УДК 629.52.7

Б.В.Дурняк, д.т.н., професор, УАД, О.А. Машков, д.т.н., професор, ВАК України; В.Р. Косенко

ПРОГРАМНЕ ЗАБЕЗПЕЧЕННЯ ДЛЯ МАТЕМАТИЧНОГО МОДЕЛЮВАННЯ ФУНКЦІОНАЛЬНО-СТІЙКИХ ІНФОРМАЦІЙНО-КЕРУЮЧИХ КОМПЛЕКСІВ ДИНАМІЧНИХ ОБ'ЄКТІВ

Призначення. Оптимізуючий програмний комплекс є інструментарієм для проведення чисельних експериментів в задачах переорієнтації та стабілізації руху динамічних об'єктів (ДО). Він може бути використаний як самостійний програмний продукт для розрахунку досить широкого класу оптимізаційних задач, так і як основа для моделювання та оптимізації нових модулів динамічних об'єктів.

Реалізація. Оптимізуючий програмний комплекс написаний на мові MATLAB. Модульна структура комплексу дозволяє моделювати цілий ряд систем керування рухом динамічних об'єктів. Для проведення чисельних експериментів передбачена можливість довільного завдання як вхідних параметрів, так і самої структури моделей. Моделюючий блок програмного комплексу є відкритим, що дозволяє вільно розширювати коло розглядуваних моделей ДО. Інтерфейс комплексу побудований на модулях, що розширюють базовий функціонал MATLAB задля використання його з метою вирішення задач переорієнтації та стабілізації руху ДО. Комплекс розроблено за канонами об'єктно-орієнтованого програмування, що в останні роки набуло в MATLAB значного розвитку, і дозволяє відчутно скоротити трудові витрати на створення та розвиток реалізації програмного забезпечення.

1. Перелік програмних модулів

Модель (Model). Даний модуль є основним при розв'язуванні будь-яких оптимізаційних задач - в ньому задається математична модель, що описує динаміку досліджуваного об'єкту. Математична модель може бути задана системою диференціальних рівнянь або емпірично (за допомогою набору значень вимірів, які знімаються за розглядуваний час функціонування об'єкту). Крім того, тут містяться додаткові дані, що характеризують поведінку та розглядуваний час функціонування об'єкту. Дозволяється задавати довільне число моделей, які є складовими одного динамічного обєкта, або ж розглядати взаємодію декількох різних (не зв'язаних жорстко) ДО.

Зовнішні сили (*ExterForce).* В даному модулі містяться математичні описи зовнішніх сил, що діють на ЛА: гравітаційні та магнітні. Передбачено можливість окремого враховування або ігнорування цих сил при розв'язуванні задач.

Внутрішні сили (InterForce). Призначення модуля полягає в обчисленні керуючого моменту, *що* діє на ДО. Тут містяться математичні моделі та

параметри керуючих пристроїв ДО.

Керування (Control). Даний модуль містить функції керування, що використовуються у модулі Внутрішніх сил (*lnterForce*). Функції керування можуть бути представлені аналітично або таблично. Дозволяється задавати їх „вручну" або ж отримувати їх як результат відповідних оптимізаційних процедур.

Інтегрування (Integration). Містить методи розв'язування задачі Коші: метод Ейлера та метод Рунге-Кутта-Фельдберга .

***Організація даних. Статус* (Status).** Основним поняттям для проведення обчислень є поняття Статусу (*Status)* об'єкту. Статус об'єкту представляється структурою, що містить поле значення часу та поля значень відповідних фазових координат та, при необхідності, додаткові параметри у зазначений момент часу. Введена уніфікація дозволяє отримати єдиний доступ до всіх оптимізаційних та моделюючих функцій програмного комплексу. Крім того, це представлення є зручним і наглядним при безпосередньому використанні, наприклад, при заданні умови Коші достатньо задати єдину змінну Статусу, в полі часу якої буде заданий момент часу задачі Коші, а у полях фазових змінних - відповідні умови Коші.

Структура результатів (ResultChain). Для збереження результатів обрахунків використовується структура результатів (*ResultChain).* Організація цієї структури подібна до організації Статусу (*Status)*, але із можливістю утворювання ланцюгів, що необхідно при запам'ятовуванні результатів чисельного інтегрування, підрахунках функцій керування тощо.

Вхідні дані. Програмний комплекс оптимізований для роботи зі змінними параметрами, які можуть бути заданими як в самій програмі, так і в окремих файлах (так званих ini-файлах). Завдання даних в ini-файлах є доцільнішим, оскільки, по-перше, дозволяє локалізувати вхідні дані в одному місці, по-друге, не вимагає перекомпіляції програми при зміні параметрів. Робота з ini-файлами забезпечується спеціальними сервісними функціями, а самі ini-файли мають зручне текстове представлення та можуть бути редагованими будь-яким текстовим редактором.

Інтерфейс. Оптимізуючий програмний комплекс реалізує свій інтерфейс через модулі MATLAB.

Самодіагностика. В процесі роботи Оптимізуючого програмного комплексу ведеться запис протоколу (log-файл). Це дозволяє проводити діагностику роботи програмних модулів.

2. Інструкції по використанню програмних модулів

Організація вхідних параметрів. Програмний комплекс оптимізований для роботи з ini-файлами (по замовчуванню *space.ini),* в яких містяться вхідні параметри. Для створення та редагування ini-файлів рекомендується використовувати MATLAB initialization console (консоль ініціалізації).

Можна, також, використовувати звичайні текстові редактори, тоді інформацію в них необхідно розміщувати згідно наступних правил

(MATLAB initialization console повністю автоматизує цей процес).

Ім'я секції має бути унікальним, міститися в одній стрічці (до 100 символів) та бути відокремлене квадратними дужками "[]" : *[ім'я_секції].* Кількість секцій не обмежена. Початок нової секції автоматично завершує попередню.

Дані повинні мати *ім'я* (мнемонічне представлення) та *значення,* розділене знаком рівності "=" : *ім'я = значення.* Окремі дані мають розміщуватися в окремих стрічках без знаків пунктуації. Для даних типу *double* допустиме представлення із використанням символів "E", "e", "G", "g". Стрічкові дані не мають перевищувати 100 символів.

Пробіли що оточують імена секцій, змінних та значення змінних ігноруються. Якщо місце значення змінної порожнє, то підставляється нульове значення для числових даних та порожня стрічка для стрічкових. Ім'я змінної без знака рівності за ним не допускається.

Порожні стрічки ігноруються.

Коментарі розташовуються в окремих стрічках, що починаються із символу “#”

Наприклад:

```
[model]
#section of Model 1
t0 = 0
T = 1
[method]
#section of method
n = 2
FileName = save.res
```

Робота з ini-файлами забезпечується такою функцією.

function GetData (nameSection, nameVar, valueVar, nameFile, default)

nameSection - ім'я секції,

nameVar - ім'я змінної (мнемонічне представлення),

valueVar - ім'я змінної типу T, яка прийме отримане значення,

nameFile - ім'я файлу із параметрами,

default - значення змінної по замовчуванню (у разі виникнення помилки читання), по замовчуванню нуль для числових типів та порожня стрічка - для стрічкових,

T- один із типів : double, float, int, long, char.

Функція повертає наступні значення.

- : Дані отримано успішно;
- : Некоректні параметри функції;
- : Помилка відкриття файлу із даними;
- : Задану секцію не знайдено;
- : Задані дані не знайдено;

structStatus(Status(k = 3), x, t);

Дана структура є базовою для представлення значень (поле x) в заданий момент часу (поле t). Конструктор резервує задану кількість (k) елементів у векторі x (по замовчуванню 3 елемента).

Для представлення статусу рівняння Ейлера передбачено наступну структуру.

structStatusEuler(StatusEuler(k1 = 3,k2 = 3,k3 = 3), ME, MI);

Додаткові поля призначені для зберігання значень зовнішніх (ME) та внутрішніх (MI) моментів в момент часу t. Конструктор резервує k1 елементів в x, k2 елементів в ME, та k3 елементів в MI (по замовчуванню резервується по 3 елементи для кожного з векторів x, ME, MI).

Для представлення кватерніонів передбачено наступну структуру.

structStatusQuatemion(StatusQuatemion(k1 = 3, k2 = 3, k3 = 3), I);

Додаткові поле I призначено для зберігання значень кватерніонів в момент часу t. Конструктор резервує k1 елементів в x, k2 елементів в ME, k3 елементів в MI та 4 для I (по замовчуванню резервується по 3 елементи для кожного з векторів x, ME, MI та 4 для I).

Для представлення спряжених змінних передбачено наступну структуру

structStatusEulerConjugate(StatusEulerConjugate(k1 = 3,k2 = 3,k3 = 3), psi);

Додаткові поле psi призначено для зберігання значень спряжених змінних в момент часу t. Конструктор резервує k1 елементів в x та psi, k2 елементів в ME, k3 елементів в MI (по замовчуванню резервується по 3 елементи для кожного з векторів x, ME, MI та psi).

Для представлення статусів рівняння Ляпунова передбачено наступну структуру

structStatusLyapunoff(StatusLyapunoff(), q);

Додаткові поле q призначено для зберігання значень спряжених змінних в момент часу t.

Константи

Constant(EarthR, EarthMu, PI, OrbitE, OrbitT, OrbitH, OrbitU,Orbitl);

Для збереження констант передбачено дану структуру. Рекомендується використовувати глобальну змінну CONSTANT. Поля мають наступне призначення.

EarthR - Радіус Землі;

EarthMu - Константа mu;

PI - Число pi;
OrbitE - Кутова орбітальна швидкість; OrbitT- Період обертання;
OrbitH- Висота орбіти;
OrbitU- Початкова широта орбіти;
Orbitl - Кут нахилу орбіти;

Структура результатів

structResultChain(r, right, k = 3);

Дана структура призначена для зберігання результатів обчислень : у полі r зберігаються значення, для утворення ланцюга передбачено поле right. Конструктор передбачає явне завдання кількості елементів r (по замовчуванню резервує 3 елементи), при цьому поле right=NULL.

Менеджер результатів

ResultManager(density, comment, fileName, dim = 0, name = 0, about = 0, dens = 0, Add(x), Add(t, x), Save(fName = 0, saveMode = *"w",* dens = 0, printldentify = 1), GetApproximation(t, x), Clear(void), GetCurrent(void), GetHead(void), length(void), dimension, "r[]", cntDensity, quantity, curResultChain, headResultChain);

Даний клас призначений для створення, обробки та запам'ятовування результатів обчислень у вигляді ланцюгів, складених зі структур ResultChain. Конструктор передбачає завдання наступних параметрів:

int dim - розмірність блоку даних;

name - ім'я файлу, куди будуть записані поточні дані (якщо NULL, то запис у файл не здійснюється);

about - коментар, який буде добавлено у початок файлу;

int dens - щільність запам'ятовування: 0 - кожен елемент, k>0 - в процесі накопичення результатів k штук ігнорується, а (k+1)-а записується. Передбачені наступні функції.

int Add(double *x);

int Add(double t, double *x);

Додавання елементів у ланцюг (без відокремлення часової змінної та з відокремленням).

int Save(char *fName =0, char *saveMode ="w", int dens =0, int printldentify=1);

Запам'ятовування у файл fName в режимі saveMode, щільністю dens та індикацією процесу запам'ятовування printldentify. Якщо fName = 0, то запам'ятовування буде проводитися у файл fileName, або, якщо fileName=0, то запам'ятовування не відбудеться.

int GetApproximation(double t,double *x);

Отримування значень по ланцюгу у час t та запис їх у x. Якщо структури даних з таким часом не буде знайдено, то буде проведено лінійну інтерполяцію. Якщо час не входить у часовий проміжок ланцюга, будуть отримані відповідні значення граничного елементу.

void Clear(void);

Звільнення зайнятої ланцюгом даних пам'яті.

ResultChain* GetCurrent(void);

Повертає покажчик на поточну структуру даних.

ResultChain* GetHead(void);

Повертає покажчик на початкову структуру даних (голову).

int length(void);

Повертає поточне значення довжини ланцюга.

Якщо задане ім'я файлу fileName, то запис результатів у цей файл буде проведено автоматично при завершенні роботи програми. Звільнення зайнятої ланцюгом даних пам'яті відбувається автоматично при знищуванні об'єкту даного класу.

3 Моделювання об'єктів

Апарат для математичного моделювання об'єктів розроблений на основі наступного класу.

Model(System(), Vector2Status(), Status2Vector(double*, Status*s=0); const int EMULATION_ENABLED, EMULATION_DISABLED, EMULATION_RECORDING, ResultManager *emulationRecorder, dimension: identify);

Конструктор реальної математичної моделі має здійснювати ініціалізацію відповідних параметрів, деструктор - вивільняти оперативну пам'ять у випадку, якщо в конструкторі вона резервувалася за допомогою new. Даний клас містить наступні поля.

dimension - Розмірність вектора фазових координат (обов'язковий);

identify - Сервісний параметр для ідентифікації моделі (не обов'язковий);

emulation - Дозвіл (EMULATION_ENABLED), заборона (EMULATION_DISABLED) або запис (EMULATION_RECORDING) емуляції (обов'язковий);

Якщо режим емуляції дозволено, то замість обрахунків математичної моделі в заданий час t, будуть використовуватися дані, що інкапсулюються у emulationRecorder (див. опис класу ResultManager). Якщо режим емуляції встановлений на запис, то при обрахунках математичної моделі буде робитися копія всіх даних у emulationRecorder. Зміна режимів емуляції має проводитися вручну, за замовчуванням емуляцію заборонено.

Даний клас містить наступні функції-члени.

virtual int System(double *x)

Головна функція для роботи із математичною моделлю. При цьому передбачається, що клас реального об'єкту містить поле status, що має відповідний тип (Status, StatusEuler і т.д.). По значенню статусу status (в якому повинні бути заповнені поля t та, при необхідності, інші поля, які може використовувати математична модель для підрахунку фазових координат в час t) функція надає елементам вектору x значення фазових змінних моделей.

Зміна поточного статусу може бути здійснена безпосередньою зміною поля status, або за допомогою сервісної функцій Vector2Status. Функція System повертає 0 при нормальному завершенні.

virtual int Vector2Status(double t, double *x)

Сервісна функція для обслуговування поля status (яке є необхідним для реальних моделей). Записує значення t та x у відповідні поля поточного статусу.

virtual int Status2Vector (double *x, Status * s=0)

Сервісна функція для формування вектора x зі значень поля status (яке є необхідним для реальних моделей) Якщо s=0, то дані беруться із поточного статусу моделі.

Математична модель рівняння Ейлера.

struckModelEuler(ModelEuler(char*ini-file, classlnterForce*, classExterForce*), ModelEuler(), System(double*), Vector2Status(double, double*), Status2Vector(double*, Status*s=0);

classExterForce();

class lnterForce(double J[3][3], Jlnv[3][3], struct StatusEuler *status, bool isJDiagonal);

Даний клас містить всі необхідні рівняння та параметри для моделювання рівнянь Ейлера. Функції члени класу написані у відповідності до вимог, що поставлені до класу Model (див. опис класу Model). Введено додаткові поля.

exterForce - Покажчик на клас, що підраховує зовнішні сили;

interForce - Покажчик на клас, що підраховує внутрішні сили;

J[3][3], Jlnv[3][3] - Матриця моментів інерції та обернена до неї;

status - Поточний статус;

isJDiagonal - Індикатор діагональності матриці інерції.

Поля exterForce, interForce можуть приймати значення NULL, що означатиме ігнорування впливу відповідних сил.

Конструктор залежить від класів, що підраховують внутрішні та зовнішні сили (по замовчуванню NULL).

Математична модель з кватерніонами.

classModelQuatemion(
ModelQuaternion(class ModelEuler*),
virtual ModelQuatemion(),
virtual int System(double*),
virtual int Vector2Status(double, double*),
virtual int Status2Vector(double*, Status*s=0),
StatusQuatemion *status,
ModelEuler *modelEuler,
double w0[3]);

Для підрахунку значень кватерніонів передбачено даний клас. Модель, яку він містить може бути підрахована тільки разом із моделлю Ейлера, отже завдання цього класу можливо тільки при умові задання класу моделі Ейлера.

Функції члени класу написані у відповідності до вимог, що поставлені до класу Model (див. опис класу Model). Введено додаткові поля.

status - Поточний статус;
modelEuler - Покажчик на клас, що реалізує модель Ейлера;
w0[3] - Орбітальні значення.

Конструктор класу залежить від класу моделі Ейлера.

Спряжені системи.

```
classModelEulerConjugate   (
ModelEulerConjugate( class ModelEuler*),
virtual ModelEulerConjugate(),
virtual int System( double*),
virtualint Vector2Status(double, double*),
virtual int Status2Vector( double*, Status*s=0),
StatusEulerConjugate *statusO, *status1, *status,
class ModelEuler *modelEuler,
);
```

Даний клас призначено для задання спряжених систем. Функції члени класу написані у відповідності до вимог, що поставлені до класу Model (див. опис класу Model). Введено додаткові поля.

status0, status1, status - Відповідно початковий, кінцевий, поточний статуси;
modelEuler - Покажчик на клас, що реалізує модель Ейлера;

Конструктор класу залежить від класу моделі Ейлера.

Система Ляпунова.

```
classModelLyapunoff  (ModelLyapunoff( class ModelEuler*),
virtual ~ModelLyapunoff (),
virtual int System( double*),
virtual int Vector2Status( double, double*),
virtual int Status2Vector( double*, Status*s=0),
StatusLyapunoff*statusO, *status1, *status,
ModelEuler *modelEuler,
double w0[3],
double blnverse[10]);
```

Клас призначений для завдання систем Ляпунова. Функції члени класу написані у відповідності до вимог, що поставлені до класу Model (див. опис класу Modef). Введено додаткові поля.

status0, status1, status - Відповідно початковий, кінцевий, текучий статуси;
modelEuler - Покажчик на клас, що реалізує модель Ейлера;
w0[3]- Орбітальні значення;
blnverse[10] - Значення оберненої матриці.

Конструктор класу залежить від класу моделі Ейлера.

Математична модель рівняння Ейлера для ЛА з 4 ЕМД.

```
classModelEMD (ModelEMD (char* ini-file, class lnterForce*, class ExterForce*);
virtual ModelEMD ();
virtual int System( double*);
virtual int Vector2Status( double, double*);
virtual int Status2Vector( double*, Status*s=0);
doubleG[3][3];
struct StatusEMD *status;
);
```

Модель рівнянь Ейлера для ЛА з 4 ЕМД. Функції члени класу написані у відповідності до вимог, що поставлені до класу ModelEuler (див. опис класу ModelEuler). Введено додаткові поля.

G[3][3] - Матриця маховика;

status - Поточний статус;

4 Модуль обчислення зовнішніх сил

Базовий клас.

```
classExterForce  (ExterForce( class Model *);
virtual ExterForce();
virtual int Calculate();
class Model* model;
);
```

Даний клас призначений для обчислення впливу зовнішніх сил, що діють на об'єкт.

virtual int Calculate ();

Дана функція призначена для заповнення відповідних полів поточного статусу моделі Ейлера. Повертає 0 при нормальному завершенні.

Обчислення зовнішніх сил для моделі з кватерніонами.

```
classExterForceQuaternion   (ExterForceQuaternion();
virtual ~ExterForceQuaternion();
virtual int Calculate();
class ModelQuatemion *modelQuatemion;
boolgravitationMomentQuatemionDepend;
boolmagnetMomentQuatemionDepend;
private:int systemLog_CannotCalculate;
);
```

Функція Calculate написана у відповідності до вимог, що поставлені до класу ExterForce (див. опис класу ExterForce). Клас містить наступні додаткові поля.

modelQuatemion - Покажчик на модель з кватерніонами;
gravitationMomentQuatemionDepend - Якщо true, то гравітаційний момент буде враховано;
magnetMomentQuatemionDepend - Якщо true, то магнітний момент буде враховано;
systemLog_CannotCalculate - *Сервісна змінна.*

5. Модуль обчислення внутрішніх сил

```
classlnterForce (lnterForce( class Control*);
virtual lnterForce();
virtual int Calculate( class Model*);
class Control *control;
);
```

Даний клас призначений для обчислення впливу внутрішніх сил (керування), що діють на об'єкт.

virtual int Calculate ();

Функція призначена для заповнення відповідних полів поточного статусу моделі Ейлера. Повертає 0 при нормальному завершенні.

Клас містить наступне поле.

control - Покажчик на клас, що реалізує функцію керування.

6. Функції керування

Функції керування об'єктами реалізовуються за допомогою спеціальних класів.

Базовий клас.

```
classControl(Control( int d = 1);
Control( int,double*);
virtual ~Control();
virtual int Calculate( double *);
int dimension;
double *level;
);
```

Для реалізації функції керування передбачено даний клас. Конструктор дозволяє резервувати задану кількість елементів у векторі level (по замовчуванню резервує 1 елемент). Передбачено також резервування заданої кількості елементів у векторі level, який буде визначено значеннями згідно другого аргументу при використанні альтернативного конструктора з двома аргументами. Клас містить наступні поля.

dimension - Розмірність вектора керування;

level - Значення вектора керування.

Функція для отримання значень керування має наступне представлення.

virtual int Calculate(double *u)

Ініціалізація вектора u згідно значень вектора level. Передбачається, що розмірність u не перевищує розмірність level. Повертає 0 у випадку успішного завершення.

Структурне керування.

```
classControlDirectFunction        (
ControlDirectFunction();
virtual ~ControlDirectFunction();
virtual int Calculate( struct Status*, double*);
);
```

Для реалізації структурного керування доцільно використовувати даний клас. Тоді у тіло функції Calculate необхідно внести відповідні формули для обчислень значень функцій згідно заданого статусу. При цьому у конструкторі необхідно зафіксувати відповідне значення для поля dimension.

Релейне керування.

```
classControlBangGeneralized(
ControlBangGeneralized();
virtual ~ControlBangGeneralized();
virtualint Calculate( double*);
struct ResultChain *controlHead;
double u1, u2, u3, u4;
double step;
);
```

Для обчислення релейних багатопозиційних програмних функцій керування розроблений даний клас. Конструктор проводить всі необхідні обчислення, результатом яких є ланцюг результатів (на який вказує controlHead). Клас має наступні поля.

controlHead - Покажчик на ланцюг результатів;

u1, u2, и3, u4 - Рівні керування;

step - Крок між точками переключення.

Функція Calculate написана у відповідності до вимог, що поставлені до класу Control (див. опис класу Control).

7. Розв'язування задачі Коші

Базовий клас інтегрування (метод Ейлера).

```
classIntegration(
lntegration( char* ini-fiIe, int k);
virtual ~Integration();
virtual ResultChain* Go( class Model **arrayModel, struct Status *s0,
```

structStatus *s1, doublestep=0, resCalc=0, int printldentify=0);
ResultChain Go(class Model *model, stЛAt Status *s0,*
*Status *s1, double step=0, resCalc=0, int printldentify=0);*
ResultChain resultEnd;
);

Конструктор дозволяє резервувати задану кількість елементів k в полі r змінної поля resultEnd (по замовчуванню резервує 25 елементів). Поле resultEnd містить результати розв'язування задачі Коші в кінцевій точці інтегрування. Наступна функція призначена для інтегрування систем диференціальних рівнянь методом Ейлера.

virtual ResultChain* Go(class Model **arrayModel, stЛAt Status *s0,
struct Status *s1, double step=0, ResultManager *resCalc=0, int printldentify=0);
arrayModel - вектор моделей;
s0, s1 - відповідно початковий та фінальний статуси;
step - крок інтегрування;
resCalc - накопичення результатів (якщо 0- проміжні результати не накопичуються);
printldentify - ідентифікатор індикації процесу;

Система диференціальних рівнянь має бути визначена в класі Model. Із елементів цього класу утворюється вектор arrayModel, причому останній елемент цього вектора має бути NULL. Ідентифікатор printldentify призначений для керування індикації процесу інтегрування. Результати обрахунків запам'ятовуються у resCalc. Функція не здійснює запис результатів у файл, тому resCalc необхідно відконфігурувати належним чином (див. опис класу ResultManager). Допускаються також наступні значення полів :

step=0 - крок покладається рівним всьому проміжку інтегрування; resCalc=0 - проміжні результати не запам'ятовуються;

По замовчуванню :

step=0, resCalc=0, printldentify=0.

Функція повертає покажчик на структуру ResultChain, в якій зберігаються результати обчислень в початковий момент часу. Значення в кінцевий момент часу можна отримати з поля resultEnd.

Для роботи необхідні глобальна структура інтерфейсу Interface *INTERFACE.

Для інтегрування тільки одної моделі передбачено функцією

virtual ResultChain* Go(class Model *model,
struct Status *s0, stЛAt Status *s1, double step=0, ResultManager *resCalc=0, int printldentify=0);

Вхідні параметри (за виключенням першого) та вихідні значення ідентичні попередній функції.

Поля статусів s0, s1, які не є необхідними для інтегрування, ігноруються.

Метод Рунге-Кутта-Фельдберга 7(8).

```
classlntegrationRKF78      (
lntegrationRKF78( char* ini-file, int k);
IntegrationRKF78();
Go(arrayModel, struct Status *s0,
s1, doublestep=0, resCalc=0, int printldentify=0);
double eps;
double tolerance;
int routine;
private:
double ch[13], alph[13], beta[13][12];
);
```

Опис основних конструкцій ідентичний класу Integration. Додаткові поля містять параметри методу Рунге-Кутта-Фельдберга.

eps - Точність виходу з методу;

tolerance - Умова збіжності;

routine - Якщо 0 : обчислення з 8 порядком, оцінка з 7, якщо 1 : навпаки.

ch[13], alph[13], beta[13][12] - Параметри методу.

8. Організація інтерфейсу

В процесі роботи головна програма працює з буферами інтерфейсу за допомогою спеціальних класів, розробленому на ПО MATLAB. Для цього зарезервовано змінну INTERFACE - покажчик на базовий клас інтерфейсу Interface:

```
class Interface INTERFACE;
```

Ініціалізація має проводитися у відповідності до обраного інтерфейсу (неможливо ініціалізувати абстрактний інтерфейс за допомогою базового класу Interface). Доступні 3 стандартні спрощені консольні інтерфейси:

- INTERFACE = new lnterfaceConsole(int tUpdate, char *f_wow = "", char *f_path = "");

повноекранний консольний (типу IBM 25*80) інтерфейс;

- INTERFACE = new lnterfaceMiniConsole(intUpdate, char*f_wow="", char*f_path="");

стиснутий консольний інтерфейс (без порожніх стрічок);

- INTERFACE = new lnterfaceLine(int tUpdate, char *f_wow = "", char *f_path = "");

послідовний вивід тільки нових стрічок (без порожніх стрічок); та 2 повні мережеві:

INTERFACE = new lnterfaceMATLAB(char *f_out, char *f_path = "", int tUpdate = 1,

char *f_in = "", char *f_wow = ""); - повнофункціональний інтерфейс для використання модулю MATLAB remote console (дистанційна консоль);

INTERFACE = new lnterfaceTxt(char *f_out, char *f_path = "", int tUpdate = 1, char *f_in = "", char *f_wow = ""); - використання текстових файлів у якості вихідного буферу;

tUpdate - частота оновлення (в секундах);

f_path - шлях до буферів інтерфейсу;

f_wow- ім'я буферу сервісу переривань;

f_out- ім'я вихідного буфера;

f_in - ім'я вхідного буфера.

Для виводу інформації передбачені наступні функції.

void INTERFACE::Print(char *msg="", int identify=0, int mod=0); - *вивід тексту;*

void INTERFACE::PrintBlank(int identify=0); - *вивід порожньої стрічки (стирання);*

void INTERFACE::PrintBlankAII(); - *стирання всього тексту;* void INTERFACE PrintNumber(Tn, intidentify=0, intmod=0); - *вивід числа;*

void INTERFACE::PrintNumber(char *msg="", T n, int identify=0, int mod=0 y, - *вивід числа із текстом;*

void INTERFACE::PrintProcess(char *msg="", int identify=0); - вивід підписаного текстом динамічного значка, що свідчить про роботу програми;

void INTERFACE::PrintPercent(char *msg="", T n, int identify=0); - *вивід відсотків із підписом;*

msg-форматована згідно ANSI C++ стрічка із повідомленням, наприклад,

INTERFACE::PrintNumber("зробпено %d ітерацій", 15,1); надрукує у стрічку з ідентифікатором 1 інформацію : зроблено 15 ітерацій

identify - ідентифікатор (позиція виводу - зазвичай номер стрічки на екрані);

mod - режим виводу :

APPEND - приєднати до існуючої стрічки (з тим самим ідентифікатором);

REMARK- вивести як коментар;

WARNING - вивести як попередження;

ERROR-вивести як помилку;

n - при підстановці цілого невід'ємного числа добавляється до існуючої стрічки (з тим самим ідентифікатором) починаючи з позиції n;

по замовчуванню - нова стрічка повністю перекриває стару (з тим самим ідентифікатором), тобто n=0; T n - числова величина типу T (int чи double).

Для вводу інформації передбачені наступна функція.

virtual T INTERFACE::Read(char *msg, T &n);

msg - стрічка із запитанням;

T - тип змінної, що вводиться (int, double чи char*).

n - змінна типу T, яка отримує введене значення.

Сервіс переривань реалізується функцією int lnterrupt(); яка повертає номер переривання, або 0 - якщо переривання не отримано.

Примусове оновлення (незважаючи на параметр tUpdate) здійснюється за допомогою

virtual void Update();

Додаткові функції та параметри :

const double version; - номер версії у форматі XX.XX; staticintMaxOutputLines(); - повертає максимальну кількість стрічок, що виводиться;

static int MaxLineLength(); - повертає максимальну довжину стрічки.

с. Для вибору інтерфейсу передбачено наступну функцію.

int SetSpacelnterface()

Ініціалізує інтерфейс згідно значення, що міститься у файлі space.ini у змінній з іменем „face" у секції [INTERFACE].

9. Протокол роботи

Для ведення статистики роботи розроблено класс classSystemLog та зарезервовано змінну class SystemLog *LOG = new SystemLog(f_log);

f_log - ім'я файлу, куди записуються повідомлення.

Використання :

int LOG::Add(char *message="WOW", bool clear=false);

static int SystemLog::Write(char *fileName, char *message="WOW", bool clear=false);

message - текст повідомлення;

clear - якщо false, то повідомлення буде добавлено в кінець файлу, якщо true, то вміст файлу витирається;

fileName - ім'я файлу, куди буде записано повідомлення; повертає 0 при відсутності помилок.

При веденні статистики автоматично додається час, коли повідомлення надійшло.

10. Представлення результатів

Файл, що отримується при запам'ятовуванні ланцюга результатів має текстовий тип та може бути переглянутий текстовим редактором. Він має наступну структуру

Dimension x_0 y_0^0 y_0^1 ... y_0^n x_1 y_1^0 y_1^1.... y_1^n....

comments - коментарі (довільна кількість стрічок, що починаються з символу „;");

Dimension - розмірність блоку інформації про точку (абсциса та ординати);

x_i , y_i^0, y_i^1,..., y_i^n - інформації про точку номер і (абсциса та ординати).

11. Додаткові функції

Інкремент числа з довільною основою

int Cortege(int *c, int dimension, int level, int i=0)

c - Вектор цілих чисел, який буде збільшено на **1;**

dimension - Розмірність вектора c;
level - Основа числення;
i - Службова змінна.

Повертає **1,** якщо число перевищує задану розрядність. При виклику функції забороняється змінювати значення службової змінної i, тобто виклик необхідно здійснювати тільки від 3 аргументів :

Cortege(c, dimension, level).

12. Розв'язування оптимізаційних задач

Оптимізуючий програмний комплекс включає в себе спеціальні модулі, які створені за допомогою описаного вище інструментарію. Вони направлені на розв'язування цілого ряду задач, що виникають при моделюванні динаміки руху ЛА.

Моделювання рівнянь Ейлера з кватерніонами з можливістю враховування чи ігнорування впливу зовнішніх сил та завдання довільного керування у вигляді загального впливу, що діє на ДО. Вхідні параметри задаються у іні-файлі та можуть бути змінені без перекомпіляції модуля.

Моделювання рівнянь Ейлера з кватерніонами для ДО з 4 ЕМД. Вхідні параметри задаються у іні-файлі та можуть бути змінені у будь-який час.

Модуль розрахунку оптимального за швидкодією синтезуючого керування по переведенню ДО із довільного початкового стану на Е - сферу.

Модуль розрахунку квазіоптимального керування для задачі переведення ЛА із початкового стану в кінцевий за найшвидший час.

Модуль розрахунку керування по утримуванню ЛА навколо заданого режиму функціонування.

Модуль пошуку релейного багатопозиційного керування, що переводить динамічний об’єкт із початкового стану в кінцевий за найшвидший час.

1. *Азарсков, Сущенко О.А.* Експериментальні випробування та дослідження систем, К.: НАУ, 2003, 268с.
2. *Башлыков А.А.* Проектирование систем принятия решений в энергетике. – М.: Энергоатомиздат, 1986. – 120 с.
3. *Бєляєв Ю.Б., Шмельова Т.Ф., Сікірда Ю.В.* Моделювання процесу прийняття рішень оператором авіаційної ергатичної системи в особливих випадках польоту / Автоматизація виробничих процесів. – 2003. - №2(17). – с. 17-23.
4. *Борисов А.Н., Алексеев А.В., Крумберг О.А. и др.* Модели принятия решений на основе лингвистической переменной. – Рига.: Зинатне, 1982. – 256 с.
5. *Гаврилова Т.А., Хорошевский В.Ф.* Базы знаний интеллектуальных систем. – СПб: Питер, 2001. – 384 с.
6. *Герасимов Б.М., Дивизинюк М.М., Субач И.Ю.* Системы поддержки принятия решений : проектирование, применение, оценка эффективности, Севастополь, СНИЯЭ и П,2004,320с.
7. *Заде Л.* Понятие лингвистической переменной и ее применение к принятию приближенных решений. – М.: Мир, 1976. – 167 с.
8. *Козелков С.В., Машков О.А., Барабаш О.В.* Методика побудови функціонально –

стійких розподілених інформаційних систем / Матеріали науково – технічної конференції „Сучасний стан та проблеми авіаційної техніки Військово – Повітряних Сил”, НЦ ВПС, 10-11 червня 2004р.
9. *Козелков С.В., Машков О.А., Барабаш О.В.* Побудова функціонально-стійких розподілених інформаційних систем наземного базування / Тез. доп. XIV науково-технічної конференції, ЖВІРЕ ім. С.П.Корольова, Житомир, 2004, с. 107-108.
10. *Машков О.А.* Концепции построения функционально-устойчивых информационно-управляющих комплексов / Тезисы докладов 6-й Всесоюзной конференции. Ч.II. – К.: АН УССР, 1991, с. 50-51.
11. *Самохвалов Ю.Я.* Декомпозиция логико-лингвистических моделей принятия решений в распределенной вычислительной среде // Кибернетика и системный анализ – 1997. - № 1. – С. 57-65.
12. *Субач І.Ю., Соколов В.В.* Організація баз даних та знань. – К: КВІУЗ, 1999.
13. *Трахтенгерц Э.А.* Компьютерная поддержка принятия решений. – М.: СИНТЕГ, 1998. – 376 с.
14. *Уотермен Д.* Руководство по экспертным системам. Пер. с англ. – М.: Мир, 1989. – 388 с.

Поступила 27.09.2010р.

УДК 683.03

О.Ю.Афанасьєва, Б.В.Дурняк, Ю.М.Коростіль

АНАЛІЗ ФАКТОРІВ, ЩО ВПЛИВАЮТЬ НА РИЗИК ЗНИЖЕННЯ РІВНЯ БЕЗПЕКИ ТЕХНІЧНОГО ОБ'ЄКТУ

В рамках системи безпеки (SB) повинні розв'язуватися наступні задачі:

- задача прогнозування виникнення несправностей,
- задача оцінки рівня безпеки функціонування технічного об'єкту (TO) , що може мінятися у зв’язку з виникненням несправності,
- задача моніторування і діагностикии несправностей, що передбачає визначення причини її виникнення,
- задача протидії розвитку несправності, або протидії причинам, що обумовили виникнення несправності,
- у випадку виникнення аварійної ситуації, SB повинна протидіяти розвитку аварії та ініціювати дії, що пов’язані з захистом оточуючого середовища від негативних факторів, що ініційовані аварією та діють на оточуюче середовище.

Приведені задачі стосуються проектних і непроектних несправностей N^P, N^N, відповідно. По відношенню до N^P і N^N розв'язки цих задач будуть різні, оскільки закономірності виникнення різних типів несправностей є різними. В даному випадку, будемо розглядати таку різницю лише на рівні

