDynamic* объединяющий в себе самые общие черты данного объектного дерева, однако содержащего лишь объявления полей и шаблоны методов.

Для детализации конкретной модели аэродинамики того или иного аппарата должен быть создан класс-наследник с перекрытыми методами вычисления аэродинамических сил и моментов.

Так на схеме приведены дочерние классы, реализующие аэродинамику ракет-носителей (*TVLSAeroDynamic*), ИСЗ (*TSCAeroDynamic*), ракет воздух-воздух (TAAMAeroDynamic), зенитных ракет (*TSAMAeroDynamic*) и т. п.

Принцип полиморфизма позволяет использовать в классе-владельце (*TFlightObject*) ссылку на родительский тип иерархии (TAbstractAeroDynamic), обеспечивая тем не менее вызов необходимого метода реального объекта, инициализированного при создании объекта *FlightObject*.

Для класса *TAeroDynamic* в ini-файле комплекса существует секция [*AeroDynamic*].

```
[AeroDynamic]
CxOFileName=c:\dss\cx0.dat
CxdFileName= c:\dss\cxd.dat
CxIndFileName= c:\dss\cxind.dat
CyFileName= c:\dss\cy.dat
MxaFileName= c:\dss\mxa.dat
MxadeltaFileName= c:\dss\mxad.dat
MxOmxFileName= c:\dss\mxomx.dat
```

Класс *TThruster* предназначен для вычисления реактивных сил и моментов, обусловленных тягой двигательной установки ЛА, и содержит внутренние объекты, являющиеся переменными от класса *TPowerPolyAppro* и предназначенные для аппроксимации данных об изменении тяги двигательной установки и массы топлива в течение полета ЛА.

В таблицах приведены названия и описания полей и методов данного класса.

Свойства класса	Тип	Описание
CurTIme	веществен.	Текущее время полета Л А, на которое вычислены силы и моменты
Force	запись	Компоненты вектора реактивной силы в связанной СК
Torque	запись	Компоненты вектора реактивного момента в связанной СК
StatAtmo	TStatAtmo	Ссылка на объект – модель атмосферы
P	TPowerPolyAppro	Текущее значение тяги ДУ
PropMass	TPowerPolyAppro	Текущее значение массы топлива ДУ

Метод	Тип	Действие
constructor Create (const alniFName: String; aStatlcAtmo: TStatlcAtmo);	конструктор класса	Создает объект, инициализирует внутренние переменные и присваивает внутренней переменной ссылку на существующий внешний объект aStatAtmo – модель атмосферы
Destructor Destroy; override;	деструктор класса	Уничтожает объект
procedure GetFM(ti: Float; aAlt: Float; DeltaGY, DeltaGZ: Float; aGMIInfo: TGMIInfo); virtual;	виртуальный	Вычисляет значения реактивных сил и моментов при значениях для текущего момента времени t_i, высоты aAlt и углах отклонения газодинамических рулей DeltaGY и DeltaGZ. Результат сохраняется в полях Force и Torque.

Для класса *TThruster* в ini-файле комплекса существует секция [Thruster].

```
[Thruster]
PFileName=c:\dss\p.dat
PropMass=c:\dss\pm.dat
Ta=3.5
Tk=4.5
```

Как видно из приведенной объектной структуры, для классов *TAeroDynamic* и *TThruster* в метод *Create*, создающий соответствующие объекты, должен быть передан уже существующий объект типа *TStaticAtmo* для последующего обращения к нему в целях получения текущих параметров атмосферы.

Класс *TAeroDynamic* имеет помимо этого внутреннюю переменную *WindModel*, куда должна быть передана соответствующая ссылка на реально существующий объект типа *TWindModel*.

Кроме того, в метод *GetFM* обсуждаемых классов должна быть передана запись типа *TGMIInfo*, содержащая текущие геометрические и массово-инерционные параметры ЛА.

Приведем описание полей и методов класса *TFlightObject*, реализующего модель изучаемого ЛА, как неуправляемого материального объекта.

Для класса *TFlightObeject* в ini-файле комплекса существует секция [*Initial Conditions*], задающая начальные условия для параметров движения объекта.

```
[Initial Conditions]
IniAlt=8000
IniY=0
IniZ=0
IniZ=0
IniVx=300
IniVy=0
IniVz=0
Iniwx=0
Iniwy=0
Iniwz=0
IniPitch=0
IniYaw=0
IniRoll=-45
```

Таким образом, сформировав модель внешней среды и модель неуправляемого ЛА (т. е. методику расчета ускорений и моментов), перейдем к классу, реализующему динамику ЛА.

Динамика ЛА определяется в результате решения системы обыкновенных дифференциальных уравнений (ОДУ) первого порядка, которую условно принято разделять на две части: уравнения динамики центра масс ЛА (в традиционной терминологии – "медленное" движение), представляющие собой векторную запись второго закона Ньютона, и уравнения углового движения ЛА ("быстрое" движение), представляющие собой векторную запись уравнений Эйлера для жесткого тела.

В практике исследований по динамике маневренных ЛА часто разделяют эту систему на "быструю" и "медленную" части с точки зрения процесса интегрирования ОДУ.

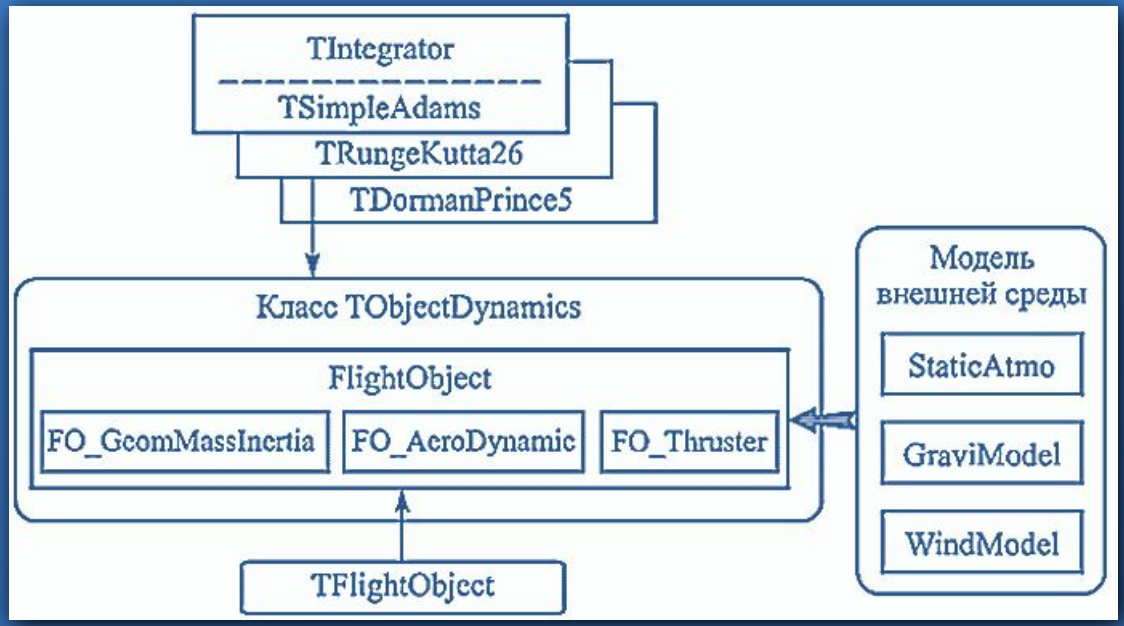
"Быстрая" часть уравнений интегрируется в отдельном блоке с меньшим шагом, результаты передаются в другой блок, интегрирующий "медленные уравнения с большим шагом".

Этот подход, несмотря на явное предпочтение с точки зрения быстродействия программного кода, имеет существенный недостаток: обе подсистемы являются зависимыми друг от друга, и такое разделение может привести к неустойчивости получаемого решения и дополнительной алгоритмической ошибке.

В этой связи в рамках излагаемой технологии интегрируется полная система уравнений, включающая в единый вектор состояния как параметры движения центра масс (компоненты положения и скорости ЛА), так и параметры углового движения объекта (угловые скорости в связанной СК, параметры Родрига-Гамильтона, или другие параметры ориентации ЛА: углы Эйлера, матрица Пуассона и т. п.).

Тот факт, что различные уравнения в этой расширенной системе должны интегрироваться с различной точностью, находит отражение в масштабировании вычисляемой локальной ошибки на шаге в соответствии с т. н. вектором масштабных коэффициентов.

Компоненты вектора подобраны таким образом, чтобы обеспечить лучшую необходимую точность вычислений для компонент вектора состояния, соответствующих "быстрому" движению. Как видно из рисунку класс *TObjectDynamics* является наследником одного из классов, реализующих численный метод интегрирования систем ОДУ (*TSimpleAdams*, *TRungeKutta26*, *TDormanPrince5*).



С точки зрения обсуждаемого класса инвариантно, какой из классов-интеграторов будет предком, поскольку принципиальное отличие *TObjectDynamics* от класса-родителя состоит в переопределении метода *Funcs* – т. е. реализации правых частей системы ОДУ.

Таким образом, решение, кто из классов-интеграторов будет предком, принимает исследователь, исходя из общей постановки задачи, требуемой точности решения, особенности бортовой реализации алгоритмов и т. п.

Кроме того данный класс содержит внутреннее поле типа *TFlightObject* и указывает на реально существующий объект данного типа.

Каждый раз внутри метода Funcs и после завершения шага интегрирования вызывается метод *TFlightObject. SetStatus* для обновления информации во внутренних полях *FlightObject*.

Приведем описание полей и методов класса *TObjectDynamics*, реализующего динамику изучаемого ЛА.

Свойства класса	Тип	Описание
Name	строка	Название объекта
InlFName	строка	имя ini-файла
Dim	целый	размерность задачи
CuirentTIme	веществен.	Текущее время полета ЛА
CurrentState	Tvector	текущий расширенный вектор состояния системы

Метод	Тип	Действие
constructor Create (const anOwner: Pointer; const acmUpdate: Byte; aFHghtObjetc: TFHghtObject);	конструктор класса	Создает объект, инициализирует внутренние переменные, передает существующий объект типа TFlightObject классу.
procedure Funcs(const Ti: Float; const aY: array of Float; var dY: array of Float);	виртуальный	Расчет правых частей системы ОДУ
procedure Report (const Ti: Float; const aY: array of Float; var dY: array of Float);	виртуальный	Вывод результатов интегрирования
procedure GetlniDataFrom-File(aFileName: String);	виртуальный	Загружает исходные данные из ini-файла

Для данного класса в ini-файле комплекса существует секция [Integrator], задающая исходные данные для интегрирования.

```
[Integrator]
WLoc = l
WVel = 10
WOmega = 100
WQuat = 10
T0 = 0
Tk = 300
Iolerance = le – 8
H=0.0005
NOutStep = 20
OutPutMode = 2
OutPutDestination = 1
```

Описав таким образом объектную модель неуправляемого движения ЛА, обратимся вновь к функциональной схеме моделирования.

Следующим звеном в контуре моделирования является блок бортового измерительного комплекса (БИК), непосредственно связанный с блоком "Летательный аппарат".

ВОПРОС 2

Моделирование бортового измерительного комплекса

Бортовой измерительный комплекс маневренного ЛА включает в себя, как правило, блок чувствительных элементов ориентации в пространстве (инерциальные указатели направлений – гиростабилизированная платформа на гироскопах, блок датчиков угловых скоростей), блок инерциальных измерителей (акселерометры), высотомеры, датчики угла атаки и т. п.

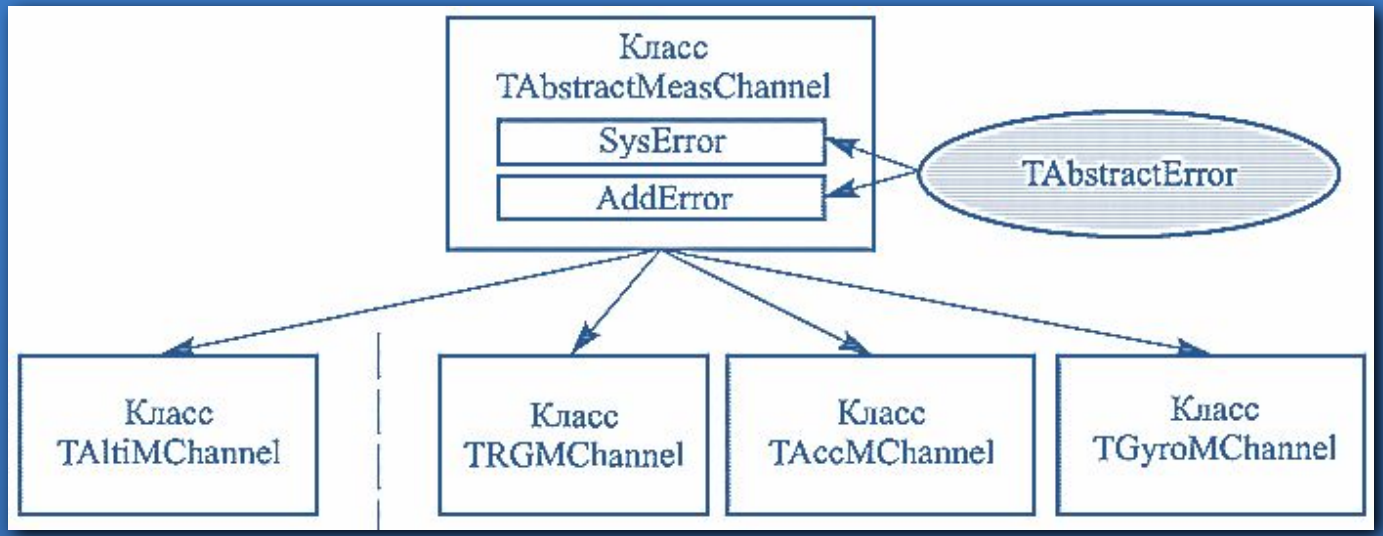
Состав конкретной аппаратуры определяется целевой функцией ЛА и алгоритмом интегрированной системы навигации и управления.

Так или иначе, в рамках излагаемой объектно-ориентированной технологии моделирования каждый из совокупности измерителей БИК будем рассматривать как функциональный элемент, на вход которого поступает расширенный вектор состояния динамической системы, а на выходе формируется фактическое значение вектора измерений, прямым или косвенным образом связанного с компонентами вектора состояния.

Одной из отличительных черт этого элемента является учет ошибок измерений различной природы.

Алгоритм учета ошибок в предлагаемой объектной структуре выделим группу наиболее часто используемых моделей стохастических случайных факторов, которые представляются в виде случайных величин (систематические ошибки) и случайных процессов (случайные аддитивные ошибки), предусмотрев соответствующие поля и алгоритмы их инициализации.

Рассмотрим цепочку классов, реализующих библиотеку моделей измерительных устройств БИК

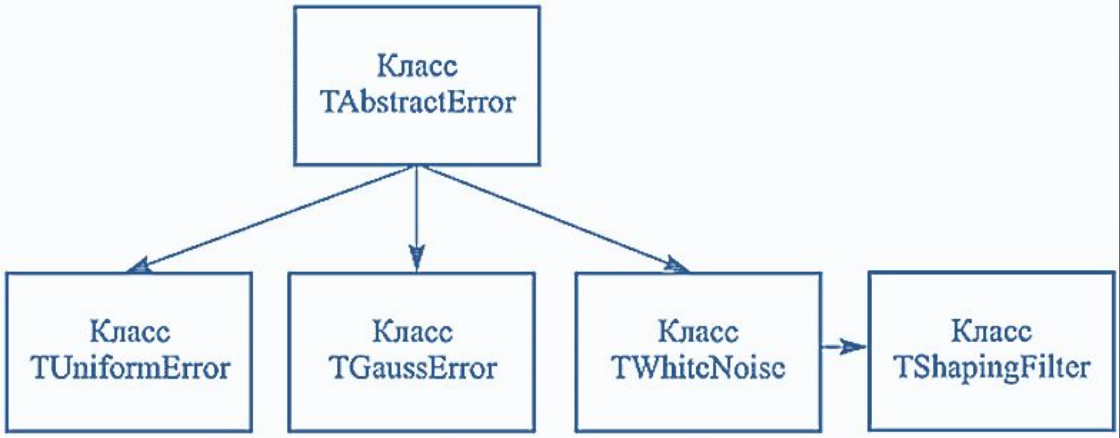


Все измерительные устройства имеют общего абстрактного предка – класс *TAbstractMeasChannel*, объединяющего самые общие родовые черты и методы рассматриваемой группы.

Одним из таких важнейших свойств являются поля *SysError* и *AddError*, представляющие собой объекты общего абстрактного класса *TAbstractError*.

Очевидно, что в реальных классах (*TRGMChannel*, *TAccMChannel* и т. п.) будут использоваться конкретные объекты, реализующие расчет этих ошибок из цепочки классов, вершиной которой является класс *TAbsiractError* – это обеспечивает базовый принцип ООП – полиморфизм.

В этой связи, прежде чем привести описание классов, реализующих модели измерителей, обратимся к цепочке классов, описывающих модели ошибок и возмущающих факторов



Основным классом-предком для всей цепочки будет абстрактный класс *TAbstractorError*, объединяющий в себе самые общие черты данного объектного дерева, содержащего, однако, лишь объявления полей и шаблоны методов.

Свойства класса	Тип	Описание
FErrValue	TVector	вектор ошибок
Dim	целый	размерность вектора ошибок

Метод	Тип	Действие
constructor Create(aDIm: Integer);	конструктор класса	Создает объект, инициализирует внутренние переменные
destructor Destroy; override;	деструктор класса	Уничтожает объект
procedure SetDIm(Value: Integer); virtual;	виртуальный	Устанавливает новое значение размерности вектора ошибок, перераспределяет память
procedure Get MaxErr (const ti: Float; const aState: array of Float); virtual; abstract;	виртуальный, абстрактный	Вычисляет максимальное (наихудшее) для данного распределения значение вектора ошибок
procedure Get Err Value (const ti: Float; const aState: array of Float); virtual; abstract;	виртуальный, абстрактный	Вычисляет текущее значение вектора ошибок

Как видно из приведенных таблиц, в данном классе отсутствует непосредственно алгоритм вычисления вектора ошибок (методы *GetMaxError* и *GetErrorValue* – абстрактные).

Основным свойством данного класса является вектор ошибок *FErrValue* размерности *Dim*.

Процедуры расчета вектора ошибок *GetMaxErr* и *GetErrValue* производят вычисление вектора ошибок в соответствии с текущим временем t_i и некоторым вектором *aState* (вектор параметров, используемый при необходимости) и сохраняют вычисленные значения в поле *FErrValue*.

Дальнейшим развитием базового класса является класс-потомок *TUniformError*, реализующий вычисление вектора ошибок в соответствии с равномерным распределением и класс *TGaussError*, реализующий вычисление вектора ошибок в соответствии с нормальным распределением.

В таблицах приведены названия и описания новых или перекрытых по отношению к родительскому полей и методов данных классов соответственно.

Свойства класса	Тип	Описание
FErrLow	TVector	вектор нижних значений интервала распределения
FEirUpp	TVector	вектор верхних значений интервала распределения

Метод	Тип	Действие
constructor Create(aDIm: Integer; const aLow, aUpp: array of Float);	конструктор класса	Создает объект, задает нижнюю и верхнюю границу для равномерного распределения
destructor Destroy; override;	деструктор класса	Уничтожает объект
procedure SetDIm(Value: Integer); virtual;	виртуальный	Устанавливает новое значение размерности вектора ошибок, пере-распределяет память
procedure GetMaxErr(const ti: Float; const aState: array of Float); override;	виртуальный	Вычисляет максимальное (наихудшее) для равномерного (концы интервала) распределения значение вектора ошибок
procedure GetErrValue(const ti: Float; const aState: array of Float); override;	виртуальный,	Вычисляет текущее значение вектора ошибок

Свойства класса	Тип	Описание
FErrMean	TVector	вектор математического ожидания
FErrCovar	TVector	ковариационная матрица

Метод	Тип	Действие
constructor Create(aDIm: Integer; const aMean, aCovar: array of Float);	конструктор класса	Создает объект, задает вектор математического ожидания и ковариационную матрицу для нормального распределения
destructor Destroy; override;	деструктор класса	Уничтожает объект
procedure SetDIm(Value: Integer); virtual;	виртуальный	Устанавливает новое значение размерности вектора ошибок, перераспределяет память
procedure GetMaxErr(const ti: Float; const aState: array of Float); override;	виртуальный	Вычисляет максимальное (наихудшее) для равномерного (концы интервала) распределения значение вектора ошибок
procedure GetErrValue(const ti: Float; const aState: array of Float); override;	виртуальный,	Вычисляет текущее значение вектора ошибок

Кроме описанных классов от *TAbstractError* создан класс *TWhiteNoise*, моделирующий значение векторного белого шума.

Ниже, в таблицах приведены названия и описания новых или перекрытых по отношению к родительскому полей и методов данного класса.

Свойства класса	Тип	Описание
WNRMS	TVector	интенсивность белого шума

Метод	Тип	Действие
constructor Create(aRMS: TVector, aFreq, adT: Float);	конструктор класса	Создает объект, задает интенсивность и интервал белого шума
destructor Destroy; override;	деструктор класса	Уничтожает объект
procedure GetErrValue(const ti: Float; const aState: array of Float); override;	виртуальный,	Вычисляет текущее значение вектора ошибок

Класс *TShapingFilter*, являющийся потомком *TWhiteNoise*, реализует случайный процесс, заданный формирующим фильтром первого порядка, на вход которого подается белый шум.

По отношению к классу-предку в *TShapingFilter* существует внутренний объект типа *TSimpleAdams*, интегрирующий дифференциальное уравнение формирующего фильтра.

Вызов метода *RunTo* данного объекта производится в методе *GetErrValue* до текущего момента t_i.

В таблицах приведены названия и описания новых или перекрытых по отношению к родительскому полей и методов данного класса.

Свойства класса	Тип	Описание
FtCor	веществ.	интервал корреляции случайного процесса
SPRMS	TVector	интенсивность случайного процесса
FSPReallzatton	TSImpleAdams	внутренний интегратор
FCurTIme	веществ.	текущее время

Метод	Тип	Действие
constructor Create (const anOwner: Pointer; const acmUpdate: Byte; atCor: Float; aRMS: TVector);	конструктор класса	Создает объект, задает интенсивность и интервал корреляции случайного процесса
destructor Destroy; override;	деструктор класса	Уничтожает объект
procedure GetlnlDataFrom-File(aFIleName: String);	виртуальный	Загружает данные из ini-файла
procedure GetErrValue(const ti: Float; const aState: array of Float); override;	виртуальный	Вычисляет текущее значение вектора ошибок

Как видно из приведенных таблиц, основным свойством данного класса является поле *FMeas*, содержащее значения вектора измерений.

Обсуждаемый класс имеет ссылку на объект, реализующий динамику ЛА (*TObjectDynamics*), который передается классу при создании.

Кроме того, в классе определены внутренние объекты *SysError* и *AddError* типа *TAbstractErr* or для последующего определения в классах-потомках.

Основной метод класса – *GetMeas*, вычисляющий вектор измерений на момент времени t_i и в соответствии с некотором вектором параметров *aState*, в данном классе объявлен абстрактным с целью последующего перекрытия в классах-потомках.

Реализацию конкретного измерителя продемонстрируем на примере класса *TRGMrChannel*, формализующего модель датчика угловой скорости.

Ниже, в таблицах приведены названия и описания новых или перекрытых по отношению к родительскому полей и методов данного класса.

Свойства класса	Тип	Описание
RealRG	запись	содержит значения «истинных» угловых скоростей ЛА в связанной СК, извлекаемых из объекта *CurObjectDynamics*

Метод	Тип	Действие
constructor Create(aInIFName: String; aCurObjectDynamlcs: TObjectDynamlcs);	конструктор класса	Создает объект, инициализирует внутренние переменные.
destructor Destroy; override;	деструктор класса	Уничтожает объект
procedure GetMeas(const ti: Float; var aState: array of Float); override;	виртуальный	Вычисляет значение вектора измеренной угловой скорости ЛА в связанной СК для момента времени t_i

В методе *Create* данного класса инициализируются объекты для учета систематических и случайных аддитивных ошибок измерений угловых скоростей как случайные гауссовские вектора, причем значение систематической ошибки остается постоянным в течение всего времени функционирования объекта.

Исходные данные для инициализации параметров измерителя считываются из ini-файла проекта в секции [*RateGyro*].

```
[Rate Gyro]
SysMeanX = 0
SysMeanY = 0
SysMeanZ = 0
SysRMSX = 0.03
SysRMSY = 0.03
SysRMSZ = 0.03
AddtCor = 120
AddRMS = 0.01
IsSysErr = 1
IsAddErr = 1
```

"Истинные" значения угловой скорости ЛА запрашиваются в методе *GetMeas* на текущий момент времени t_i у объекта *CurObjectDynamics*.

В соответствии с выставленными флагами учета ошибок *IsSysErr* и *IsAddErr* происходит зашумление "истинных" значений измеряемых величин и результат сохраняется в поле *FMeas*.

С точки зрения процесса моделирования к группе измерителей также относится блок многоканального приемника СНС ГЛОНАСС/GPS, непосредственно осуществляющий измерения псевдодальности, псевдоскорости и фазы навигационного сигнала (блок предварительной обработки).

Необходимо отметить, что описание процесса моделирования данных измерений выходит за рамки материала книги вследствие большого объема и сложности используемых моделей (движение навигационных КА, уход часов приемника, тропосферная и ионосферная задержка и т.п.).

Сформированная в БИК измерительная информация является основой для решения прицельно-навигационной задачи, решение которой в рамках представленной принципиальной схемы моделирования осуществляется в интегрированном блоке навигации и управления.

Рассмотрим ниже объектную модель реализации данного звена.

ВОПРОС 3

Моделирование блока навигации и управления

Как уже отмечалось выше, бортовая интегрированная система управления и навигации беспилотного маневренного ЛА включает, как правило, бескарданную инерциальную навигационную систему (БИНС) и многоканальный ГЛОНАСС/GPS приемник.

Решение задачи навигации и определения ориентации ЛА осуществляется навигационным контуром путем комплексирования выходных данных этих устройств, при котором в соответствии с той или иной схемой осуществляется коррекция как навигационного решения, так и параметров БИНС.

Основой для любой модели навигационной системы является класс *TAbstractNavi*, представляющий собой абстрактного предка для всех классов, реализующих конкретную систему

Модели используемых в интегрированном бортовом комплексе навигационных систем отличаются составом используемой измерительной информации, составом вектора оцениваемых параметров и принципом функционирования (так, например БИНС интегрирует показания акселерометров и ДУС, формируя оценку положения, скорости и ориентации ЛА, а при использовании ГЛОНАСС/GPS приемника измеренные псевдодальности и псевдоскорости обрабатываются, как правило, методом динамической фильтрации, в результате чего оцениваются положение и скорость ЛА).

Тем не менее, класс *TAbstractNavi* содержит объявления свойств и шаблоны абстрактных методов, общие для всей цепочки классов, реализующих модели навигационных систем

Свойства класса	Тип	Описание
Dim	целый	размерность вектора оцениваемых параметров
NameNS	строка	название навигационной подсистемы
FEstlm	TVector	вектор оцениваемых параметров
CurObject Dynamics	TObjectDynamics	ссылка на объект, реализующий динамику ЛА
NSCTIme	веществ.	текущее время

Метод	Тип	Действие
constructor Create(aDIm: Integer; aNameNS: String; aCurObjectDynamlcs: TObjectDynamics);	конструктор класса	Создает объект, инициализирует внутренние переменные.
destructor Destroy; override;	деструктор класса	Уничтожает объект
procedure GetEstim(const ti: Float; var aState: array of Float); virtual; abstract;	виртуальный, абстрактный	Вычисляет значение вектора оцениваемых параметров для момента времени t_i

Как видно из приведенных таблиц, основным свойством данного класса является поле *FEstim*, содержащее значения вектора оцениваемых параметров.

Обсуждаемый класс имеет ссылку на объект, реализующий динамику ЛА (*TObjectDynamics*), который передается классу при создании.

Основной метод класса – *GetMeas*, вычисляющий вектор измерений на момент времени t_i и в соответствии с некотором вектором параметров *aState* в данном классе объявлен абстрактным с целью последующего перекрытия в классах-потомках.

! В данном классе не определены ссылки на объекты, реализующие модель измерений.

Это обусловлено большим разнообразием вариантов измерительных устройств, используемых конкретной навигационной подсистемой.

Таким образом, при реализации каждого класса-наследника, реализующего конкретный тип НС, необходимо предусмотреть соответствующие поля и передать в методе *Create* реальные объекты-измерители.

При реализации класса *TBINS*, формализующего модель БИНС, необходимо организовать интегрирование основного навигационного уравнения, в правые части которого входят измеренные значения перегрузок и угловых скоростей ЛА.

Кроме того, в классе должна быть предусмотрена ссылка на объект, реализующий модель гравитационного поля Земли.

По отношению к классу-предку в *TBINS*, существует внутренний объект типа *TSimpleAdams*, интегрирующий основное навигационное уравнение.

Вызов метода *RunTo* данного объекта производится в методе *GetEstim* до текущего момента t_i.

В качестве интегратора необходимо использовать именно объект *TSimpleAdams*, реализующий метод прогноза-коррекции и учитывающий в численном решении значения измерений за шаг интегрирования.

В таблицах приведены названия и описания новых или перекрытых по отношению к родительскому полей и методов данного класса.

Свойства класса	Тип	Описание
FNSIntegrator	TSimpleAdams	внутренний интегратор
AccMC	TAbstractMeasChannel	ссылка на объект, реализующий модель акселерометров
RGMC	TAbstractMeasChannel	ссылка на объект, реализующий модель ДУС
Gravl	TGravIModel	ссылка на объект, реализующий модель гравитационного поля Земли
ofQuaternlon	логический	флаг использования параметров Родрига-Гамильтона для определения ориентации ЛА (если нет – используется уравнение Пуассона)
Param	TVector	вектор параметров модели

Метод	Тип	Действие
constructor Create (const anOwner: Pointer; const acmUpdate: Byte; aAccMC, aRGMC: TAbstractMeasChannel; aGravi: TGravIModel);	конструктор класса	Создает объект, инициализирует внутренние переменные.
destructor Destroy; override;	деструктор класса	Уничтожает объект
procedure GetInlDataFrom-FIle(aFIleName: String); virtual;	виртуальный	Считывает исходные данные из ini-файла проекта
procedure Initialize(atime: Float; aEstim: TVector; aParam: TVector): virtual;	виртуальный	Инициализирует начальные значения вектора оцениваемых параметров и вектора параметров модели
procedure GetEstim(const ti: Float; var aState: array of Float); override;	виртуальный	Вычисляет значение вектора оцениваемых параметров для момента времени t_i

Исходные данные для инициализации параметров БИНС (точность интегрирования, шаг, имя файла результатов и т. п.) считываются из inl-файла проекта в секции [*BINS*].

[BUS]
Tolerance=le – 6
H = 0.01
NOutStep = 10
OutPutMode = 0
OutPutDestination = 1
BINSFile = d:\app\dss\res\bins.dat
ofQuatenion = 0

Помимо перечисленных навигационных устройств в бортовом интегрированном контуре существует блок комплексирования, в котором в соответствии с той или иной схемой осуществляется коррекция, как навигационного решения, так и параметров модели БИНС.

Необходимо отметить, что классов, реализующих данный блок, может существовать несколько, в зависимости от выбранной схемы комплексирования, количества комплексированных навигационных подсистем несколько, иными словами, невозможно указать единого предка для обсуждаемых классов.

Ниже мы приведем описание класса *TSimpleDataFusion*, реализующего слабосвязанную схему комплексирования БИНС и спутниковой навигации, корректирующую начальные условия БИНС.

```
TSimpleDataFusion = class(TObject)
constructor Create(aBINS: TBINS; aGNSS: TGNSS; aIniFIame: String);
protected
FCurTime: Float;
FofCorrection: Boolean;
FMeasVector: TVector;
procedure GetFMatrix;
procedure GetHMatrix;
public
DFState: TVector;
DFCovMatrix: Tfector;
CurBINS: TBINS;
CurGNSS: TGNSS;
property ofCorrection: Boolean read FofCorrection;
procedure GetIniData(aIniFName); virtual;
procedure Eval(ti: Float); virtual;
end;
```

Как видно из приведенного описания, при создании объекта в методе *Create* классу передаются объекты, реализующие алгоритмы работы БИНС и многоканального ГЛОНАСС/GPS приемника.

На основании результатов работы данных подсистем формируется вектор «измерений» *FMeasVector*, представляющий собой невязку определения положения и скорости ЛА с помощью БИНС и приемника соответственно.

В качестве вектора состояния динамической системы *FDFState* используется вектор ошибок БИНС, который оценивается методом динамической фильтрации в соответствии со сформированным вектором измерений и моделью ошибок БИНС. Оценка вектора состояния сопровождается соответствующей ковариационной матрицей *DFGovMatrix*.

Процедура фильтрации содержится в методе *Eval* класса.

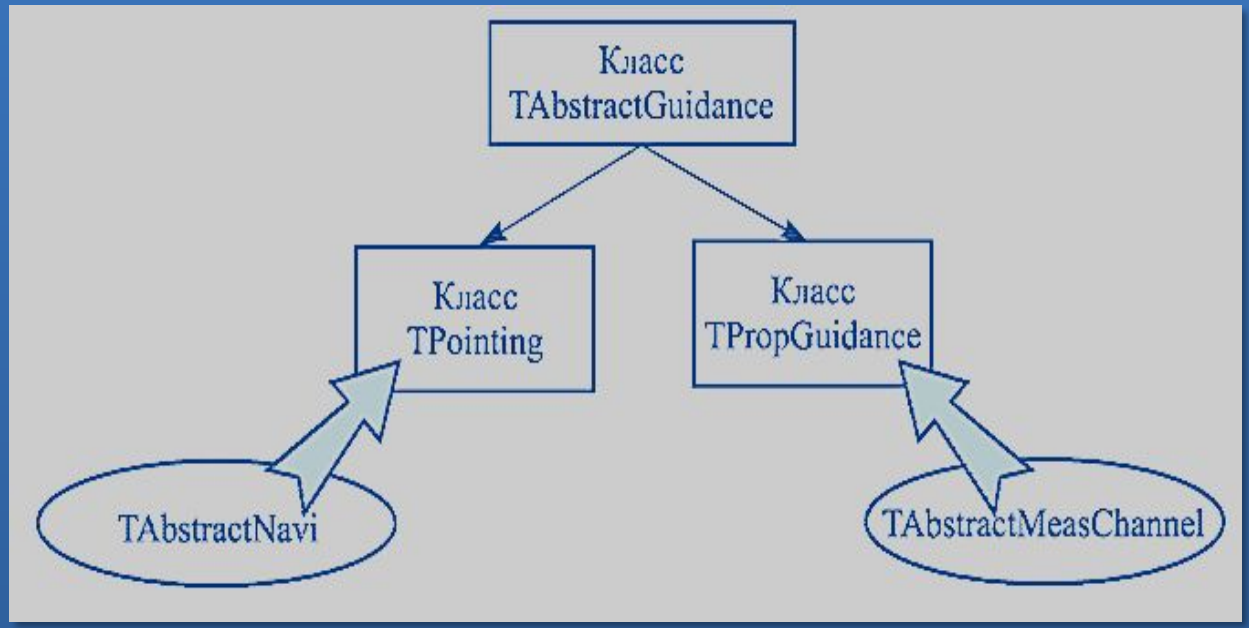
В случае установки флага коррекции *ofCorrection* после обработки измерений производится коррекция выходных данных БИНС и вектора ее ошибок путем вызова метода *Initialize* класса *TBINS*.

Начальные условия для алгоритма комплексирования задаются вызовом метода *GetIniData*, который задает значения внутренним полям класса, считывая их из ini-файла проекта.

В зависимости от целевой задачи конкретного ЛА и аппаратных требований могут использоваться системы наведения, отличающиеся принципом действия, составом измеряемых параметров, алгоритмами предварительной обработки информации и т. п.

Кроме того, зачастую практически невозможно разделить задачу наведения и навигации, а также задачу управления, поскольку задача наведения тесно связана с принятием решения и выработкой сигналов командного управления.

Тем не менее, в рамках обсуждаемой технологии рассмотрим иерархическую цепочку классов, реализующих алгоритмы наведения



Алгоритмы наведения

Класс *TAbstractGuidance* содержит объявления свойств и шаблоны абстрактных методов, общие для всей цепочки классов, реализующих модели систем наведения.

Ниже приводится описание данного класса.

```
TabstractGuidance = class(TObject)
    constructor              Create(aCurObjectDynamics,
aTargetDynamics:
    TObjectDynamics; aIniFName: String);
    protected
    FCurTime: Float;
    public
    CCState: TVector;
    CurObjectDynamics: TObjectDynamics;
    TargetDynamics: TObjectDynamics;
    procedure GetIniData(aIniFIame); virtual; abstract;
    procedure GetGuidance(ti: Float); virtual; abstract;
end;
```

Как видно из приведенного описания, при создании в методе *Create* классу передаются объекты типа *TObjectDynamics*, реализующие модель динамики ЛА и цели, ссылки на которые сохраняются во внутренних переменных класса *CurObjectDynamics* и *TargetDynamics*.

Основной метод класса, вычисляющий вектор командного управления *CCState* на текущий момент времени t_i, – процедура *GetGuidance*.

Начальные условия и параметры алгоритма наведения задаются методом *GetIniData*, считывающий значения из ini-файла проекта.

Необходимо отметить, что реальное наполнение вектора командного управления и алгоритма наведения зависит от конкретной системы и должно быть реализовано в классах-потомках.

Ниже приведен пример класса *TPointing*, реализующего алгоритм решения задачи наведения как краевой задачи управления движущегося объекта.

```
TPointing = class (TAbstractGuidan.ee)
    constructor Create(aTargetDynamics: TObjectDynamics; aIniFIame: String;
    aBINS: TBINS; aRGMChannel: TRGMChannel);
    public
    BINS: TBINS;
    RGMChannel: TRGMChanne1;
    procedure GetIniData(aIniFName); override;
    procedure GetGuidance(ti: Float); override;
    end;
```

Данный класс формирует вектор требуемых перегрузок в связанной СК, приближенно решая краевую задачу на основе информации от БИНС и априорных сведений о координатах цели (*TargetDynamic*).

Помимо этого, при формировании требуемых перегрузок учитывается ограничение на текущее значение вектора абсолютной угловой скорости ЛА, поступающее с объекта *RGMChannel*.

Сформированное значение вектора командного управления подается на вход системы управления (автопилота).

Как правило, сложно разделить прицельно-навигационную систему и систему управления в силу их высокой интегрированности.

Тем не менее, в рамках предлагаемой технологии в целях универсализации ПМО и построения иерархической объектной структуры будем определять класс, реализующий алгоритм системы управления как блок, осуществляющий сравнение сигналов командного управления, полученных в результате решения задачи наведения, с текущими параметрами движения ЛА и последующее формирование управляющих сигналов для исполнительных органов (сервоприводы), а также отработку этих сигналов в виде значений углов отклонения органов управления (аэродинамические и газодинамические рули, отклоняющиеся сопла и т. п.).

Для сложных авиационных систем возможно и иногда целесообразно более детальное и глубокое разделение данной системы на отдельные объекты (например, выделение отдельных классов для каждого из управляющих органов).

Однако для обсуждаемых ЛА будем использовать единый класс *TControlSystem*, реализующий модель автопилота как единого механизма, замыкающего контур управления ЛА.

Ниже приведем описание данного класса.

```
TControlSystem = class(TObject)
  constructor Create(aCurObjectDynamics: TObjectDynamics;
  aIniFName: String; aRGMChannel: TRGMChannel;
  aAccMChannel: TAccMChannel; aGuidance: TPointing);
  protected
  FCurTime: Float;
  public
  CSState: TVector;
  AeroDelta: TLoc;
  ThrustDelta: TLoc;
  RGMChanne1: TRGMChanne1;
  AccMChannel: TAccMChannel;
  Guidance: TPointing;
  CSInt: TSimpleAdams;
  CurObjectDynamics: TObjectDynamics;
  procedure GetIniData(aIniFIame); virtual;
  procedure GetControl(ti: Float); virtual;
  end;
```

Как видно из приведенного описания, при создании объекта в методе *Create* классу передаются объект *aCurObjectDynamics* типа *TObjectDynamics*, реализующий модель динамики ЛА, ссылка на который сохраняется во внутренней переменной класса *CurObjectDynamics*, а также объекты *aRGMChannel* и *aAccMChannel*, реализующие модели измерителей ДУС и акселерометров соответственно.

Кроме того, в этом методе передается объект *aGuidance* типа *TPointing*, реализующий алгоритм метода наведения и позволяющий использовать вектор командного управления, формируемого системой наведения при решении задач управления.

Основной метод класса, вычисляющий вектор управляющих сигналов *CSState* на текущий момент времени t_i, – процедура *GetControl*.

В этом методе происходит сравнение вектора командного управления и соответствующих измеренных значений перегрузки ЛА, формирование управляющих сигналов и отработку этих сигналов в виде значений углов отклонения аэродинамических и газодинамических рулей (*AeroDelta*, *ThrustDelta*).

Для реализации динамики звеньев автопилота предусмотрен внутренний объект-интегратор *CSIni* типа *TSimpleAdams*.

Начальные условия и параметры автопилота задаются методом GetIniData, считывающим значения из ini-файла проекта.

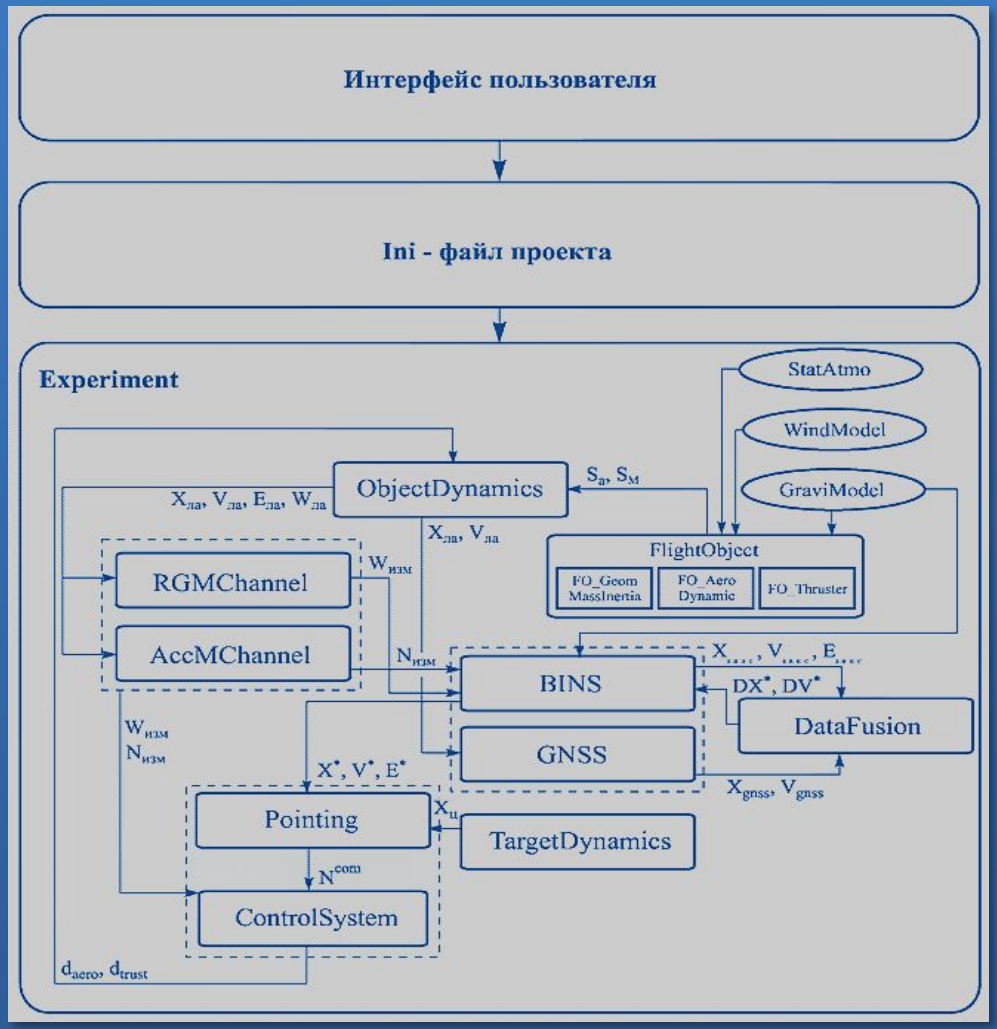
Вычисленные здесь значения отклонения органов управления передаются на вход объекта *FlightObject*, замыкая тем самым контур моделирования управляемого движения маневренного ЛА.

ВОПРОС 4

Полная объектная структура моделирования

Были рассмотрены все классы, формализующие блоки функциональной схемы моделирования бортовой интегрированной системы навигации и наведения маневренных ЛА.

Следующая задача состоит в объединении созданных фрагментов ПМО в объектную схему моделирования, соответствующую исходной задаче имитационного моделирования. Такая схема приведена на рисунке.



Все описанные выше объекты являются полями основного класса проекта – *TExperiment*. Данный класс предназначен для создания, хранения и обеспечения необходимых связей между объектами, реализующими блоки функциональной схемы моделирования.

Кроме того, этот класс реализует непосредственно конкретную техническую задачу имитационного моделирования.

Иными словами, непосредственной реализацией замкнутого контура ЛА в соответствии с функциональной схемой (т. е. моделированием управляемого движения ЛА) не исчерпывается диапазон задач имитационного моделирования.

В зависимости от технической постановки задачи в рамках излагаемой технологии могут изучаться вопросы анализа влияния возмущающих факторов на точность решения целевой задачи, анализа точности решения целевой задачи, анализа допустимых траекторий полета ЛА, проектирования схем комплексирования, наведения и системы управления, статистическое моделирование возмущенного движения ЛА и т. п.

Приведем описание данного класса.

```
TExperiment = class(TObject)
constructor Create(const anOwner: Pointer; const acmUpdate: Byte);
destructor Destroy; override;
protected
FCurTime: Float;
FTBegin, FTEnd, Fdt: Float;
{-------------------------------------------------------------------------------------------------------------}
FUpdate: Byte;
FOfPausedCalc: Boolean;
FOfBINS, FOfGNSS, FOfPointing, FOfCS: Boolean;
procedure SetOfPausedCalc(Value: Boolean);
public
FODFile, FBINSFile, FGNSSFile, FPointingFile, FCSFile: TextFile;
Owner: Pointer;
ObjetcDynamics: TObjectDynamics;
TargetDynamics: TObjectDynamic;
BINS: TBINS;
GNSS: TGNSS;
Pointing: TPointing;
DataFusion: TSimpleDataFusion;
CS: TControlSystem;
AccMChannel: TAccMChannel;
RGMChanne1: TRGMChanne1;
Gravi: TGravi;
StatAtmo: TAtmo;
WindModel: TWindModel;
procedure InformOwner; virtual;
procedure Dolt; virtual;
procedure SetIniData(IniFName: String); virtual;
property CurTime: Float read FCurTime;
property OfPausedCalc: Boolean read FOfPausedCalc write SetOfPausedCalc;
end;
```

Данный класс располагает внутренними полями, представляющими собой объекты от вышеуказанных классов и тремя основными методами, объявленными как виртуальные.

При создании класса в методе *Create* происходит инициализация полей *Owner* (указатель на форму-владельца класса) и константы сообщения *FUpdate*.

Данные поля необходимы классу для формирования Windows-сообщения, обрабатываемого визуальной формой проекта в целях визуализации процесса моделирования и возможности управления вычислительным процессом.

После создания объект *Experiment* пустой, т. е. не заполнены основные внутренние поля и не созданы внутренние объекты, соответствующие структуре контура управления ЛА.

Для наполнения структуры необходимо вызвать метод *SetIniData*, который подсоединяет ini-проекта и считывает значения внутренних полей (FT0 – начало процесса моделирования, FTk – окончание моделирования, FdT – шаг моделирования, имена файлов результатов, флаги использования подсистем).

Кроме того, в соответствии с установленными флагами и именем ini-файла проекта происходит создание всех требуемых для моделирования объектов. После этого устанавливается флаг *ofPausedCalc*, сигнализирующей о приостановке процесса моделирования.

Для старта моделирования вызывается метод *Dolt*, реализующий непосредственно саму задачу моделирования.

В данном методе в цикле организован вызов основных выполняемых методов объектов структуры моделирования в последовательности, определяемой целевой задачей моделирования.

Помимо этого, здесь же размещаются вычислительные и логические блоки, формализующие специфику задачи моделирования.

Например, в случае решения задачи моделирования управляемого движения ЛА на каждом «шаге» моделирования, имитирующем функционирование всех подсистем, для текущего момента времени *FCurTime* происходит следующее:

1. Вызывается метод *ObjectDynamics.RunTo*(*FGurTime*), позволяющий спрогнозировать движение ЛА на момент *FCurTime*.

2. Вызываются методы *RGMChannel.GetMeas*(*FCurTime*) и *AccM-Channel.GetMeas*(*FCurTime*), формирующие измерения ДУС и акселерометров на основе вектора состояния ЛА (объект *ObjectDynamics*).

3. Вызывается метод *BINS.GetEstim*(*FCurTime*), позволяющий получить оценку положения, скорости и ориентациии ЛА на момент *FCurTime* в соответствии с текущими измерениями (объекты *RGMChannel* и *AccMChannel*).

4. Вызывается метод *GNSS.GetEstim*(*FCurTime*), позволяющий получить оценку положения и скорости ЛА на момент *FCurTime* в соответствии с текущим состоянием ЛА на основе методов спутниковой навигации.

5. Вызывается метод *DataFusion.Eval*(*FCurTlme*), позволяющий получить оценку ошибок БИНС на основе используемого алгоритма комплексирования.

6. Вызывается метод *Pointing.GetGuldance*(*FCurTlme*), позволяющий сформировать вектор командного управления на основе данных БИНС и информации о параметрах движения цели (объект *TargetDynamics*).

7. Вызывается метод *CS.GetControl*(*FCurTime*), рассчитывающий углы отклонения органов управления ЛА в соответствии с командным управлением и измеренными параметрами движения ЛА.

8. Вызывается метод *InformToOwner*, формирующий Windows-сообщение, обрабатываемое формой-владельцем для визуализации процесса моделирования.

Далее описанный выше процесс повторяется до момента окончания моделирования. Если решается более сложная задача (например, статистическое моделирование управляемого полета ЛА по методу Монте-Карло), то здесь же реализуются операции по сбору и обработке статистики по реализациям с последующей выдачей результатов.

Необходимо отметить, что ПМО организовано таким образом, что связь объекта *Experiment* с интерфейсом пользователя организована посредством использования ini-файла для задания исходных данных и параметров моделирования, а также при помощи ссылки *Owner* на форму интерфейса для вывода результатов моделирования.

Такой подход позволяет использовать различные интерфейсы, инвариантные по коду к основной объектной структуре прикладной задачи.

Помимо приведенных описаний классов остановимся на ряде вспомогательных процедур и функций, оформленных в виде отдельных модулей и скомпонованных по функциональному признаку.

В рамках обсуждаемого ПМО были созданы следующие модули

Наименование модуля	Описание
Global	Описание глобальных типов, констант и переменных проекта
Cmath	Реализация функций алгебры комплексных чисел
Mathem	Реализация математических процедур и функций
Matrix	Реализация процедур и функций матричной алгебры
Qmath	Реализация процедур и функций алгебры кватернионов
Tmath	Реализация процедур и функций тензорной алгебры
VMath	Реализация процедур и функций векторной алгебры
Frames	Реализация процедур и функций для расчета матриц перехода между используемыми системами координат

Вопросы на самостоятельную подготовку

1. Моделирование неуправляемого движения летательных аппаратов.
2. Моделирование бортового измерительного комплекса.
3. Моделирование блока навигации и управления.
4. Полная объектная структура моделирования.