ISSN 2224-087X. Електроніка та інформаційні технології. 2012. Випуск 2. С. 184–196
Electronics and information technologies. 2012. Issue 2. P.184–196

УДК 519.622.2

РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ ДЛЯ МОДЕЛЮВАННЯ ДИНАМІЧНИХ СИСТЕМ ІЗ ЖОРСТКИМИ МАТЕМАТИЧНИМИ МОДЕЛЯМИ

Я. Кость, І. Хвищун

Львівський національний університет імені Івана Франка
вул. Тарнавського, 107, 79017 Львів, Україна
kostjerry@gmail.com

Сформульовано основні поняття математичного моделювання динамічних систем. Визначено етапи процесу моделювання. Наведено означення похибок, які виникають у процесі математичного моделювання. Конкретизовано поняття стійкості розв'язку диференційного рівняння та стійкості числового методу. Описано розроблене програмне забезпечення для моделювання динамічних режимів систем із жорсткими математичними моделями.

Ключові слова: математичне моделювання, жорсткість, стійкість, локальна похибка, глобальна похибка, числовий метод, програмне забезпечення, задача Коші.

Упродовж останніх десятиліть постійну увагу приділяють математичному моделюванню динамічних режимів технічних пристроїв і систем. Значні зусилля науковців зосереджено в напрямі дослідження рівнянь математичних моделей (ММ) систем, які мають властивість жорсткості, оскільки подібні системи найскладніші з погляду їхнього розв'язування. Сьогодні у світі розроблено чимало математичного, алгоритмічного та програмного забезпечення (ПЗ), загального чи спеціалізованого, яке здатне успішно інтегрувати жорсткі задачі. Сюди належать такі програмні комплекси, як Matlab, Mathcad, Octave, SciLab, Maple, Mathematica та ін. Найвідоміше ПЗ для моделювання радіоелектронних систем, математичні моделі яких є жорсткими, – система Pspice [1].

Для ефективного використання названих систем під час розв'язування практичних задач недостатньо вміти користуватись інтерфейсом користувача того чи іншого ПЗ. Необхідно добре розуміти особливості числових методів, використаних для математичного моделювання, їхні можливості, галузі оптимального застосування та діапазони значень параметрів, за допомогою яких конкретний метод можна налаштувати для розв'язування необхідної задачі.

Нижче наведено принципи побудови та архітектуру програмного комплексу для дослідження особливостей математичного моделювання систем із жорсткими ММ і сформульовано основні теоретичні поняття, які стосуються цього розділу науки.

Відомо, що процес математичного моделювання динамічних режимів деякої фізичної системи можна умовно розділити на три етапи [2]:

1) побудова її ММ у вигляді системи звичайних диференціальних рівнянь (ЗДР) у канонічній формі Коші:

$$\dot{\bar{x}} = \frac{d\bar{x}}{dt} = \bar{f}(\bar{x}, t), \quad (1)$$

або у неявній формі:

$$\bar{F}(\bar{x}, \dot{\bar{x}}, t) = 0, \quad (2)$$

тут $\bar{x}, \bar{f}, \bar{F}$ – вектор-функції;

2) числове інтегрування рівняння (1) чи (2) на певному проміжку часу $[t_0, t_k]$, тобто визначення сім'ї кривих, які є шуканими розв'язками. Якщо задати початкові умови $\bar{x}(t_0) = \bar{x}_0$, то матимемо розв'язання так званої задачі Коші [3–5];

3) аналіз розв'язків для отримання певних характеристик поведінки системи.

З теорії диференціальних рівнянь відомо, що задача Коші має єдиний розв'язок, якщо функція $f(x,t)$ є визначеною і неперервною на інтервалі $[t_0, t_k]$ і задовольняє на ньому *умову Ліпшиця* [4]: $|f(x_2,t) - f(x_1,t)| \le L \cdot |x_2 - x_1|$, де L – скалярна величина, яку називають *константою Ліпшиця*. Вона є верхньою границею норми *матриці Якобі* системи (1): $\left\| \frac{\partial f}{\partial x} \right\| \le L$.

Головні вимоги, яким повинні відповідати ММ фізичних об'єктів [2]:

1) ММ повинна максимально чітко і вичерпно описувати закони взаємодії її елементів між собою, а також правильно враховувати реакцію об'єкта на зовнішні збурення;
2) до ММ застосовують вимоги точності, економічності й універсальності:
   - *точність* – це характеристика моделі, яка полягає в найповнішому якісному та кількісному відображенні нею дійсних властивостей об'єкта;
   - *економічнішою* є та модель, яка для досягнення заданої точності потребує менших затрат ресурсів комп'ютера;
   - під *універсальністю* ММ розуміють можливість її застосування до широкого класу об'єктів.

Похибки, що виникають під час математичного моделювання

Глобальною похибкою (*global error*) у точці t_{n+1} називають різницю між точним та обчисленим значеннями функції $x(t)$ у цій точці [6]: $e_{n+1} = x(t_{n+1}) - x_{n+1}$.

Локальною похибкою усікання (*local truncation error*) називають різницю між точним розв'язком $x^*(t)$, який зміщений так, щоб проходити через попередню обчислену точку x_n, у точці t_{n+1} та обчисленим значенням x_{n+1}: $T_{n+1} = x^*(t_{n+1}) - x_{n+1}$.

Із зазначеного вище випливає таке: якщо $x_n = x(t_n)$, тобто на попередньому кроці числовий розв'язок точно відповідав аналітичному, то локальна похибка усікання дорівнює глобальній похибці.

Суму локальних похибок усікання на всіх попередніх кроках називають *глобальною похибкою усікання* (*global truncation error*).

Під час виконання розрахунків над даними з плаваючою комою на електронно-обчислювальних машинах (ЕОМ) виникає ще один вид похибок, пов'язаний зі скінченністю розрядної сітки ЕОМ. Такі похибки називають *похибками заокруглення* (*round-off errors*).

Позначимо через δ_n похибку заокруглення на n-му кроці інтегрування системи (1). Вона може складатись із двох частин:

- заокруглення під час уведення значень початкових умов;
- заокруглення, що виникають унаслідок виконання арифметичних операцій.

Зазначимо, що глобальна похибка заокруглення не дорівнює сумі локальних похибок заокруглення на всіх попередніх кроках, оскільки δ_n може виникати внаслідок як заокруглення вгору, так і заокруглення вниз.

У літературі [6] зазначають, що значення глобальної похибки залежить від кроку інтегрування h і ця залежність має один мінімум. Чим менше h, тим більша похибка δ_n. З іншого боку, зі зменшенням h зменшується локальна похибка усікання T_n.

Для процесу моделювання важливим є поняття *збіжності* числового методу. Дж. Холл [4] визначає його так: числовий метод збігається, якщо виконується рівність $\lim\limits_{h\to 0} e_n = 0$ або $\lim\limits_{h\to 0} x_n = x(t_n)$.

Стійкість

Як відомо [7], систему (1) можна локально лінеаризувати і звести до вигляду $\dot{X} = AX$, де A – матриця Якобі $\left[\dfrac{\partial f_i}{\partial x_j}\right]$ розмірності $[N \times N]$.

Якщо $\lambda_1, \lambda_2, ..., \lambda_N$ є множиною власних значень матриці A, то існує ортогональна матриця H така, що $H^T AH = \Lambda = diag\{\lambda_1, \lambda_2, ..., \lambda_N\}$. Зробивши заміну $X = Hx$, отримаємо

$$H\dot{x} = AHx. \tag{3}$$

Помножимо (3) на H^T зліва, отримаємо $\dot{x} = \Lambda x$. Тобто ми маємо N рівнянь вигляду $\dot{x}_i = \lambda_i x_i$, де λ_i в загальному випадку є комплексним числом. Отже, для дослідження числових методів, без втрати загальності, можна використовувати таке *модельне диференціальне рівняння* [6]: $\dot{x} = \lambda x$, де λ відповідає $\dfrac{\partial f}{\partial x}$ у випадку одного рівняння або власним значенням матриці Якобі у випадку системи диференціальних рівнянь.

Розрізняють стійкість розв'язків диференціального рівняння та стійкість числового методу інтегрування. Диференціальне рівняння є *стійким*, якщо для всіх λ_i виконується умова [6]:

$$\mathrm{Re}(\lambda_i) < 0. \tag{4}$$

На рис. 1 проілюстровано поведінку розв'язків нестійкої та стійкої систем, відповідно. Тут зображено інтегральні криві, з-поміж яких виділена крива $x(t)$, яка відповідає шуканому розв'язку.

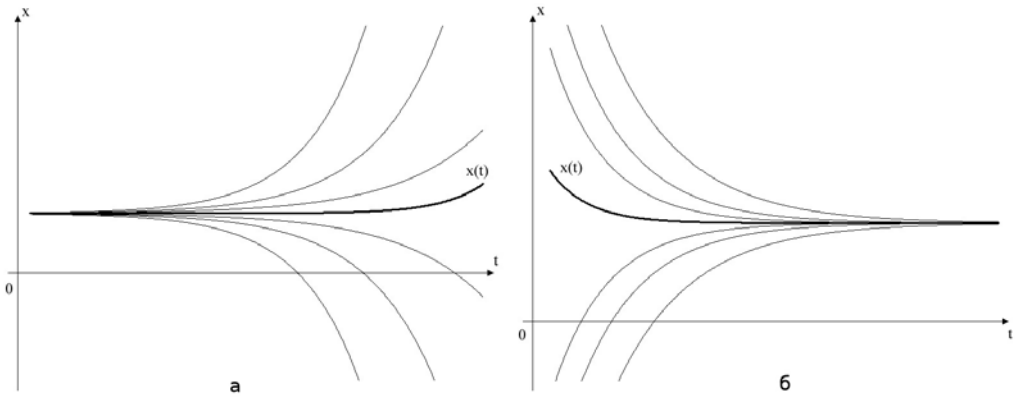

Рис. 1. Поведінка розв'язків нестійкої (*а*) та стійкої (*б*) систем диференціальних рівнянь.

У випадку нестійкої системи диференціальних рівнянь незначна похибка переведе числовий метод на сусідню інтегральну криву, що спричинить розбіжність процесу інтегрування. Якщо ж умова (4) виконається, то навіть коли числовий метод спочатку рухатиметься по хибній інтегральній кривій, зрештою, прийде до розв'язку.

Стійкість числових методів прийнято досліджувати на основі модельного диференціального рівняння $\dot{x} = \lambda x$, аналітичним розв'язком якого є функція $x(t) = e^{\lambda t}$.

Наприклад, умова стійкості явного методу Ейлера (ЯМЕ) задана виразом [2] $h < \dfrac{2}{\lambda_{\max}}$, де $\lambda_{\max} = \max\{|\lambda_1|, |\lambda_2|, ..., |\lambda_N|\}$. На рис. 2 показано результати процесу інтегрування за допомогою ЯМЕ рівняння $\dot{x} = -10x, x_0 = 1$. У цьому випадку обмеження на крок таке: $h < 0.2$. На рис. 2 зображено розв'язки цього рівняння з кроками $h = 0.01$, $h = 0.18$ та $h = 0.21$.

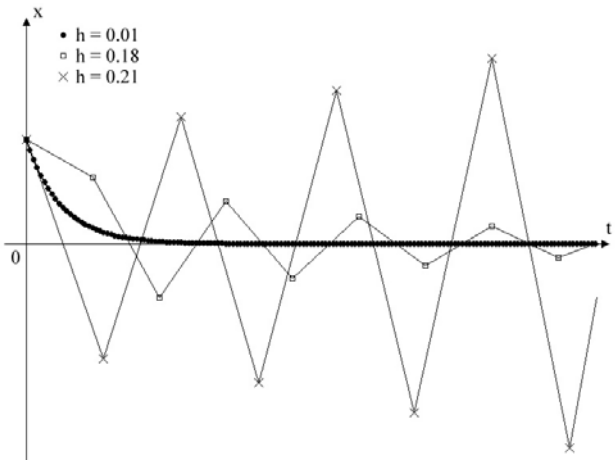

Рис. 2. Розв'язки модельного рівняння з різними кроками інтегрування.

Із наближенням до краю області стійкості виникають коливання розв'язку, які, проте, з часом загасають і приводять до шуканого результату. За межею стійкості похибка наростає і числовий метод не знаходить розв'язку. Область стійкості ЯМЕ є колом на площині $h\lambda$ з центром у точці $(-1;0)$ та радіусом 1.

Якщо метод стійкий на всій комплексній півплощині $\mathrm{Re}(h\lambda) < 0$, то його називають *A-стійким* [4,6]. Поняття *A*-стійкості використовують для якісної оцінки числових методів інтегрування.

Дж. Далквіст довів [8], що неявні лінійні багатокрокові методи, які мають порядок точності більше 2, не можуть бути *A*-стійкими. Отже, найточнішим *A*-стійким методом є метод трапецій (метод Адамса–Маултона другого порядку).

З огляду на це для чіткішої характеристики методів було запропоновано поняття *A*(α)-*стійкості* [9]. Метод є *A*(α)-стійким, якщо його область абсолютної стійкості містить $\{h\lambda : -\alpha < \pi - \arg(h\lambda) < \alpha\}$ для $\alpha \in \left(0;\frac{\pi}{2}\right)$.

Поряд з *A*(α)-стійкістю існує поняття *A*(0)-*стійкості*. Метод є *A*(0)-стійким, якщо він *A*(α)-стійкий для деякого достатньо малого $\alpha \in \left(0;\frac{\pi}{2}\right)$.

У [10] С. Гір увів поняття *жорсткостійкості*. Метод є жорсткостійким, якщо він абсолютно стійкий в області $\{h\lambda : \mathrm{Re}(h\lambda) \le -a\}$ і точний в області $\{h\lambda : -a < \mathrm{Re}(h\lambda) < b, -c < \mathrm{Im}(h\lambda) \le c\}$, де a, b, c – додатні константи.

Жорсткість математичної моделі

Загалом жорсткість є досить нечітким поняттям. У різних літературних джерелах трапляються різні визначення [4–6, 11, 12]. Узагальнивши їх, наводимо таке формулювання. Математичну модель можна вважати *жорсткою*, якщо виконуються умови:

1) $S = \dfrac{\lambda_{\max}}{\lambda_{\min}} >> 1$, де $\lambda_{\max} = \max\{|\lambda_1|, |\lambda_2|, ..., |\lambda_N|\}$, $\lambda_{\min} = \min\{|\lambda_1|, |\lambda_2|, ..., |\lambda_N|\}$. Величину S називають *коефіцієнтом жорсткості*;
2) $\mathrm{Re}(\lambda_i) < 0$. Як зазначено вище, ця умова є умовою стійкості системи диференціальних рівнянь.
3) $(b-a)\lambda_{\max} >> 1$, де $[a;b]$ – інтервал, на якому відбувається інтегрування. Тобто проміжок інтегрування має бути набагато більшим, ніж тривалість перехідних процесів у системі.

У літературі [6] стверджують, що система є жорсткою, якщо під час її інтегрування значення кроку обмежене не точністю, а стійкістю. Проте якщо інтегрування відбувається на дуже короткому проміжку часу, то системи з великим коефіцієнтом жорсткості вже не є жорсткими, оскільки поняття жорсткості свідчить про необхідність великих затрат обчислювальних ресурсів, а у наведеному випадку така необхідність зникає.

Найменш придатні для інтегрування жорстких систем явні методи. Розглянемо, наприклад, ЯМЕ. Величина λ_{max} задає обмеження зверху на крок інтегрування, а λ_{min} визначає тривалість перехідних процесів у системі. Отже, оскільки $h < \dfrac{2}{\lambda_{\max}} = \dfrac{2}{S\lambda_{\min}}$, то чим

більше S (чим жорсткішою є система), тим меншим має бути крок інтегрування. З міркувань точності після закінчення швидких процесів крок можна було б збільшити. Проте умова стійкості не дає змоги цього зробити. Тому явними методами доводиться роботи дуже велику кількість кроків.

Для розв'язування жорстких задач зазвичай використовують неявні методи, побудовані за схемою прогнозу та корекції, оскільки їхні області стійкості набагато ширші, ніж явних методів.

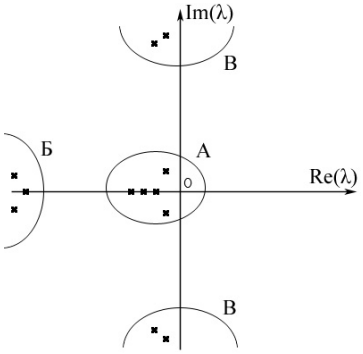

Рис. 3. Різні випадки систем, які трапляються в моделюванні.

На рис. 3 на комплексній площині λ з погляду на жорсткість систем зображено різні випадки, які трапляються в моделюванні [2].

1. ***Область А***. Цей випадок є найліпшим для моделювання. Таку систему можна легко проінтегрувати явними методами.
2. ***Області А+Б***. Цей випадок відповідає жорсткій системі, для розв'язування якої необхідно використати неявні методи.
3. ***Області А+В***. Цей випадок найгірший з погляду моделювання. Система є жорсткою з осцилювальними функціями-розв'язками. Перехідний процес у таких системах іноді триває впродовж десятків, а то й сотень квазіперіодів коливань осцилювальної функції. Інтегрування кожного з цих квазіперіодів потребує декількох сотень кроків h, що, відповідно, призводить до величезних затрат процесорного часу.
4. Якщо власні числа λ розташовані праворуч від уявної осі, то така система є нестійкою і промодельованою бути не може.

Опис розробленого програмного забезпечення

Короткий огляд використаних технологій. Проект написано об'єктно-орієнтованою мовою програмування *C#* у середовищі розробки *Visual Studio 2010*. Інтерфейс користувача розроблено з використанням технології *Windows Presentation Foundation* (*WPF*) [13]. Для обчислення власних значень матриці Якобі застосовано стандартну бібліотеку *alglib.net* [14], а для візуалізації результатів – бібліотеку *Dynamic Data Display* (*DDD*) [15]. На рис. 4 зображено графік, побудований за допомогою зазначеної бібліотеки.

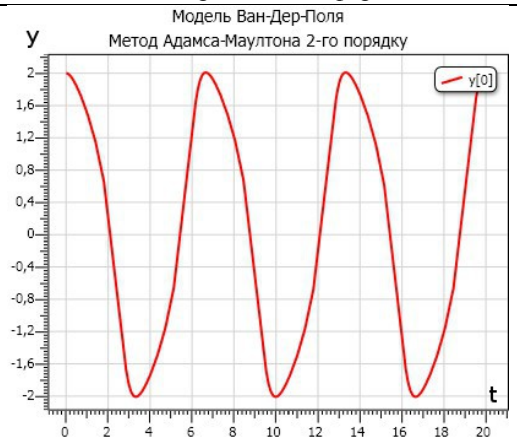

Рис. 4. Графік, побудований за допомогою бібліотеки DDD.

Архітектура проекту. Однією з особливостей задач моделювання є те, що процес інтегрування та ММ ніяк не пов'язані між собою. Будь-який метод інтегрування можна застосувати до будь-якої моделі. Інша річ, що не будь-яка комбінація метод–модель дасть правильний результат. З огляду на зазначені особливості задач моделювання, опис методу інтегрування та опис ММ виконано в окремих класах.

У проекті реалізовано різні методи інтегрування та ММ. Відтак створено два абстрактні батьківські класи: *Intergator* та *Model*. Відомо, що від абстрактних класів об'єкти створити неможливо. Класи *Intergator* та *Model* можна використати лише як класи-предки. Очевидно, що у них зібрані всі риси, притаманні, відповідно, методам інтегрування та ММ. Тому завдяки принципу поліморфізму стає можливим робота інтегратора із заздалегідь невизначеною ММ, яка, проте, мусить бути нащадком класу *Model*.

На рис. 5 зображено спрощену UML-діаграму архітектури проекту. Класи із суфіксом *Integrator* є конкретними реалізаціями методів інтегрування. Аналогічно, класи із суфіксом *Model* є реалізаціями ММ.

Клас *Tab* відіграє роль сполучника, який з'єднує інтерфейс користувача з класами *Integrator* та *Model*.

На діаграмі також показано найголовніші допоміжні класи:

- *Integrators* – клас, де зареєстровано реалізовані методи інтегрування. За його допомогою на головній вкладці програми користувач має змогу вибрати певний інтегратор для подальшого дослідження;
- *Models* – клас, аналогічний до класу *Integrators*, призначений для реєстрації ММ;
- *Methods* – клас, у якому реалізовано метод Гауса для розв'язування систем лінійних алгебричних рівнянь (СЛАР) та метод для числового знаходження елементів матриці Якобі ММ;
- *Stiffness* – клас, у якому реалізовано визначення коефіцієнта жорсткості ММ;
- *GraphicsHelper* – клас для виведення результатів моделювання на графіки.

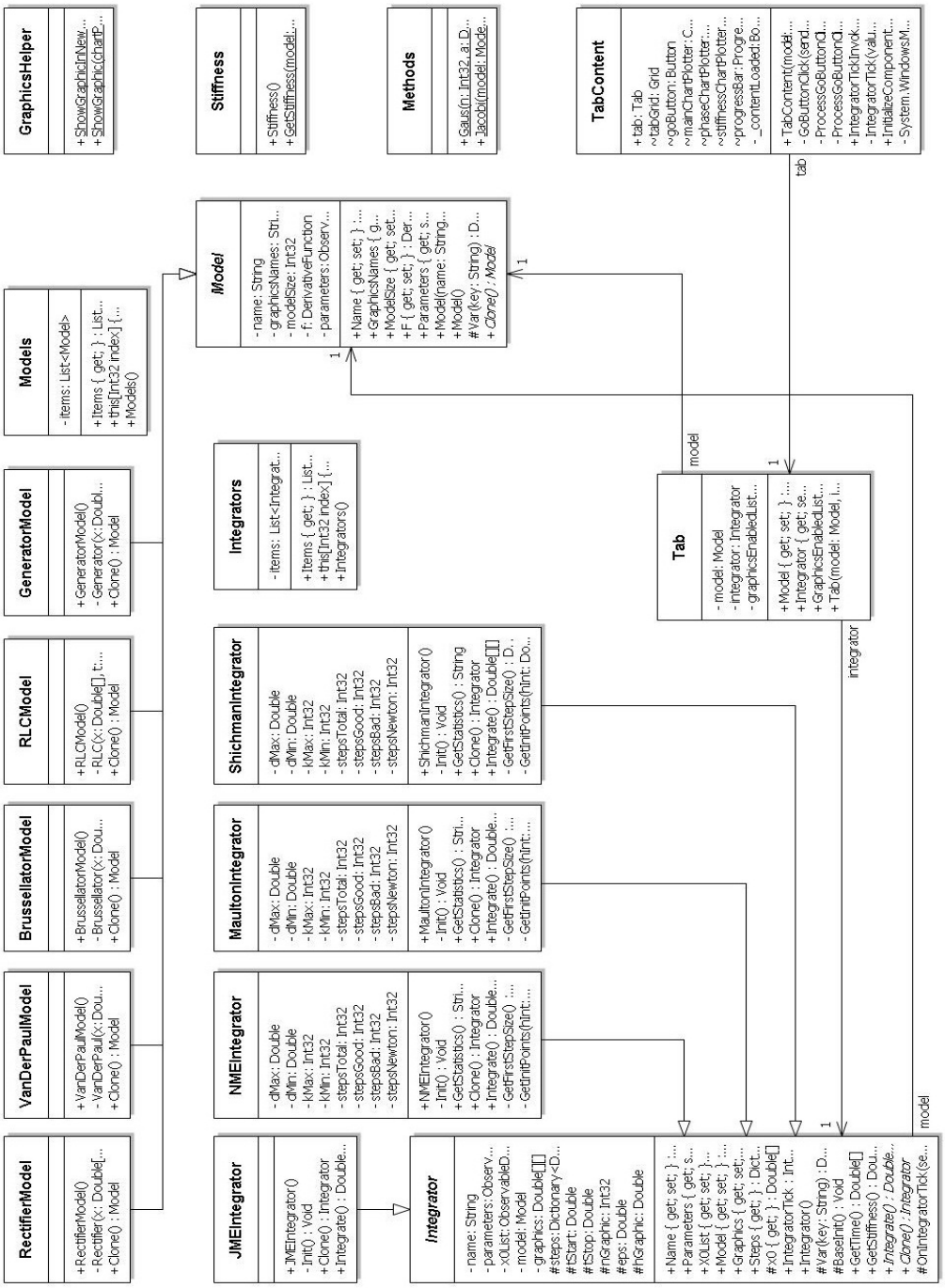

Рис. 5. Спрощена UML-діаграма проекту

Інтерфейс користувача. Інтерфейс користувача побудовано у вигляді вкладок, одна з яких є головною, а інші – робочими (їх можна створювати або закривати). На кожній із робочих вкладок зафіксовано конкретний метод інтегрування та конкретну математичну модель, а також надано можливість налаштовувати та виконувати процес інтегрування. Робочі вкладки можна створювати динамічно з головної (рис. 6).

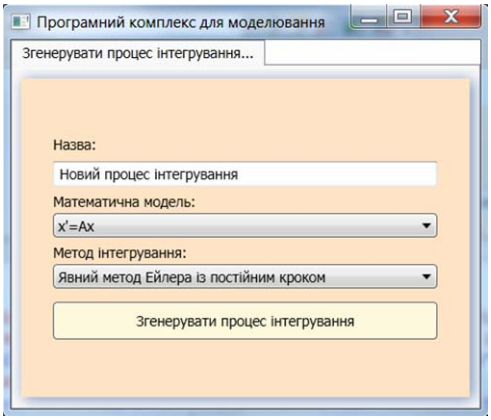

Рис. 6. Вигляд головної вкладки проекту.

На рис. 7 зображено робочу вкладку після інтегрування моделі Лоренца методом Шичмена [2].

Як бачимо з рис. 7, інтерфейс користувача розділено на чотири частини:

1) параметри інтегрування, параметри ММ та початкові умови;
2) часові залежності шуканих функцій (правий верхній графік);
3) фазовий портрет системи;
4) часова залежність зміни коефіцієнта жорсткості.

Особливості обчислень. У нашому проекті матрицю Якобі обчислюємо так:

$$Ja_{i,j} = \frac{f(x_j + q)_i - f(x_j)_i}{q},$$

де q – достатньо мале збурення відповідної компоненти вектора правої частини рівнянь моделі.

Коефіцієнт жорсткості на *k*-му кроці обчислюємо за формулою

$$S_k = \frac{\max\left\{\sqrt{\mathrm{Re}(\lambda_i)^2 + \mathrm{Im}(\lambda_i)^2}\right\}}{\min\left\{\sqrt{\mathrm{Re}(\lambda_i)^2 + \mathrm{Im}(\lambda_i)^2}\right\}},$$

де λ_i – комплексні власні значення матриці Якобі правої частини системи диференціальних рівнянь ММ.

Спосіб розширення проекту. У проекті реалізовано відкриту архітектуру, яка дає змогу додавати нові моделі та нові методи інтегрування.

Щоб додати нову ММ, необхідно виконати такі кроки:

1) написати клас, успадкований від класу-предка *Model*;
2) визначити у ньому метод *Clone*, що створює новий об’єкт цього класу;
3) написати конструктор, який ініціалізує такі поля:
   - *Name* – назву ММ;
   - *ModelSize* – розмір ММ (кількість диференціальних рівнянь);
   - *GraphicsNames* – назви графіків (їхня кількість дорівнює значенню поля *ModelSize*);
   - *F* – функція, у якій міститься математичний опис моделі;
   - *Parameters* – додаткові параметри, які використовують у диференціальних рівняннях ММ. Ці параметри автоматично відображаються в інтерфейсі користувача, де їх можна змінювати в процесі виконання програми;
4) у конструкторі класу *Models* додати об’єкт написаного класу в колекцію *items*.

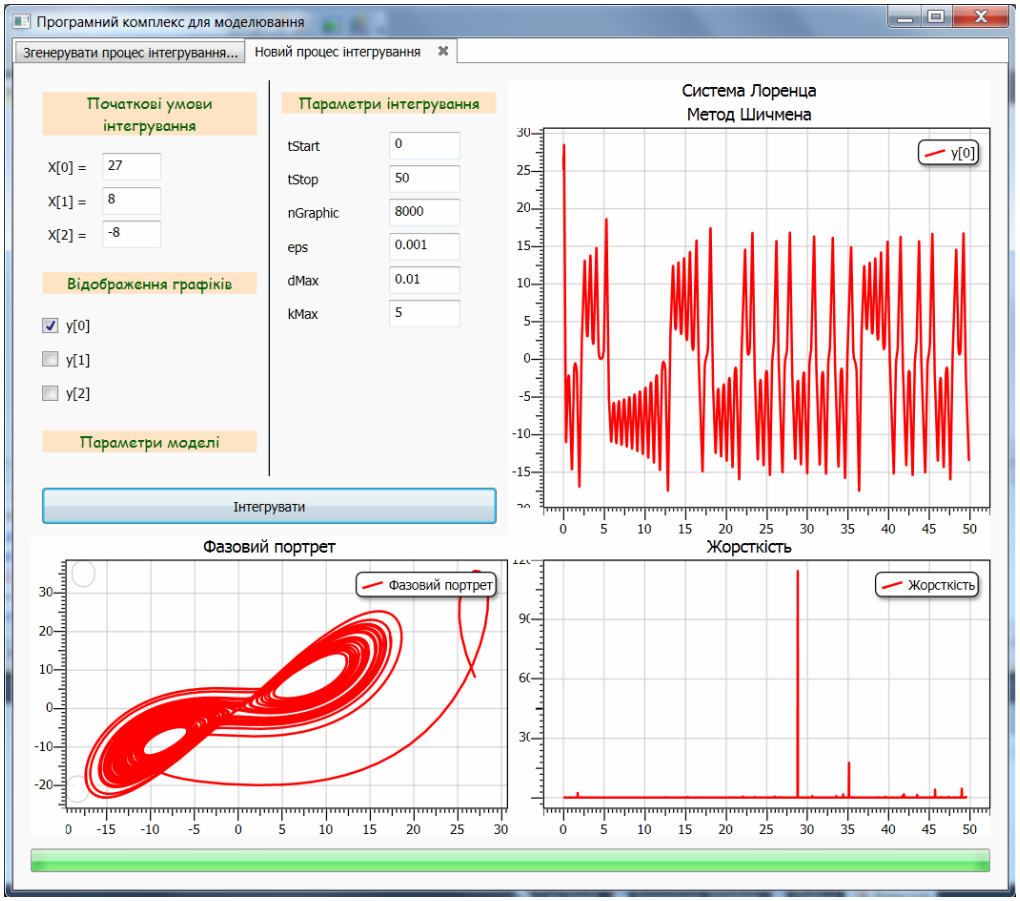

Рис. 7. Робоча вкладка після інтегрування моделі Лоренца методом Шичмена.

Послідовність кроків додавання нового методу інтегрування:

1) написати клас-нащадок від класу *Integrator*;
2) визначити в ньому метод *Clone*, який створює новий об'єкт цього класу;
3) написати конструктор, у якому ініціалізувати такі поля:
   - *Name* – назва методу інтегрування;
   - *Parameters* – додаткові параметри, які використовують в алгоритмі інтегрування. Ці параметри з'являються в інтерфейсі користувача, де їх можна змінювати під час виконання програми;
4) написати метод *Integrate*, який виконуватиме інтегрування;
5) у конструкторі класу *Integrators* додати об'єкт написаного класу в колекцію *items*.

Нижче наведено приклад реалізації математичної моделі системи Лоренца:

```
public class LorenceModel : Model
{ public LorenceModel()
  { this.Name = "Система Лоренца"; this.ModelSize = 3;
    this.GraphicsNames = new string[] { "y[0]", "y[1]", "y[2]" };
    this.F = Lorence;
  }
  private double[] Lorence(double[] x, double t)
  { double s = 10.0; double r = 28.0; double b = 8.0 / 3.0;
    double[] res = new double[3];
    res[0] = -s * (x[0] - x[1]);
    res[1] = r * x[0] - x[1] - x[0] * x[2];
    res[2] = -b * x[2] + x[0] * x[1]; return res;
  }
  public override Model Clone()
  { return new LorenceModel(); }
}
```

Приклад реалізації ЯМЕ у розробленій системі:

```
public class JMEIntegrator : Integrator
{ public JMEIntegrator() : base()
  { this.Name = "Явний метод Ейлера ";
    this.Parameters.Add("hInt", 0.001);
  }
  private void Init()
  { BaseInit();
    this.steps = new Dictionary<double, double>();
  }
  public override Integrator Clone()
  { return new JMEIntegrator(); }
  public override double[][] Integrate()
  { Init();
    double[][] x = new double[2][];
    for (int i = 0; i < 2; i++)
      x[i] = new double[Model.ModelSize];
    int modelSize = Model.ModelSize;
    double hInt = Var("hInt");
    double t = tStart;
    double tGraphic = hGraphic;
    int nInt = 0;
    for (int i = 0; i < modelSize; i++)
```

```
        { Graphics[i][0] = x0[i];
          x[0][i] = x0[i];
        }
        int kg = 1;
        while (true)
        { for (int i = 0; i < modelSize; i++)
            x[1][i] = x[0][i] + hInt * Model.F(x[0], t)[i];
          x[0] = x[1];
          this.steps.Add(t, hInt);
          t += hInt;
          nInt++;
          if (t >= tStop)  // Перевірка умови закінчення моделювання
            break;
          if ((t >= tGraphic) && (kg < nGraphic))
          { for (int i = 0; i < modelSize; i++)
              Graphics[i][kg] = x[0][i];
            tGraphic += hGraphic;
            kg++;
          }
        }
        return Graphics;
      }
    }
```

1. *Хайнеман Р.* PSPISE. Моделирование работы электронных схем. – М.: “ДМК”, 2005. – 318 с.
2. *Хвищун І. О.* Програмування і математичне моделювання : підручник. – К.: Ін Юре, 2007. – 544 с.
3. *Ракитский Ю. В., Устинов С. М.* и др. Численные методы решения жестких систем. – М.: Наука, ГРФМЛ, 1979. – 208 с.
4. *Холл Дж., Уатт Дж.* Современные численные методы решения обыкновенных дифференциальных уравнений. – М.: Мир, 1979. – 312 с.
5. *Чуа Л.О., Пен-Мин Лин.* Машинный анализ электронных схем: Алгоритмы и вычислительные методы. – М.: Энергия, 1980. – 640 с.
6. *Noye J.* Computational Techniques for Differential Equations. – North-Holland: Elsevier Science Publishers B.V., 1984. – 679 p.
7. *Самарский А. А., Гулин А. В.* Численные методы. – М. : Наука. Гл. ред. физ-мат. л-ры, 1989. – 432 с.
8. *Dalhquist G.* A Special Stability Problem for Linear Multistep Methods // B.I.T. – 1963. – N 3. – P. 27–43.
9. *Widlund O. B.* A Note on Unconditionally Stable Linear Multistep Methods // B.I.T. – 1967. – N 7. – P. 65–70.
10. *Gear C. W.* The Automatic Integration of Stiff Ordinary Differential Equations // Information Processing. – 1969. – N 68. – P. 187–193.
11. *Новиков Е. А.* Явные методы для жестких систем / Под ред. А. Н. Горбань. – Новосибирск: Наука. Сиб. Предприятие РАН, 1997. – 195 с.
12. *Хайрер Э., Ваннер Г.* Решения обыкновенных дифференциальных уравнений. Жесткие и дифференциально-алгебраические задачи. – М.: Мир, 1999. – 685 с.

13. *Nathan A., Lehenbauer D*. Windows Presentation Foundation Unleashed. – USA: Sams, 2007. – 655 p.
14. ALGLIB – a cross-platform numerical analysis and data processing library. – http://www.alglib.net/.
15. Dynamic Data Display – a set of Silverlight controls for adding interactive visualization of dynamic data to Silverlight application. – http://dynamicdatadisplay.codeplex.com/.

DEVELOPMENT OF SOFTWARE FOR MODELLING OF DYNAMIC SYSTEMS WITH STIFF MATHEMATICAL MODELS

J. Kost, I. Khvyshchun

Ivan Franko National University of Lviv
107 Tarnavsky St., UA-79017 Lviv, Ukraine
kostjerry@gmail.com

The main concepts are formulated for mathematical modelling of dynamic systems. The stages of the modelling process are distinguished. The definition is given of the errors which may occur in the modelling process. The notions of solution stability of differential equation and of stability of the numerical methods are specified. The software developed for modelling of dynamic systems with stiff mathematical models is described.

Key words: mathematical modelling, stiffness, stability, local error, global error, numerical method, software, Cauchy problem.

РАЗРАБОТКА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ ДЛЯ МОДЕЛИРОВАНИЯ ДИНАМИЧЕСКИХ СИСТЕМ, ИМЕЮЩИХ ЖЕСТКИЕ МАТЕМАТИЧЕСКИЕ МОДЕЛИ

Я. Кость, И. Хвыщун

Львовский национальный университет имени Ивана Франко
ул. Тарнавского, 107, 79017 Львов, Украина
kostjerry@gmail.com

Сформулировано основные понятия математического моделирования динамических систем. Определено этапы процесса моделирования. Дано определение погрешностей, возникающих в процессе математического моделирования. Конкретизировано понятие устойчивости решения дифференциального уравнения и устойчивости численного метода. Описано разработанное программное обеспечение для моделирования динамических режимов систем, имеющих жесткие математические модели.

Ключевые слова: математическое моделирование, жесткость, устойчивость, локальная ошибка, глобальная ошибка, численный метод, программное обеспечение, задача Коши.

Стаття надійшла до редколегії 04.04.2012
Прийнята до друку 15.05.2012