7. Вирішення фізичних та математичних завдань з Maxima

7.1 Операції з поліномами та раціональними функціями

Розглянемо рішення з допомогою Maxima кількох завдань із класичного збірника за редакцією М.І. Сканаві. У Maxima "покрокове" спрощення виразів із послідовним використанням стандартного набору примітивів (формул суми або різниці кубів, формул зведення суми або різниці в ступінь тощо) виконати складно, тому результат є фактично довідковим, на який слід орієнтуватися при вирішенні вручну, за допомогою ручки та паперу.

7.1.1 Спрощення алгебраїчних виразів

Приклад:

Спростити вираз і обчислити його, якщо дані числові значення параметрів:

```
(%i1) g:(1/a-1/(b+c))/(1/a+1/(b+c))*(1+(b^2+c^2-)
                                    a^2)/2/b/c)/((abc)/a/b/c);
```

$$(\%o1)\quad \frac{a\,b\,c\,\left(\frac{1}{a}-\frac{1}{c+b}\right)\,\left(\frac{c^2+b^2-a^2}{2\,b\,c}+1\right)}{(-c-b+a)\,\left(\frac{1}{c+b}+\frac{1}{a}\right)}$$

```
(%i2) ratsimp(%);
```

$$(\%o2)\quad -\frac{a\,c+a\,b-a^2}{2}$$

```
(%i3) %, a = 0.02, b = -11.05, c = 1.07;
```

$$(\%o3)\quad 0.1$$

Приклад: Спростити вираз і обчислити його, якщо дані числові значення параметрів:

```
(%i1) (sqrt(x)+1)/(x*sqrt(x)+x+sqrt(x))/(1/(x^2-
sqrt(x)));
```

$$(\%o1)\quad \frac{\left(\sqrt{x}+1\right)\,\left(x^2-\sqrt{x}\right)}{x^{\frac{3}{2}}+x+\sqrt{x}}$$

```
(%i2) ratsimp(%);
```

$$(\%o2)\quad x-1$$

Приклад: Зробити вказану підстановку та результат спростити:

```
(%i3) expr:(x^3-a^(-2/3)*b^(-
1)*(a^2+b^2)*x+b^(1/2))/(b^(3/ 2)*x^2);
```

$$(\%o3)\quad \frac{x^3-\frac{\left(b^2+a^2\right)\,x}{a^{\frac{2}{3}}\,b}+\sqrt{b}}{b^{\frac{3}{2}}\,x^2}$$

```
(% i4)     ratsimp(%);
```

$$(\%o4)\quad \frac{a^{\frac{2}{3}}\,b\,x^3+\left(-b^2-a^2\right)\,x+a^{\frac{2}{3}}\,b^{\frac{3}{2}}}{a^{\frac{2}{3}}\,b^{\frac{5}{2}}\,x^2}$$

```
(% i5)      radcan(%);
```

$$(\%o5)\quad \frac{a^{\frac{2}{3}}\,b\,x^3+\left(-b^2-a^2\right)\,x+a^{\frac{2}{3}}\,b^{\frac{3}{2}}}{a^{\frac{2}{3}}\,b^{\frac{5}{2}}\,x^2}$$

Без зазначеної підстановки спрощення за допомогою комбінації функцій $ratsimp$ і $radcan$ не вдається.

```
(%i6) %,x=a^(2/3)*b^(-1/2);
```

$$(\%o6)\quad \frac{\frac{a^{\frac{2}{3}}\left(-b^2-a^2\right)}{\sqrt{b}}+a^{\frac{2}{3}}\,b^{\frac{3}{2}}+\frac{a^{\frac{8}{3}}}{\sqrt{b}}}{a^2\,b^{\frac{3}{2}}}$$

```
(%i7) ratsimp(%);
```

Кінцевий результат виявляється простим

$$(\%o7)\quad 0$$

7.1.2 Розкладання поліномів та раціональних виразів на множники

7.1.2.1 Розв'язання рівнянь алгебри

Maxima(як і будь-який інший пакет символьної математики) не завжди здатний отримати остаточне рішення. Однак отриманий результат може виявитися все ж таки простіше, ніж вихідне завдання.

Приклад (також зі збірки під ред. М.І. Сканаві): Розв'язати рівняння $\sqrt{(x-2)}=x-4$:

```
(%i1) solve([sqrt(x-2)=x-4],[x]);
```

$$(\%o1)\quad [x=\sqrt{x-2}+4]$$

Рівняння має одне рішення: $X=6$ Однак для пошуку його за допомогою Maxima доведеться вдатися до заміни вихідного рівняння його наслідком:

```
(%i3) solve([(x-2)=(x-4)^2],[x]);
```

$$(\%o3)\quad [x=6,x=3]$$

Рішення для подальшого використання можна отримати зі списку функцією ev:

```
(%i1) sol:solve([x-2=(x-4)^2],[x]);
```

$$(\%o1)\quad [x=6,x=3]$$

```
(%i2) ev(x, sol[1]);
```

$$(\%o2)\quad 6$$

```
(%i3) ev(x, sol[2]);
```

$$(\%o3)\quad 3$$

Ще два приклади розв'язання рівнянь алгебри:

```
(%i1) eq:7*(x+1/x)-2*(x^2+1/x^2)=9;
```

$$(\%o1)\quad 7\left(x+\frac{1}{x}\right)-2\left(x^2+\frac{1}{x^2}\right)=9$$

```
(%i2) sol:solve([eq],[x]);
```

$$(\%o2)\quad [x=2,x=\frac{1}{2},x=-\frac{\sqrt{3}\,i-1}{2},x=\frac{\sqrt{3}\,i+1}{2}]$$

```
(%i3) x1:ev(x, sol [1]); x2:ev(x,sol[2]);
/*комплексне коріння не розглядаємо*/
```

$$(\%o4)\quad 2\frac{1}{2}$$

Рівняння з радикалами перед рішенням у Maxima доводиться перетворювати до статечної форми (для виділення лівої та правої частини виразу використовують функції lhs і rhs відповідно):

```
(%i1) eq:sqrt(x+1)+sqrt(4*x+13)=sqrt(3*x+12);
```

$$(\%o1)\quad \sqrt{4\,x+13}+\sqrt{x+1}=\sqrt{3\,x+12}$$

```
(%i2) eq1:lhs(eq) ^ 2 = rhs (eq) ^ 2;
```

$$(\%o2)\quad \left(\sqrt{4\,x+13}+\sqrt{x+1}\right)^2=3\,x+12$$

```
(%i3) solve([eq1],[x]);
```

$$(\%o3)\quad [x=-\sqrt{x+1}\,\sqrt{4\,x+13}-1]$$

```
(% i4)      eq2:x+1=rhs(%[1])+1;
```

$$(\%o4)\quad x+1=-\sqrt{x+1}\,\sqrt{4\,x+13}$$

```
(% i5)      eq3:lhs(eq2)^2=rhs(eq2)^2;
```

$$(\%o5)\quad (x+1)^2=(x+1)\,(4\,x+13)$$

Остання команда дозволила отримати статечне рівняння, що дозволяється аналітично в Maxima (для цього знадобилося двічі звести в квадрат вихідне рівняння).

```
(%i6) solve([eq3],[x]);
```

$$(\%o6)\quad [x=-4,x=-1]$$

Перевірку рішення виконуємо за допомогою функції ev .

Рішення $x=-4$ не задовольняє вихідне рівняння.

```
(%i7) ev(eq, % [1]);
```

$$(\%o7)\quad 2\sqrt{3}\,i=0$$

Рішення $x=-1$ перетворює вихідне рівняння на вірну рівність:

```
(% i8)      ev(eq, % o6 [2]);
```

$$(\%o8)\quad 3=3$$

Розглянемо ще один приклад, що ілюструє заміну та підстановку при вирішенні рівнянь алгебри:

```
(%i1) eq:sqrt(x+3-4*sqrt(x-1))+sqrt(x+8-6*sqrt(x-1))=1;
```

Вихідне рівняння:

$$(\%o1)\quad \sqrt{x-4\sqrt{x-1}+3}+\sqrt{x-6\sqrt{x-1}+8}=1$$

Виконаємо заміну $\sqrt{x+1}=z,\ z=x^2+1$

```
(%i2) eq1:subst(z, sqrt (x-1), eq);
```

$$(\%o2)\quad \sqrt{-4z+x+3}+\sqrt{-6z+x+8}=1$$

```
(%i3) eq2:subst(z^2+1,x,eq1);
```

$$(\%o3)\quad \sqrt{z^2-4z+4}+\sqrt{z^2-6z+9}=1$$

Спрощуємо отриманий результат:

```
(% i4)      radcan(%);
```

$$(\%o4)\quad 2z-5=1$$

```
(% i5)      solve([%],z);
```

$$(\%o5)\quad [z=3]$$

```
(%i6) solve([sqrt(x-1)=3],[x]);
```

$$(\%o6)\quad [x=10]$$

Виконаємо перевірку

```
(%i7) ev(eq,%[1]);
```

$$(\%o7)\quad 1=1$$

Значна частина тригонометричних рівнянь шкільного курсу також можна розв'язати в Maxima, але безпосереднє рішення вдається отримати далеко не завжди.

Приклади:

```
(%i1) solve([sin(%pi/6-x)=sqrt(3)/2],[x]);
solve: використовуючи arc-trig функцій, щоб отримати
рішення.
Кілька рішень буде втрачено.
```

$$(\%o1)\quad [x=-\frac{\pi}{6}]$$

Більшість тригонометричних рівнянь у Maxima (крім найпростіших) доводиться вирішувати приведенням їх до алгебраїчних.

Логарифмічні та показові рівняння також вирішуються в Maxima шляхом заміни змінних та зведення до алгебраїчних (див. вище специфічні функції для спрощення логарифмічних виразів).

7.2 Деякі фізичні завдання

Застосування систем символьної математики у викладанні фізики та хімії дозволяє зосередитися на змістовній частині матеріалу, що викладається. Крім того, учні отримують можливість вирішувати куди складніші завдання, ніж при ручних розрахунках. Наявність у Maxima чітко вираженого алгоритмічного мови (на відміну Matcad) істотно знижує ризик підміни понять, коли прогалини власного підходи до вирішення завдання учні відносять наявність помилок і неточностей в програмному забезпеченні.

Ідеї розглянутих завдань взяті з відомих посібників з використання MathCad, однак, на думку автора, використання Maxima може бути не меншим, а в багатьох випадках і більш ефективним.

7.2.1 Обчислення середньої квадратичної швидкості молекул

Вираз, що містять змінні, по суті може використовуватися як функція користувача Maxima.

Розглянемо можливість обчислення середньоквадратичної швидкості молекул для різних газів. Використовувана формула: $v = \sqrt{\frac{3RT}{M}}$, де $R = 8.314 Дж/(моль \cdot К), T$ - Абсолютна температура, M - Молярна маса.

Обчислимо середньоквадратичну швидкість молекул CO_2 (M = 0.044 кг/моль) при температурі 273 К:

```
(%i1) v:sqrt(3*R*T/M);
```

$$(\%o1) \quad \sqrt{3}\sqrt{\frac{RT}{M}}$$

```
(%i2) vCO2:float(v),M=0.044,T=273,R=8.314;
```

$$(\%o2) \quad 393.3875604633079$$

Розрахунок для кількох різних газів нескладно провести, варіюючи молярну масу:

```
(%i3) vVozd:float(v),M=0.029,T=273,R=8.314;
```

$$(\%o3) \quad 484.5604478145187$$

```
(% i4)      vH2:float(v),M=0.002,T=273,R=8.314;
```

$$(\%o4) \quad 1845.151213315592$$

7.2.2 Розподіл Максвелла

Аналогічно попередньому розрахунку створимо вираз, що описує розподіл Максвелла (див. блок команд Maxima нижче).

```
(%i1)     fun:4*%pi*(M/2/%pi/R/T)^(3/2)*exp(-
M*v^2/2/R/T)*v^2;
```

$$(\%o1) \quad \frac{2\,v^2\left(\frac{M}{RT}\right)^{\frac{3}{2}} e^{-\frac{v^2 M}{2RT}}}{\sqrt{2}\sqrt{\pi}}$$

Для аналізу отриманих виразів у формулу розподілу Максвелла підставляємо лише температуру:

```
(%i2)     fun70:fun,T=70;
```

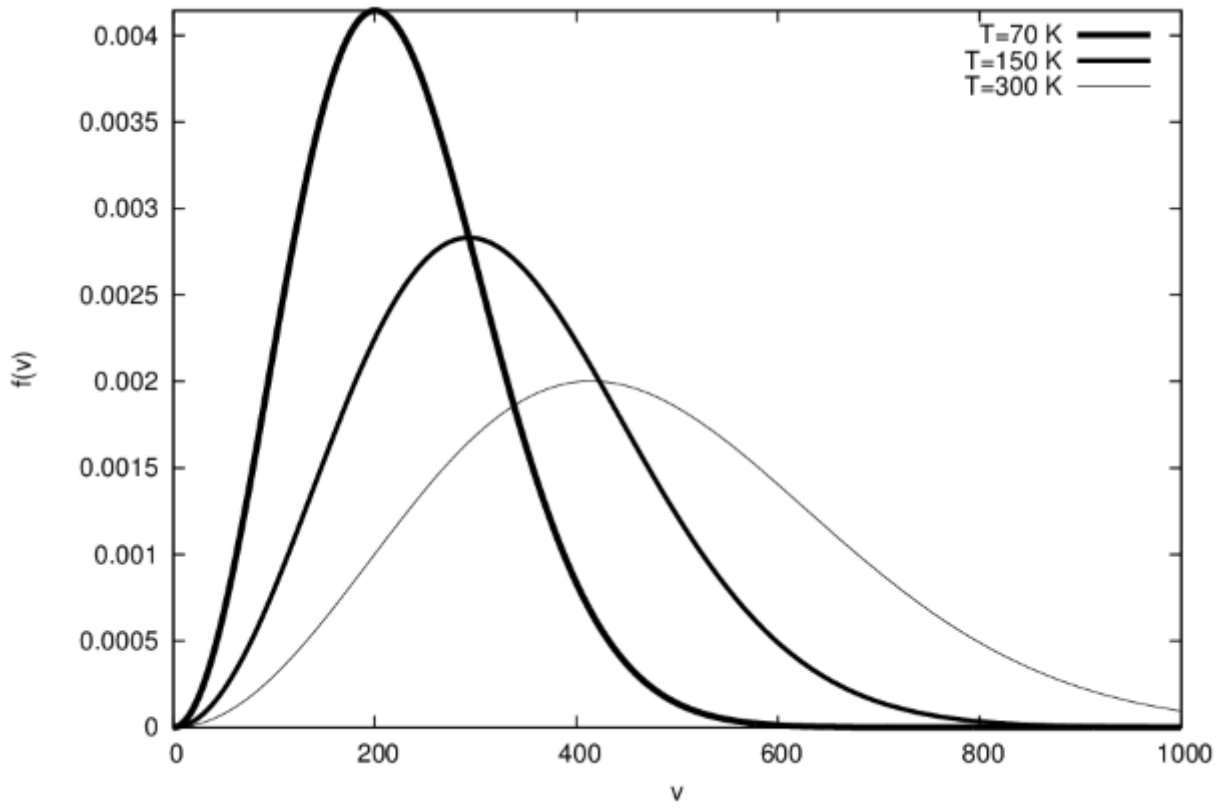
$$(\%o2) \quad \frac{v^2\left(\frac{M}{R}\right)^{\frac{3}{2}} e^{-\frac{v^2 M}{140R}}}{35\sqrt{2}\sqrt{70}\sqrt{\pi}}$$

```
(%i3)     fun150:fun,T=150;
```

$$(\%o3)\quad \frac{v^2\left(\frac{M}{R}\right)^{\frac{3}{2}} e^{-\frac{v^2 M}{300\,R}}}{375\sqrt{2}\sqrt{6}\sqrt{\pi}}$$

```
(% i4)   fun300:fun,T=300;
```

$$(\%o4)\quad \frac{v^2\left(\frac{M}{R}\right)^{\frac{3}{2}} e^{-\frac{v^2 M}{600\,R}}}{1500\sqrt{2}\sqrt{3}\sqrt{\pi}}$$



Мал. 7.1.Розподіл Максвелла за швидкістю молекул повітря для різних температур

Для побудови графіка залежності функції розподілу від температури підставляємо молярну масу повітря та величину універсальної газової постійної (див. результати на рис. 7.1):

```
(% i5)   plot2d([fun70,fun150,fun300],[v,0,1000]),
M=0.029,R=8.314;
```

Можна вивчити вплив температури на форму кривої, а також на положення максимуму функції розподілу. За допомогою інтегрування $f(v)$ можна вважати частку молекул, що володіють швидкостями в якомусь інтервалі, а також визначити середню і середню квадратичну швидкості молекул.

Приклад:

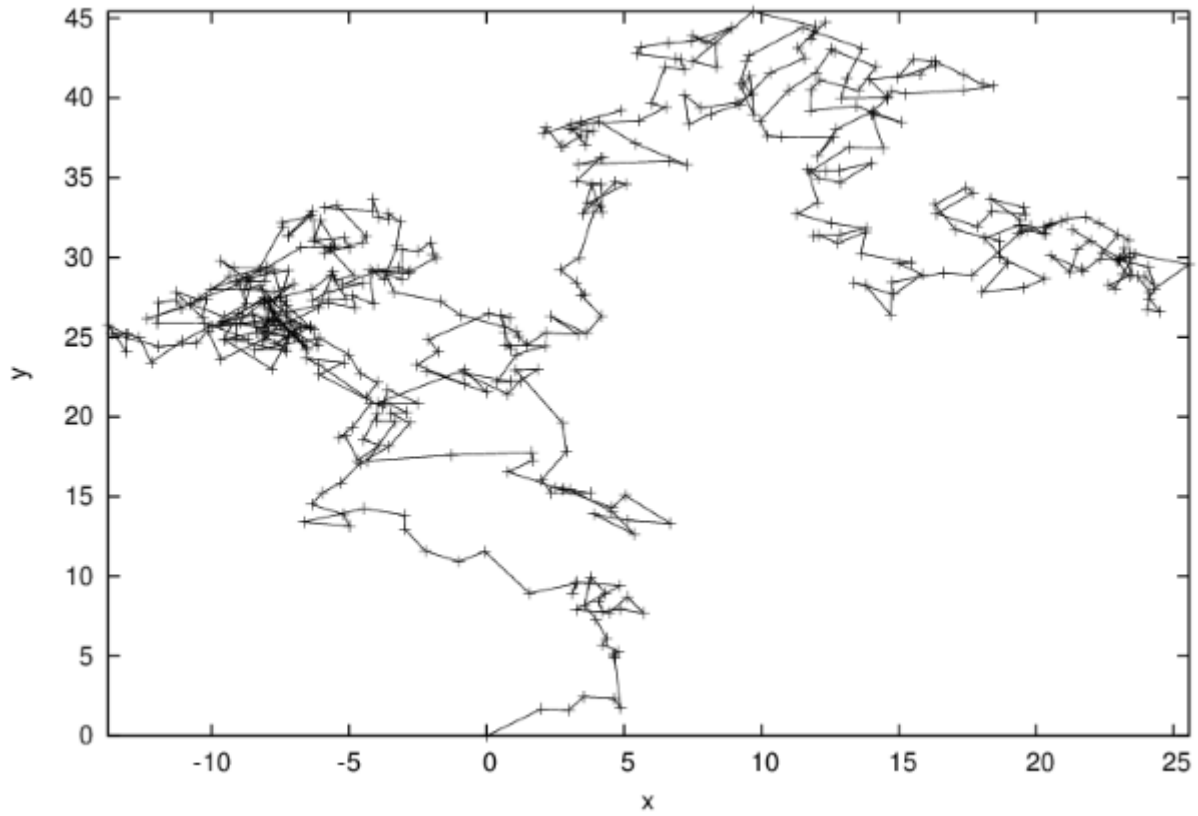
```
(%i6)    integrate(v*v*fun,v,0,inf);
Чи є M\,R\,T positive, negative, or zero?
p;
```

$$(\%o6)\quad \frac{3\,R\,T}{M}$$

Таким чином, $\langle v^2 \rangle = \frac{3RT}{M}$, звідки середньоквадратична швидкість молекул газу $\langle \sqrt{v^2} \rangle = v_{sq} = \sqrt{\frac{3RT}{M}}$.

7.2.3 Броунівський рух

Наявність генератора випадкових чисел дозволяє моделювати рух броунівської частки.



Мал. 7.2.Траєкторія броунівського руху модельної частки

Ейнштейн перший розрахував параметри броунівського руху, показавши, що нерегулярне переміщення частинок, зважених у рідині, викликано випадковими ударами сусідніх молекул, які здійснюють тепловий рух. Відповідно до теорії Смолуховського-Ейнштейна, середнє значення квадрата зміщення броунівської частки (s^2) за час t прямо пропорційно температурі T і назад пропорційно в'язкості рідини h, розміру частки r та постійної Авогадро N_A: $s^2 = \frac{2RTt}{6\pi hrN_A}$, де R - Постійна газова.

Броунівські частки мають обсяг порядку 0,1–1 мкм, тобто. від однієї тисячної до десятитисячної частки міліметра.

Побудуємо дещо спрощену модель броунівського руху, припускаючи, що усунення частки по кожній із координат — нормально розподілена випадкова величина з нульовим математичним очікуванням. Для генерації випадкових чисел використовуємо пакет distrib, що включає необхідні функції (використовується генератор random_normal

```
(%i1)    load("distrib")$
```

```
x:0$ y:0$ xy:[[0,0]]$ m:0$ s:1$
Nmax:500$ for i:1 до Nmax do (x:x+random_normal(m,s),
y:y+random_normal(m,s), xy:append(xy,[[x,y]]))$
plot2d([discrete, xy]);
```

Результат побудови графіка наведено на рис. 7.2.

7.3 Приклад побудови статистичної моделі

Розглянемо побудову завдання із практичним змістом.

У табл. 7.1 наведено дані (взяті зі статті В. Ф. Очкова: http://twt.mpei.ac.ru/ochkov/) про залежність ціни уживаного автомобіля від його пробігу та "віку" (часу використання). У статті-першоджерелі завдання дослідження цієї залежності вирішувалося засобами MathCad.

Розглянемо її рішення засобами Maxima.

Таблиця 7.1. Вартість уживаного автомобіля в залежності від його віку та пробігу

Вік (років)	**Пробіг (миль)**	**Ціна ($)**
11.5	88000	1195
10.5	82000	1295
12.5	97000	800
8.5	51000	2295
9.5	79000	1995
13.5	120000	495
3.5	39000	4995
6.5	52000	2695
4.5	39000	3995
12.5	92000	795
7.5	41000	3495
10.5	77000	1595
12.5	83000	895
4.5	38000	3990
13.5	92000	795
13.5	103000	750
10.5	65000	1495
10.5	70000	1495
10.5	80000	1495
6.5	57000	2695
11.5	101000	895
10.5	78000	1295
9.5	84000	1995
4.5	46000	3675

Вік (років)	Пробіг (миль)	Ціна ($)
11.5	108000	975
13.5	124000	850
6.5	56000	3495
9.5	67000	2495
6.5	43000	3400
11.5	78000	1295

Надалі передбачається, що вихідні дані рішення підготовлені як файла cars.txt. Для зчитування використовуємо пакет numericalio. У пам'яті дані надаються матрицею, а для побудови окремих графіків — списками (змінні $age, mile, price$ - Див. нижче).

```
(%i1)    load("draw")$
(%i2)    load("numericalio")$
(%i3)    data:read_matrix("cars1.txt")$
(% i4)   age:makelist(data[k,1], k, 1, 30)$
(% i5)   mile:makelist(data[k,2], k, 1, 30)$
(%i6)    price:makelist(data[k,3], k, 1, 30)$
```

Найпростішу лінійну регресію можна побудувати, використовуючи функцію $simple_{l}inear_{r}egression$ (Пакет stats). Побудуємо залежність ціни автомобіля від його вартості та пробігу:

```
(%i21)      xy:makelist([age[k],price[k]], k, 1, 30)$
(%i22)      simple_linear_regression(xy);
```

$$(\%o22)\quad \left(\begin{array}{c} SIMPLE\ LINEAR\ REGRESSION \\ model = 5757.594446543255 - 392.7181715149224\,x \\ correlation = -.9688177942467208 \\ v\_estimation = 95364.34912839333 \\ b\_conf\_int = [-431.5987157329751, -353.8376272968697] \\ hypotheses = H0 : b = 0, H1 : b\#0 \\ statistic = 20.69021212080514 \\ distribution = [student\_t, 28] \\ p\_value = 0.0 \end{array}\right)$$

Побудуємо аналогічну залежність ціни автомобіля від пробігу, але не в лінійній, а в експоненційній формі:

```
(%i26)      xy:makelist([mile[k],log(price[k])], k, 1,
30)$
(%i27)      simple_linear_regression(xy);
```

$$(\%o27)\quad \left(\begin{array}{c} SIMPLE\ LINEAR\ REGRESSION \\ model = 9.174960600286802 - 2.3747715120748164\,10^{-5}\,x \\ correlation = -.9301125564244438 \\ v\_estimation = .05467789749118319 \\ b\_conf\_int = [-2.7377780810631264\,10^{-5}, -2.0117649430865062\,10^{-5}] \\ hypotheses = H0 : b = 0, H1 : b\#0 \\ statistic = 13.40058098403749 \\ distribution = [student\_t, 28] \\ p\_value = 5.928590951498336\,10^{-14} \end{array}\right)$$

Отримані залежності представлені у вигляді виразів Maxima:

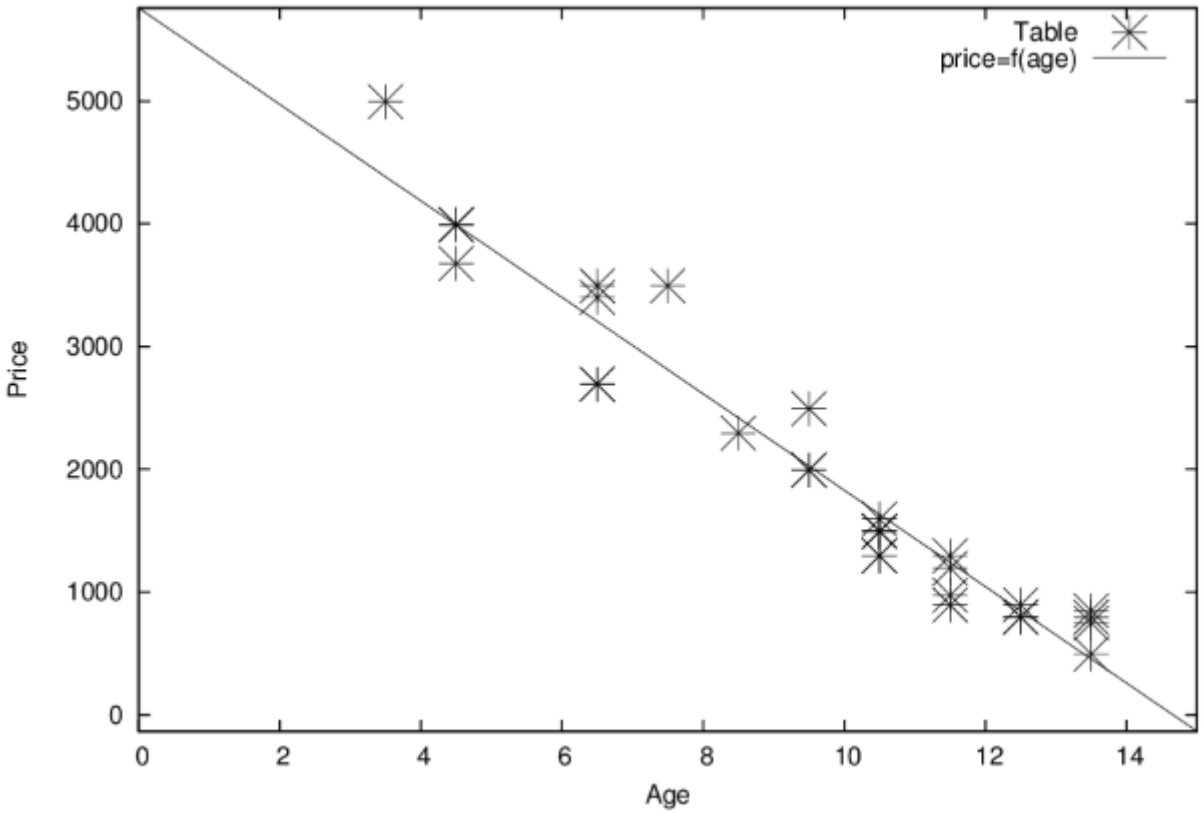
```
(%i28)     fun1:5757.6-392.7*x$(%i29)exp(9.175);
```

$$(\%o29)\qquad 9652.768071616591$$

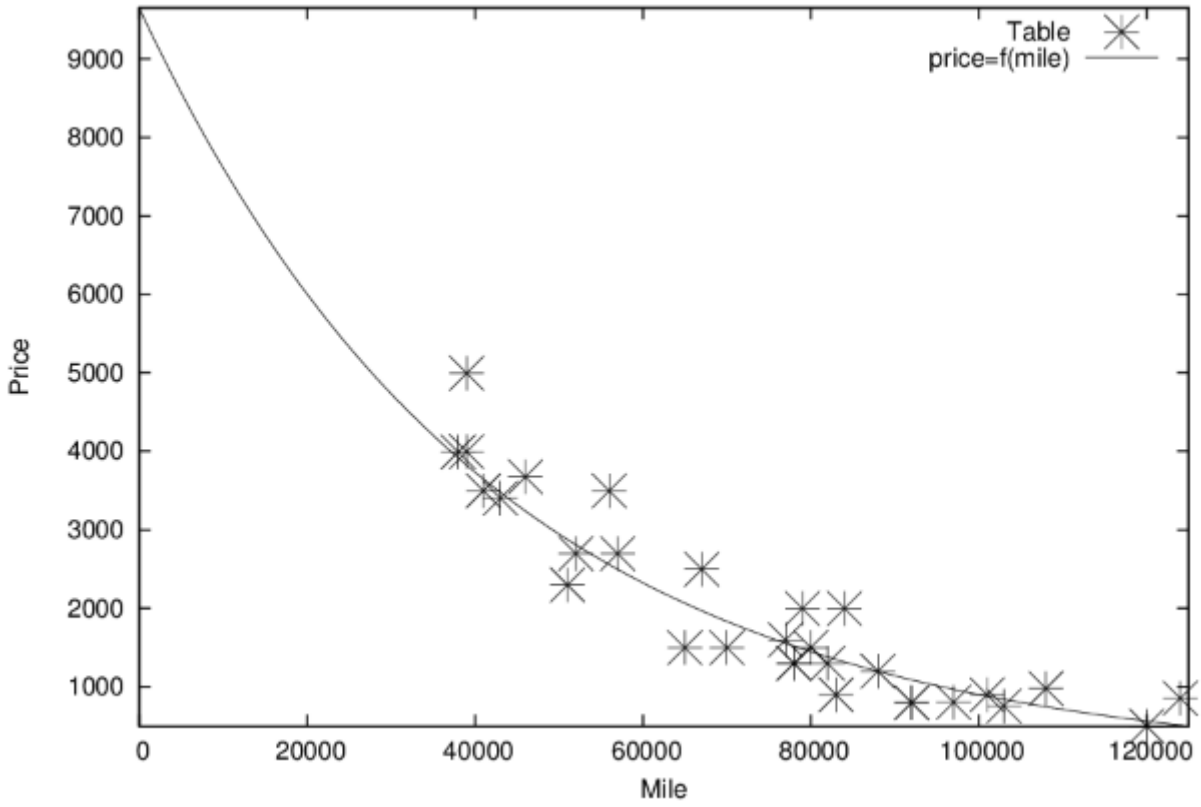
```
(% i30)    fun2:9653*exp(-2.375*10^(-5)*x);
```

$$(\%o30)\qquad 9653\,e^{-2.3750000000000001\,10^{-5}\,x}$$

Проілюструємо отримані результати графічно (рис. 7.3 та рис. 7.4):



Мал. 7.3.Залежність ціни уживаного автомобіля від віку



Мал. 7.4.Залежність ціни автомобіля від його пробігу

```
(%i34)     draw2d(terminal=eps,key="Table",
xlabel="Age",ylabel="Price",
     point_size = 3,point_type=3,points(age,price),
     key="price=f(age)",explicit(fun1,x,0,15));
(%i41)     draw2d(terminal=eps,key="Table",
xlabel="Mile",ylabel="Price",
     point_size = 3,point_type=3,points(mile,price),
     key="price=f(mile)",explicit(fun2,x,0,125000));
```

Для побудови моделі у вигляді залежності ціни автомобіля від пробігу та віку одночасно доцільно використовувати складнішу функцію $lsquares_estimates$ (пакет lsquares). Шукана модель була представлена рівнянням:

$$Price = a + b * Age + c * Mile + d * Mile^2$$

Необхідні команди Maxima:

```
(% i5)     lsquares_estimates(data,[x,y,z],
z=a+b*x+c*y+d*y^2,[a,b,c,d]);
```

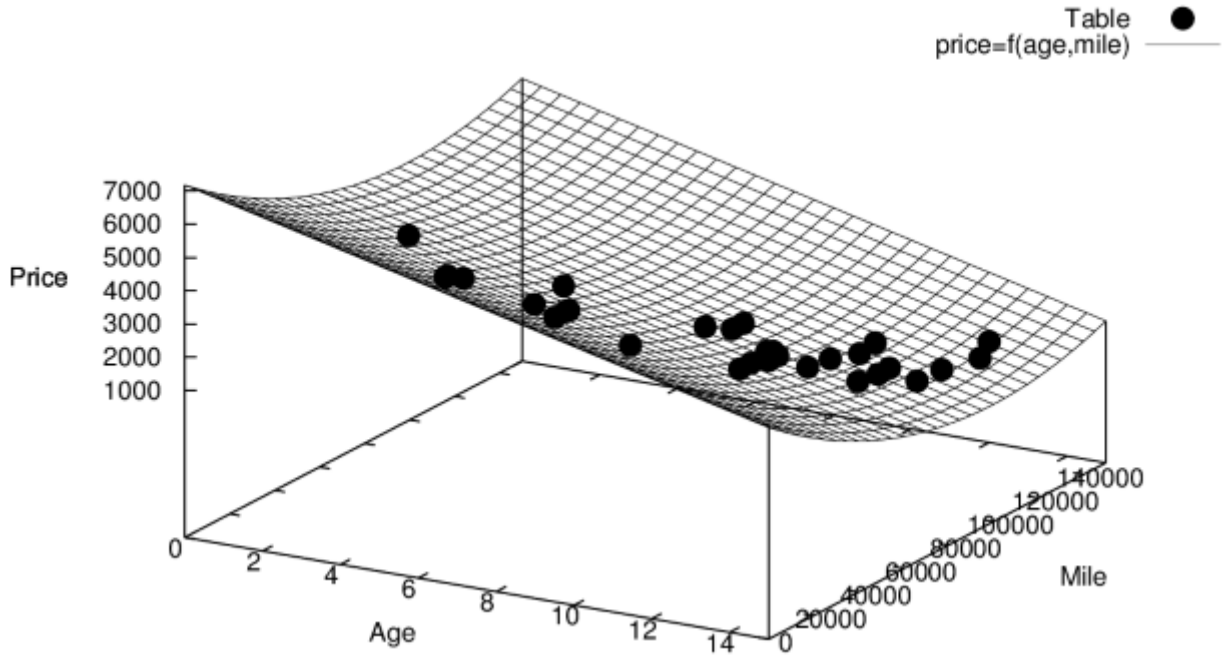
$$(\%o5)\ [[a = \frac{36712000090549571}{5117101479342}, b = -\frac{80056614985946}{284283415519},$$

$$c = -\frac{194393701258481}{3411400986228000}, d = \frac{2937180994967}{1023420295868400000}]]$$

```
(%i6) float(%);
```

$$(\%o6)\ [[a = 7174.37405506961, b = -281.6084604857839,$$
$$c = -0.056983539033745, d = 2.8699655525931528\,10^{-7}]]$$

Слід зазначити, що дуже нелінійні завдання вирішуються за допомогою $lsquares_estimate$ повільно, тому результати побудови моделі залежать від обгрунтованості постановки завдання оцінювання. Графічна ілюстрація представлена на рис. 7.5.



Мал. 7.5.Ілюстрація залежності відгуку (ціни уживаного автомобіля) від двох незалежних факторів (віку та пробігу автомобіля)

8. Реалізація деяких чисельних методів

8.1 Програмування методів розв'язання нелінійних рівнянь у Maxima

Необхідність відшукання коренів нелінійних рівнянь зустрічається в ряді завдань: розрахунках систем автоматичного управління та регулювання, власних коливань машин і конструкцій, у завданнях кінематичного аналізу та синтезу, плоских та просторових механізмів та інших задачах.

Нехай дано нелінійне рівняння $f(x) = 0$, і необхідно вирішити це рівняння, тобто знайти його корінь $\overline{x}$.

Якщо функція має вигляд багаточлена ступеня m, $f(x) = a_0x^m + a_1x^{m-1} + a_2x^{m-2} + \ldots + a_{m-1}x + a_m$,

де a_i - Коефіцієнти многочлена, $i = \overline{1, m}$, то рівняння $f(x) = 0$ має m коріння (основна теорема алгебри).

Якщо функція $f(x)$ включає тригонометричні або експоненційні функції від деякого аргументу x, то рівняння $f(x) = 0$ називається трансцендентним рівнянням. Такі рівняння зазвичай мають безліч рішень.

Як відомо, не всяке рівняння можна вирішити точно. Насамперед це стосується більшості трансцендентних рівнянь.

Доведено також, що не можна побудувати формулу, за якою можна було б вирішувати довільні рівняння алгебри ступеня, вище четвертого.

Однак, точне рішення рівняння не завжди є необхідним. Завдання відшукання коренів рівняння вважатимуться майже вирішеною, якщо ми зможемо визначити коріння рівняння із заданим ступенем точності. І тому використовуються наближені (чисельні) методи рішення.

Більшість наближених методів розв'язання рівнянь, що вживаються, є, по суті, способами уточнення коренів. Для їх застосування необхідне знання інтервалу ізоляції $[a, b]$, в якому лежить корінь рівняння, що уточняється.

Процес визначення інтервалу ізоляції $[a, b]$, Що містить лише один з коренів рівняння, називається відділенням цього кореня.

Процес відділення коренів проводять виходячи з фізичного сенсу прикладного завдання, графічно за допомогою таблиць значень функції $f(x)$ або за допомогою спеціальної програми відділення коріння. Процедура відокремлення коренів заснована на відомій властивості безперервних функцій: якщо функція безперервна на замкнутому інтервалі $[a, b]$ і його кінцях має різні знаки, тобто. $f(a) \cdot f(b) < 0$, то між точками a і b є хоча б один корінь рівняння $f(x) = 0$. Якщо при цьому функція $f(x)$ на відрізку $[a, b]$ монотонна, то вказаний корінь єдиний.

Процес визначення коренів алгебраїчних та трансцендентних рівнянь складається з двох етапів:

1. відділення коріння, - тобто. визначення інтервалів ізоляції $[a, b]$, всередині якого лежить кожен корінь рівняння;
2. уточнення коренів, тобто. звуження інтервалу $[a, b]$ до величини рівного заданого ступеня точності ε.

Для алгебраїчних і трансцендентних рівнянь придатні одні й самі методи уточнення наближених значень дійсних коренів:

- метод половинного поділу (метод дихотомії);
- метод простих ітерацій;
- метод Ньютона (метод дотичних);
- модифікований метод Ньютона (метод сікучих);
- метод хорд та ін.

8.1.1 Метод половинного поділу

Розглянемо таке завдання: дано нелінійне рівняння $f(x) = 0$, необхідно знайти корінь рівняння, що належить інтервалу $[a, b]$, із заданою точністю ε.

Для уточнення кореня методом половинного поділу послідовно здійснюємо наступні операції:

- обчислюємо значення функції $f(x)$ у точках a і $t = (a + b)/2$;
- якщо $f(a) \cdot f(t) < 0$, то корінь знаходиться у лівій половині інтервалу $[a, b]$ тому відкидаємо праву половину інтервалу і приймаємо $b = t$;
- якщо умова $f(a) \cdot f(t) < 0$ не виконується, то корінь знаходиться у правій половині інтервалу $[a, b]$ тому відкидаємо ліву половину інтервалу $[a, b]$ за рахунок присвоєння $a = t$.

В обох випадках новий інтервал $[a, b]$ у 2 рази менший за попередній.

Процес скорочення довжини інтервалу невизначеності циклічно повторюється доти, доки довжина інтервалу $[a, b]$ стане рівної чи меншою заданої точності, тобто. $|b - a| \leq \varepsilon$.

Реалізація методу половинного поділу у вигляді функції Maxima представлена в наступному прикладі:

власне функція $bisect$, в яку передається вираз f, Визначальне рівняння, яке необхідно вирішити

```
(%i1)
    bisect(f,sp,eps):=block([a,b],a:sp[1],b:sp[2],p:0,
      while abs(ba)>eps do (
         p:p+1, c:(a+b)/2,
         fa:float(subst(a,x,f)), fc:float(subst(c,x,f)),
```

```
          якщо fa*fc<0 then b:c else a:c
        ),
float(c));
```

$$(\%o1)\ \text{bisect}\,(f, sp, eps) := \text{block}([a, b], a : sp_1, b : sp_2, p : 0,\ \text{while}\,|b - a| > eps\ \text{do}(p : p{+}1, c : \tfrac{a+b}{2}, fa : \text{float}\,(\text{subst}\,(a, x, f)),\ fc : \text{float}\,(\text{subst}\,(c, x, f)),\ \text{if}\ fa\,fc < 0\ \text{then}\ b : c\ \text{else}\ a : c),\ \text{float}\,(c))$$

послідовність команд, що організує звернення до bisect та результати обчислень

```
(%i2)    f:exp(-x)-x$
a:-1$ b:2$ eps:0.000001$ xrez:bisect(f,[-2,2],eps)$
print("Рішення",xrez,"Нев'язка",subst(xrez,x,f))$
Рішення 0.56714344024658 Нев'язка - 2.3481572652972410-7
```

$$(\%o2)\ -2.3481572652972\,10^{-7}$$

У наведеному прикладі вирішується рівняння $e^{-x} - x = 0$. Пошук кореня здійснюється на відрізку [-2, 2] з точністю 0.000001.

Слід зазначити особливість програмування для Maxima, яка полягає в тому, що рівняння, що розв'язується, задається у вигляді математичного виразу (тобто фактично текстового рядка). Числове значення нев’язкі рівняння обчислюється за допомогою функції subst, за допомогою якої виконується підстановка значень x = a або $x = c$ у заданий вираз. Обчислення нев’язкі після рішення здійснюється також шляхом встановлення результату рішення $xrez$ на вираз f .

8.1.2 Метод простих ітерацій

У ряді випадків дуже зручним прийомом уточнення кореня рівняння є метод послідовних наближень (метод ітерацій).

Нехай із точністю ε необхідно знайти корінь рівняння $f(x) = 0$, що належить інтервалу ізоляції $[a, b]$. Функція $f(x)$ та її перша похідна безперервні на цьому відрізку.

Для застосування цього методу вихідне рівняння $f(x) = 0$ має бути приведено до вигляду $x = \varphi(x)$.

Як початкове наближення може бути обрана будь-яка точка інтервалу $[a, b]$.

Далі ітераційний процес пошуку кореня будується за схемою:

$$x_1 = f(x_0),$$
$$x_2 = f(x_1),$$
$$\ldots$$
$$x_n = f(x_{n-1})$$

В результаті ітераційний процес пошуку реалізується рекурентною формулою. Процес пошуку припиняється, як тільки виконується умова $|x_n - x_{n-1}| \le \varepsilon$ чи кількість ітерацій перевищить задане число N.

Для того, щоб послідовність $x_1, x_2, ..., x_n$ наближалася до шуканого кореня, необхідно, щоб виконувалася умова збіжності $|\varphi'(x)| < 1$.

Приклад реалізації методу ітерацій наведено нижче:

```
(%i1)    f:exp(-x)-x$
beta:0.1$ x1:1$ x0:0$ eps:0.000001$ p:0$
while abs(x1-x0)>eps do
        (x0:x1, p:p+1,
x1:float(x0+beta*(subst(x0,x,f))))$
print("Кількість ітерацій",p,"","Рішення",float(x1),
"Нев'язка ",float(abs(x1-x0)))$
```

Число итераций 67 Решение 0.56714848327814

Невязка $9.650298036234517 10_{7}$

8.1.3 Метод Ньютона (метод дотичних)

Розглянуті раніше методи розв'язання нелінійних рівнянь є методами прямого пошуку. У них для знаходження кореня використовується знаходження значення функції у різних точках інтервалу $[a, b]$.

Метод Ньютона належить до градієнтних методів, у яких знаходження кореня використовується значення похідної.

Розглянемо нелінійне рівняння $f(x) = 0$, для якого необхідно знайти корінь на інтервалі $[a, b]$ з точністю ε.

Метод Ньютона ґрунтується на заміні вихідної функції $f(x)$, на кожному кроці пошуку дотичної, проведеної до цієї функції. Перетин дотичної з віссю X дає наближення кореня.

Виберемо початкову точку $x_0 = b$ (Кінець інтервалу ізоляції). Знаходимо значення функції у цій точці і проводимо до неї дотичну, перетин якої з віссю X дає перше наближення кореня x_1:

$x_1 = x_0 - h_0,$ где

$h_0 = \frac{f(x_0)}{\mathrm{tg}(\alpha)} = \frac{f(x_0)}{f'(x_0)}.$

Тому $$x_1 = x_0 - \frac{f(x_0)}{f'(x_0)}.$$

В результаті, ітераційний процес сходження до кореня реалізується рекурентною формулою

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)}$$

Процес пошуку продовжуємо доти, доки не виконається умова: $|x_{n+1} - x_n| \le \varepsilon$, звідки $|\frac{f(x_n)}{f'(x_n)}| \le \varepsilon$.

Метод забезпечує швидку збіжність, якщо виконується умова: $f(x_0) \cdot f''(x_0) > 0$, тобто. першу дотичну рекомендується проводити в тій точці інтервалу $[a, b]$ де знаки функції $f(x_0)$ та її кривизни $f''(x_0)$ збігаються.

Приклад реалізації методу Ньютона Maxima представлений нижче:

```
(%i1)    newton(f,x0,eps):=block([df,xn,xn0,r,p],
    xn0:x0, df:diff(f,x),
    p:0, r:1,
    while abs(r)>eps do (
        p:p+1, xn:xn0-
float(subst(xn0,x,f)/subst(xn0,x,df)),
        print("x0,x1",xn0,xn),r:xn-xn0, xn0:xn
    ),
    [xn, p]) $
```

Послідовність команд для звернення до функції $newton$ та результати обчислень представлені в наступному прикладі:

```
(%i2)    f:exp(-x)-x$
    eps:0.000001$ xrez:newton(f,1,eps)$
    print("Рішення", xrez [1], "Кількість ітерацій", xrez
[2],
    "Нев'язка ", subst (xrez [1], x, f)) $
```

$x0, x1 \quad 1.53788284273999$

x0,x1 0.53788284273999 0.56698699140541

$x0, x1 \quad 0.56698699140541 \quad 0.56714328598912$

x0,x1 0.56714328598912 0.56714329040978

Решение 0.56714329040978 Число итераций 4 Невязка 0.0

Особливості наведеного прикладу — проміжний друк результатів і значення, що повертається у вигляді списку, що дозволяє одночасно отримати як значення кореня, так і необхідне для досягнення заданої точності число ітерацій. Істотному зменшенню числа ітерацій сприяє і аналітичне обчислення похідної.

8.1.4 Модифікований метод Ньютона (метод сікучих)

У цьому методі для обчислення похідних на кожному кроці пошуку використовується чисельне диференціювання за такою формулою:

$$f'(x) = \frac{\Delta f(x)}{\Delta x}$$

Тоді рекурентна формула методу Ньютона набуде вигляду:

$$x_{n+1} = x_n - \frac{f(x_n)}{f'(x_n)} = x_n - \frac{f(x_n)\Delta x}{\Delta f(x_n)} =$$
$$= x_n - \frac{f(x_n)\Delta x}{f(x_n+\Delta x)-f(x_n)},$$

де $\Delta x \approx \varepsilon$.

Реалізація цього методу в Maxima представлено нижче:

```
(%i1)    secant(f,sp,eps):=block([x0,x1,d,y,r],
    x0:sp[1],x1:sp[2],
    p:0, r:x1-x0, d:float(subst(x0,x,f)),
    while abs(r)>eps do (
        p:p+1, y:float(subst(x1,x,f)), r:r/(dy)*y,
        d:y, x1:x1+r
    ),
    x1)$
(%i2)    f:exp(-x)-x$
    eps:0.000001$ xrez:secant(f,[-2,2],eps)$
    print("Рішення",xrez,"Нев'язка",subst(xrez,x,f))$
```

Решение 0.56714329040978 Невязка $-1.1102230246251565\,10^{-16}$

Особливості програмування для Maxima, використані в цьому прикладі, аналогічні наведеним вище в прикладі щодо методу половинного поділу.

8.1.5 Метод хорд

Метод заснований на заміні функції $f(x)$ на кожному кроці пошуку хордою, перетин якої з віссю X дає наближення кореня.

При цьому в процесі пошуку сімейство хорд може будуватися:

- при фіксованому лівому кінці хорд, тобто. $z = a$ тоді початкова точка $x0 = b$;
- при фіксованому правому кінці хорд, тобто. $z = b$ тоді початкова точка $x0 = a$.

В результаті ітераційний процес сходження до кореня реалізується рекурентною формулою:

$$x_{n+1} = x_n - \frac{f(x_n)}{f(x_n)-f(a)}(x_n - a) \quad \text{для случая а);}$$
$$x_{n+1} = x_n - \frac{f(x_n)}{f(x_n)-f(b)}(x_n - b) \quad \text{для случая б);}$$

Процес пошуку триває доти, доки не виконається умова

$$|x_{n+1} - x_n| \le \varepsilon \text{ или } |h| \le \varepsilon.$$

Метод забезпечує швидку збіжність, якщо $f(z) \cdot f''(z) > 0$, тобто. хорди фіксуються на тому кінці інтервалу $[a, b]$ де знаки функції $f(z)$ та її кривизни $f''(z)$ збігаються.

8.2 Чисельне інтегрування

Завдання чисельного інтегрування полягає у заміні вихідної підінтегральної функції $f(x)$, для якої важко або неможливо записати первісну в аналітиці, деякою функцією, що апроксимує $\varphi(x)$. Такою функцією зазвичай є поліном (шматковий поліном) $\varphi(x) = \sum_{i=1}^{n} c_i \phi_i(x)$.

Таким чином

$$\mathrm{I} = \int_a^b f(x)dx = \int_a^b \phi(x)dx + R,$$

де $R = \int_a^b r(x)dx$ - апріорна похибка методу на інтервалі інтегрування, а $r(x)$ - апріорна похибка методу на окремому етапі інтегрування.

8.2.1 Огляд методів інтегрування

Методи обчислення одноразових інтегралів називаються квадратурними (для кратних інтегралів — кубатурними), поділяються на такі групи:

- Методи Ньютона-Котеса. Тут $\varphi(x)$ - Поліном різних ступенів. Сюди належать метод прямокутників, трапецій, Сімпсона.
- Методи статистичних випробувань (методи Монте-Карло). Тут вузли сітки для квадратурного чи кубатурного інтегрування вибираються з допомогою датчика випадкових чисел, відповідь має імовірнісний характер. В основному використовуються для обчислення кратних інтегралів.
- Сплайнові методи. Тут $\varphi(x)$ - кусковий поліном з умовами зв'язку між окремими поліномами за допомогою системи коефіцієнтів.
- Методи найвищої точності алгебри. Забезпечують оптимальне розміщення вузлів сітки інтегрування та вибір вагових коефіцієнтів $\rho(x)$ у завданні $\int_a^b \phi(x)\rho(x)dx$ (Характерний приклад - метод Гауса).

8.2.2 Метод прямокутників

Розрізняють метод лівих, правих та середніх прямокутників. Суть методу зрозуміла з малюнка. На кожному етапі інтегрування функція апроксимується поліномом нульового ступеня - відрізком, паралельним осі абсцис.

Формули методу прямокутників можна отримати з аналізу розкладання функції $f(x)$ у ряд Тейлора поблизу деякої точки $x = x_i$:

$$f(x)|_{x=x_i} = f(x_i)+(x-x_i)\,f'(x_i)+\frac{(x-x_i)^2}{2!}f''(x_i)+\dots.$$

Розглянемо діапазон інтегрування від x_i до x_i+h, де h - Крок інтегрування. Обчислимо інтеграл від досліджуваної функції у цьому проміжку:

$$\int\limits_{x_i}^{x_i+h} f(x)dx = x\cdot f(x_i)|_{x_i}^{x_i+h} + \frac{(x-x_i)^2}{2}\,f'(x_i)|_{x_i}^{x_i+h} +$$
$$+\frac{(x-x_i)^3}{3\cdot 2!}\,f''(x_i)|_{x_i}^{x_i+h}+\dots =$$
$$= f(x_i)h+\frac{h^2}{2}f'(x_i)+O(h^3) = f(x_i)h+r_i.$$

таким чином, на базі аналізу ряду Тейлора отримано формулу правих (або лівих) прямокутників та апріорну оцінку похибки r на окремому етапі інтегрування. Основний критерій, за яким судять про точність алгоритму - ступінь за величиною кроку у формулі апріорної оцінки похибки. У разі рівного кроку h на всьому діапазоні інтегрування загальна формула має вигляд

$$\int\limits_a^b f(x)dx = h\sum_{i=0}^{n-1} f(x_i)+R,$$

де n - число розбиття інтервалу інтегрування, $R=\sum\limits_{i=0}^{n-1} r_i = \frac{h}{2}\cdot h\sum\limits_{i=0}^{n-1} f'(x_i) = \frac{h}{2}\int\limits_a^b f'(x)dx$. Отримана оцінка справедлива за наявності безперервної похідної підінтегральної функції $f'(x)$.

8.2.3 Метод середніх прямокутників

Тут на кожному інтервалі значення функції вважається в середній точці відрізка $[x_i, x_i+h]$, тобто $\int\limits_{x_i}^{x_i+h} f(x)dx = hf(\bar{x})+r_i$.

Розкладання функції ряд Тейлора показує, що у разі середніх прямокутників точність методу значно вище:

$$r=\frac{h^3}{24}f''(\bar{x}),\ R=\frac{h^2}{24}\int\limits_a^b f''(x)dx.$$

Приклад функції Maxima, що реалізує метод середніх прямокутників, наведено нижче:

```
(%i1)    intpr(f,n,a,b):=block([h,i,s,_x],h:(ba)/n,
_x:a+h/2, s:0,
    for i:1 thru n do
(s:s+float(subst(_x,x,f)),_x:_x+h),s:s*h)$
(%i2)    intpr(x^2,100,-1,1);
```

(%o2) 0.6666

8.2.4 Метод трапецій

Апроксимація у цьому методі здійснюється поліномом першого ступеня. На одиничному інтервалі

$$\int_{x_i}^{x_i+h} f(x)dx = \frac{h}{2}\left(f(x_i) + f(x_i+h)\right) + r_i.$$

У разі рівномірної сітки (h = const)

$$\int_a^b f(x)dx = h\left(\frac{1}{2}f(x_0) + \sum_{i=1}^{n-1} f(x_i) + \frac{1}{2}f(x_n)\right) + R$$

При цьому $r_i = -\frac{h^3}{12}f''(x_i)$, а $R = -\frac{h^3}{12}\int_a^b f''(x)dx$.

Похибка методу трапецій вдвічі вища, ніж у методу середніх прямокутників. Однак на практиці знайти середнє значення на елементарному інтервалі можна тільки у функцій, заданих аналітично (а не таблично), тому використовувати метод середніх прямокутників вдається далеко не завжди. У силу різних знаків похибки у формулах трапецій та середніх прямокутників справжнє значення інтегралу зазвичай лежить між двома цими оцінками.

Приклад реалізації методу трапецій у вигляді функції наведено нижче:

```
(%i1)    f:x^2;
```

(%o1) x^2

```
(%i2)    inttrap(f,n,a,b):=block([h,i,s],h:(ba)/n,
     s:(float(subst(a,x,f))+float(subst(b,x,f)))/2,
     for i:1 до n-1 do (s:s+float(subst(a+i*h,x,f))),
s:s*h)$
(%i3)    inttrap(f,100,-1,1);
```

(%o3) 0.6668

Підінтегральна функція задається як виразу Maxima. Вираз, що визначає підінтегральну функцію, можна ставити безпосередньо при зверненні до методу, як у наступному прикладі:

```
(% i4)   inttrap(x*sin(x),100,-1,1);
```

(%o4) 0.60242947746101

8.2.5 Метод Сімпсона

При використанні цього методу підінтегральна функція $f(x)$ замінюється інтерполяційним поліномом другого ступеня $P(x)$ — параболою, що проходить через три сусідні вузли. Розглянемо два кроки інтегрування (

$h = const = x_{i+1} - x_i$), тобто три вузли x_0, x_1, x_2, через які проведемо параболу, скориставшись рівнянням Ньютона:

$$P(x) = f_0 + \frac{x - x_0}{h}(f_1 - f_0) + \frac{(x - x_0)(x - x_1)}{2h^2}(f_0 - 2f_1 + f_2).$$

Нехай $z = x - x_0$ тоді

$$P(z) = f_0 + \frac{z}{h}(f_1 - f_0) + \frac{z(z-h)}{2h^2}(f_0 - 2f_1 + f_2) =$$
$$= f_0 + \frac{z}{2h}(-3f_0 + 4f_1 - f_2) + \frac{z^2}{2h^2}(f_0 - 2f_1 + f_2)$$

Скориставшись отриманим співвідношенням, обчислимо інтеграл за цим інтервалом:

$$\int_{x_0}^{x_2} P(x)dx = \int_0^{2h} P(z)dz =$$
$$= 2hf_0 + \frac{(2h)^2}{4h}(-3f_0 + 4f_1 - f_2) + \frac{(2h)^3}{6h^2}(f_0 - 2f_1 + f_2) =$$
$$= 2hf_0 + h(-3f_0 + 4f_1 - f_2) + \frac{4h}{3}(f_0 - 2f_1 + f_2) =$$
$$= \frac{h}{3}(6f_0 - 9f_0 + 12f_1 - 3f_2 + 4f_0 - 8f_1 + 4f_2).$$

$$\int_{x_0}^{x_2} f(x)dx = \frac{h}{3}(f_0 + 4f_1 + f_2) + r$$

У результаті .

Для рівномірної сітки та парного числа кроків n формула Сімпсона набуває вигляду:

$$\int_a^b f(x)dx = \frac{h}{3}(f_0 + 4f_1 + 2f_2 + 4f_3 + \cdots + 2f_{n-2} + 4f_{n-1} + f_n) + R,$$

де $r = -\frac{h^5}{90} f^{IV}(x_i)$, а $R = -\frac{h^4}{180}\int_a^b f^{IV}(x)dx$ у припущенні безперервності четвертої похідної підінтегральної функції.

Реалізація методу Сімпсона засобами Maxima представлена в наступному прикладі:

```
(%i1)    intsimp(f,n,a,b):=block([h,h2,i,_x,s],h:(ba)/n,
h2:h/2,
     s:(float(subst(a,x,f))+float(subst(b,x,f)))/2+
                  2*float(subst(a+h2,x,f)),
     _x:a,
     for i:1 thru n-1 do (_x:_x+h,

     s:s+2*float(subst(_x+h2,x,f))+float(subst(_x,x,f))),
     s:s*h/3)$
(%i2)    intsimp(x^2,100,-1,1);
(%o2) 0.666666666666667
```

8.3 Методи розв'язання систем лінійних рівнянь

8.3.1 Загальна характеристика та класифікація методів вирішення

Розглянемо систему лінійних рівнянь алгебри (скорочено — СЛАУ):

$$A \cdot \bar{x} = \bar{f}, \quad (8.1)$$

де A - матриця $m \times m$, $\bar{x} = (x_1, x_2, \ldots, x_m)^T$ - Шуканий вектор, $\bar{f} = (f_1, f_2, \ldots, f_m)^T$ - Вектор заданий. Припускатимемо, що визначник матриці А відмінний від нуля, тобто. Рішення системи (8.1) існує.

Методи чисельного рішення системи (8.1) поділяються на дві групи: прямі методи ("точні") та ітераційні методи.

Прямими методами називаються методи, що дозволяють отримати рішення системи (8.1) за кінцеве число арифметичних операцій. До цих методів належать метод Крамера, метод Гауса, LU-метод і т.д.

Ітераційні методи (методи послідовних наближень) полягають у тому, що рішення системи (8.1) перебуває як межа послідовних наближень $\bar{x}^{(n)}$ при $n \to \infty$, де n - Номер ітерації. При використанні методів ітерації зазвичай задається деяка мала кількість ε та обчислення проводяться доти, доки не буде виконано оцінку $\left\| \bar{x}^{(n)} - \bar{x} \right\| < \varepsilon$. До цих методів належать метод Зейделя, Якобі, метод верхніх релаксацій тощо.

Слід зазначити, що реалізація прямих методів комп'ютері призводить до рішення з похибкою, т.к. всі арифметичні операції над змінними з плаваючою точкою виконуються із заокругленням. Залежно від властивостей матриці вихідної системи, ці похибки можуть досягати значних величин.

8.3.2 Метод Гауса

Напишемо систему (8.1), у розгорнутому вигляді

$$\begin{aligned}
&a_{11}x_1 + a_{12}x_2 + \cdots + a_{1m}x_m = f_1,\\
&a_{21}x_1 + a_{22}x_2 + \cdots + a_{2m}x_m = f_2,\\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\\
&a_{m1}x_1 + a_{m2}x_2 + \cdots + a_{mm}x_m = f_m.
\end{aligned}$$

Метод Гаусса полягає у послідовному виключенні невідомих із цієї системи. Припустимо, що $a_{11} \neq 0$. Послідовно помножуючи перше рівняння на $-\frac{a_{i1}}{a_{11}}$ і складаючи з i -м рівнянням, виключимо x_1 із усіх рівнянь крім першого. Отримаємо систему

$$\begin{aligned}
&a_{11}x_1 + a_{12}x_2 + \cdots + a_{1m}x_m = f_1,\\
&a_{22}^{(1)}x_2 + \cdots + a_{2m}^{(1)}x_m = f_2^{(1)},\\
&\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\\
&a_{m2}^{(1)}x_2 + \cdots + a_{mm}^{(1)}x_m = f_m^{(1)}.
\end{aligned}$$

де

$$a_{ij}^{(1)} = a_{ij} - \frac{a_{i1}a_{1j}}{a_{11}}, \quad f_i^{(1)} = f_i - \frac{a_{i1}f_1}{a_{11}}, \quad i,j = 2,3,\ldots,m.$$

Аналогічним чином із отриманої системи виключимо x_2. Послідовно, крім всіх невідомих, отримаємо систему трикутного вигляду

$$\begin{aligned}
a_{11}x_1 + a_{12}x_2 + \cdots + a_{1k}x_k + \cdots + a_{1m}x_m &= f_1,\\
a_{22}^{(1)}x_2 + \cdots + a_{2k}^{(1)}x_k + \cdots + a_{2m}^{(1)}x_m &= f_2^{(1)},\\
\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots\ldots&,\\
a_{m-1,m-1}^{(m-1)}x_{m-1} + a_{m-1,m}^{(m-1)}x_m &= f_{m-1}^{(m-1)},\\
a_{m,m}^{(m-1)}x_m &= f_m^{(m-1)}.
\end{aligned}$$

Описана процедура називається прямим перебігом методу Гаусса. Зауважимо, що її виконання було можливе за умови, що все $a_{i,i}^{(l)},\ l = 1,2,\ldots,m-1$ не дорівнюють нулю. Виконуючи послідовні підстановки в останній системі (починаючи з останнього рівняння) можна отримати всі значення невідомих.

$$x_m = \frac{f_m^{(m-1)}}{a_{m,m}^{(m-1)}},$$

$$x_i = \frac{1}{a_{i,i}^{(i-1)}}\left(f_i^{(i-1)} - \sum_{j=i-1}^{m} a_{ij}^{(i-1)}x_j\right).$$

Ця процедура одержала назву зворотний хід методу Гаусса.

Метод Гауса може бути легко реалізований на комп'ютері. При виконанні обчислень, як правило, проміжні значення матриці А не становлять інтересу. Тому чисельна реалізація методу зводиться до перетворення елементів масиву розмірності $m \times (m+1)$, де $m+1$-й стовпець містить елементи правої частини системи

Для контролю помилки реалізації методу використовуються так звані контрольні суми. Схема контролю ґрунтується на наступному очевидному положенні. Збільшення значення всіх невідомих на одиницю дорівнює заміні даної системи контрольною системою, в якій вільні члени дорівнюють сумам всіх коефіцієнтів відповідного рядка. Створимо додатковий стовпець, що зберігає суму елементів матриці рядками. На кожному кроці реалізації прямого ходу методу Гауса виконуватимемо перетворення і над елементами цього стовпця, і порівнюватимемо їх значення із сумою по рядку перетвореної матриці. У разі не збігу значень рахунок переривається.

Один із основних недоліків методу Гауса пов'язаний з тим, що при його реалізації накопичується обчислювальна похибка.

Для систем порядку m число дій множення та поділу близько $\frac{m^3}{3}$ та швидко зростає з величиною m .

Щоб зменшити зростання обчислювальної похибки застосовуються різні модифікації методу Гаусса. Наприклад, метод Гауса з вибором головного елемента по стовпцях, у цьому випадку на кожному етапі прямого ходу рядка матриці переставляються таким чином, щоб діагональний кутовий елемент був максимальним. За винятком відповідного невідомого з інших рядків поділ проводитиметься на найбільший із можливих коефіцієнтів і, отже, відносна похибка буде найменшою.

Приклад реалізації методу Гауса в Maxima наведено у функції нижче (застосований метод без вибору головного елемента):

```
(%i1)    gauss(a0,b0,n):=block([a,b,i,j,k,d],
    a:copymatrix (a0), b:copymatrix(b0),
x:copymatrix(b0),
    for i:1 thru n-1 do
    (
        for k:i+1 thru n do
        (
            d:a[k,i]/a[i,i],
            for j:i+1 thru n do (a[k,j]:a[k,j]-
a[i,j]*d),
            b[k,1]:b[k,1]-b[i,1]*d
        )
    ),
    for i:n thru 1 step -1 do
    (
        for j:i+1 thru n do (
            b[i,1]:b[i,1]-a[i,j]*x[j,1]),
        x[i,1]:b[i,1]/a[i,i]
    ),
    x)$
```

прикладзвернення до функції, що реалізує метод Гауса:

```
(%i2)    aa:matrix([3,1,1],[1,3,1],[1,1,3]); bb: matrix
([6], [6], [8]);
    zz: gauss (aa, bb, 3);
```

$$(\%o2)\ \begin{pmatrix}3 & 1 & 1\\ 1 & 3 & 1\\ 1 & 1 & 3\end{pmatrix}\quad (\%o3)\ \begin{pmatrix}6\\ 6\\ 8\end{pmatrix}\quad (\%o4)\ \begin{pmatrix}1\\ 1\\ 2\end{pmatrix}$$

Перевірка обчислень показує, що перемноження матриці A на вектор рішення zz дає вектор, що збігається з вектором правих частин bb .

```
(% i5)   aa.zz;
```

$$(\%o5)\ \begin{pmatrix}6.0\\ 6.0\\ 8.0\end{pmatrix}$$

Існує метод Гауса з вибором головного елемента по всій матриці. І тут переставляються як рядки, а й стовпці. Використання модифікацій методу Гаусса призводить до ускладнення алгоритму збільшення числа операцій та відповідно до зростання часу рахунку.

Перетворення прямого ходу, що виконуються в методі Гауса, що привели матрицю A системи до трикутного вигляду дозволяють обчислити визначник матриці

$$\det A = \begin{vmatrix} a_{11} & a_{12} & \dots & a_{1m} \\ 0 & a_{22}^{(1)} & \dots & a_{2m}^{(1)} \\ \dots & \dots & \dots & \dots \\ 0 & 0 & \dots & a_{m,m}^{(m-1)} \end{vmatrix} = a_{11} \cdot a_{22}^{(1)} \dots a_{m,m}^{(m-1)}$$

Метод Гаус дозволяє знайти і зворотну матрицю. Для цього необхідно вирішити матричне рівняння

$$A \cdot X = E,$$

де E - Поодинока матриця. Його рішення зводиться до рішення m систем

$$A\bar{x}^{(j)} = \bar{\delta}^{(j)}, \quad j = 1, 2, \dots, m,$$

у вектора $\bar{\delta}^{(j)}$ j -я компонента дорівнює одиниці, а інші компоненти дорівнюють нулю.

8.3.3 Метод квадратного кореня

Метод квадратного кореня заснований на розкладанні матриці A у твір

$$A = S^T S,$$

де S - Верхня трикутна матриця з позитивними елементами на головній діагоналі, S^T - транспонована до неї матриця.

Нехай A - матриця розміром $m \times m$. Тоді

$$\left(S^T S\right)_{ij} = \sum_{k=1}^{m} s_{ik}^T s_{kj} \qquad (8.2)$$

З умови (8.2) виходять рівняння

$$\sum_{k=1}^{m} s_{ik}^T s_{kj} = a_{ij}, \qquad i, j = 1, 2, \dots .., m \qquad (8.3)$$

Оскільки матриця A симетрична, не обмежуючи спільності, вважатимуться, що у системі (8.3) виконується нерівність $i \le j$. Тоді (8.3) можна переписати у вигляді

$$\sum_{k=1}^{i-1} s_{ik}^T s_{kj} + s_{ii} s_{ij} + \sum_{k=i+1}^{m} s_{ik}^T s_{kj} = a_{ij}$$

$$s_{ii} s_{ij} + \sum_{k=1}^{i-1} s_{ik}^T s_{kj} = a_{ij}, \quad i \leqslant j.$$

Зокрема, при $i = j$ вийде

$$|s_{ii}|^2 = a_{ii} - \sum_{k=1}^{i-1} |s_{ki}|^2$$

$$s_{ii} = \left(\left| a_{ii} - \sum_{k=1}^{i-1} |s_{ki}|^2 \right| \right)^{1/2}$$

Далі, при $i < j$ отримаємо

$$s_{ij} = \frac{a_{ij} - \sum_{k=1}^{i-1} s_{ik}^T s_{kj}}{s_{ii}}$$

За наведеними формулами знаходяться рекурентно всі ненульові елементи матриці S.

Зворотний хід методу квадратного кореня полягає у послідовному розв'язанні двох систем рівнянь із трикутними матрицями.

$$S^T y = f,$$

$$Sx = y.$$

Рішення цих систем знаходяться за рекурентними формулами:

$$\begin{cases} y_i = \dfrac{f_i - \sum_{k=1}^{i-1} s_{ki} y_k}{s_{ii}}, \quad i = 2, 3, \ldots, m \\ y_1 = \dfrac{f_1}{s_{11}} \end{cases}$$

$$\begin{cases} x_i = \dfrac{y_i - \sum_{k=i+1}^{m} s_{ik} x_k}{s_{ii}}, \quad i = m-1, m-2, \ldots, 1 \\ x_m = \dfrac{y_m}{s_{mm}} \end{cases}$$

Усього метод квадратного кореня при факторизації $A = S^T S$ вимагає $\dfrac{m^3}{3}$ приблизно операцій множення та поділу та m операцій вилучення квадратного кореня.

Приклад функції, що реалізує метод квадратного кореня:

```
(%i1)    holetsk(a0,b0):=block([L,Lt,x,y,i,j,k,n],
    n:length(a0), L:zeromatrix(n,n),
    for i:1 thru n do (
        for j:1 thru i-1 do (
        s:0,
        for k:1 до j-1 do (s:s+L[i,k]*L[j,k]),
```

```
        L[i,j]:1/L[j,j]*(a0[i,j]-s)
        ),
        s:0,
        for k:1 thru i-1 do (s:s+L[i,k]^2),
        L[i,i]:sqrt(a0[i,i]-s)
        ),Lt:transpose(L),
    y:zeromatrix(n,1), x:zeromatrix(n,1),
    for i:1 thru n do (
        s:0,
        for k:1 thru i-1 do (s:s+L[i,k]*y[k,1]),
            y[i,1]:(b0[i,1]-s)/L[i,i]
    ),
    for i:n thru 1 step -1 do (
        s:0,
        for k:n thru i+1 Step -1 do
(s:s+Lt[i,k]*x[k,1]),
        x[i,1]:(y[i,1]-s)/Lt[i,i]
    ), x
    )$
```

Тест цієї функції (вирішення системи $Ax = B, B$ - матриця $n \times 1,\ A$ - Квадратна симетрична матриця $n \times n$, результат рішення $n \times 1$):

```
(%i2)    A:matrix([4,1,1],[1,4,1],[1,1,4])$
B:matrix([1],[1],[1])$
```

Результати обчислень:

```
(% i4)   x:holetsk(A,B);
```

$$(\%o4)\ \begin{pmatrix} \frac{1}{6} \\ \frac{1}{6} \\ \frac{1}{6} \end{pmatrix}$$

Перевірка:

```
(% i5)   Ax;
```

$$(\%o5)\ \begin{pmatrix} 1 \\ 1 \\ 1 \end{pmatrix}$$

8.3.4 Коректність постановки задачі та поняття обумовленості

При використанні чисельних методів для вирішення тих чи інших математичних завдань необхідно розрізняти властивості самої задачі та властивості обчислювального алгоритму, призначеного для її вирішення.

Кажуть, що завдання поставлене коректно, якщо рішення існує і єдине, і якщо воно безперервно залежить від вхідних даних. Остання властивість називається також стійкістю щодо вхідних даних.

Коректність вихідної математичної завдання ще гарантує хороших властивостей чисельного методу її рішень і вимагає спеціального дослідження.

Відомо, що розв'язання задачі (8.1) існує тоді і лише тоді, коли $\det A \neq 0$. У цьому випадку можна визначити зворотну матрицю A^{-1} та рішення записати у вигляді $\bar{x} = A^{-1}\bar{f}$.

Дослідження стійкість завдання (8.1) зводиться до вивчення залежності її вирішення від правих частин $\bar{f}$ та елементів a_{ij} матриці A. Для того, щоб можна було говорити про безперервну залежність вектора рішень від деяких параметрів, необхідно на безлічі m-мірних векторів, що належать лінійному простору $\mathbb{H}$ ввести метрику.

У лінійній алгебрі пропонується визначення безлічі метрик l_p - норма $\|\bar{x}\|_p = \left(\sum_{i=1}^{m} |x_i|^p\right)^{1/p}$ з якого легко отримати метрики, що найчастіше використовуються

- при $p = 1,\ \|\bar{x}\|_1 = \sum_{i=1}^{m} |x_i|$,
- при

$$p = 2,\ \|\bar{x}\|_2 = \left(\sum_{i=1}^{m} |x_i|^2\right)^{1/2}$$

- при

$$p \to \infty,\ \|\bar{x}\|_\infty = \lim_{p\to\infty} \left(\sum_{i=1}^{m} |x_i|^p\right)^{1/p}$$

.

Підпорядковані норми матриць визначаються як $\|A\| = \sup_{0\neq x\in\mathbb{H}} \frac{\|A\bar{x}\|}{\|\bar{x}\|}$ відповідно записуються в наступному вигляді:

$$\|A\|_1 = \max_{1 \leqslant j \leqslant m} \sum_{i=1}^{m} |a_{ij}|,$$

$$\|A\|_\infty = \max_{1 \leqslant i \leqslant m} \sum_{i=1}^{m} |a_{ij}|,$$

$$\|A\|_2 = \sqrt{\rho(A^T A)} = \sqrt{\sum_{i=1}^{m} \sum_{j=1}^{m} a_{ij}^2}.$$

Зазвичай розглядають два види стійкості розв'язання системи (8.1):

- з правих частин;
- за коефіцієнтами системи (8.1) та з правих частин.

Поряд із вихідною системою (8.1) розглянемо систему з "обуреними" правими частинами

$A \cdot \tilde{x} = \tilde{f},$

де $\tilde{f} = \bar{f} + \delta \bar{f}$ обурена права частина системи, а $\tilde{x} = \bar{x} + \delta \bar{x}$ обурене рішення.

Можна отримати оцінку, що виражає залежність відносної похибки рішення від відносної похибки правих частин

$$\frac{\|\delta \bar{x}\|}{\|\bar{x}\|} \leqslant M_A \frac{\|\delta \bar{f}\|}{\|\bar{f}\|},$$

де $M_A = \|A\| \cdot \left\|A^{-1}\right\|$ - Число обумовленості матриці A (У сучасній літературі це число позначають як $cond(A)$). Якщо число обумовленості велике ($M_A \sim 10^k, \quad k > 2$), то кажуть, що матриця A погано зумовлена. І тут малі обурення правих елементів системи (8.1), викликані чи неточністю завдання вихідних даних, чи викликані похибками обчислення істотно впливають рішення системи.

Якщо обурення внесено до матриці A , то щодо відносних обурень рішення має місце така оцінка:

$$\frac{\|\delta \bar{x}\|}{\|\bar{x}\|} \leqslant \frac{M_A}{1 - M_A \dfrac{\|\delta A\|}{\|A\|}} \left(\frac{\|\delta A\|}{\|A\|} + \frac{\|\delta \bar{f}\|}{\|\bar{f}\|} \right).$$

Maxima матричні норми обчислюються за допомогою функції mat_norm. Синтаксис виклику: $mat_norm(M, type)$, де M - матриця, $type$ - Тип норми, $type$ може дорівнювати 1 (норма $\|A\|_1$), inf (норма $\|A\|_\infty$), $frobenius$ (норма $\|A\|_2$).

Приклад обчислення зазначених видів норми Maxima:

```
(%i1)    A: matrix ([1,2,3], [4,5,6], [7,8,9]);
```

$$(\%o1)\ \begin{pmatrix} 1 & 2 & 3 \\ 4 & 5 & 6 \\ 7 & 8 & 9 \end{pmatrix}$$

```
(%i2)    mat_norm(A,1);
```

$$(\%o2)\ 18$$

```
(%i3)    mat_norm(A,inf);
```

$$(\%o3)\ 24$$

```
(% i4)   mat_norm(A,frobenius);
```

$$(\%o4)\ \sqrt{285}$$

Обчислимо число обумовленості для погано та добре обумовлених матриць:

```
(%i1)    A: matrix ([1,1], [0.99,1]);
```

$$(\%o1)\ \begin{pmatrix} 1 & 1 \\ 0.99 & 1 \end{pmatrix}$$

```
(%i2)    nrA:mat_norm(A,frobenius);
```

$$(\%o2)\ 1.995018796903929$$

```
(%i3)    A1: invert (A);
```

$$(\%o3)\ \begin{pmatrix} 99.99999999999992 & -99.99999999999992 \\ -98.99999999999992 & 99.99999999999992 \end{pmatrix}$$

```
(% i4)   nrA1:mat_norm(A1,frobenius);
```

$$(\%o4)\ 199.5018796903927$$

```
(% i5)   MA: nrA * nrA1;
```

$$(\%o5)\ 398.0099999999997$$

Таким чином, для погано обумовленої матриці число обумовленості сягає майже 400.

Аналогічним шляхом (з використанням норми Фробеніуса) для матриці $B = \begin{bmatrix} 1 & 0 \\ 0 & 1 \end{bmatrix}$ число обумовленості становило $MB = 2$.

8.3.5 Про обчислювальні витрати

Один з важливих факторів, що зумовлюють вибір того чи іншого методу при вирішенні конкретних завдань, є обчислювальна ефективність методу. Враховуючи, що операція додавання виконується набагато швидше, ніж операція множення та поділу, зазвичай обмежуються підрахунком останніх. Для вирішення СЛАУ методом Гауса без вибору головного елемента потрібно $\frac{m^3}{3} + m^2 - \frac{m}{3}$ множень та поділів, рішення СЛАУ методом квадратного кореня вимагає $\frac{m^3}{6} + \frac{3m^2}{2} + \frac{m}{3}$ і m операцій вилучення коренів. При великих

значеннях розмірності m, можна сказати, що обчислювальні витрати на операції множення та поділу у методі Гаусса становлять величину $O\left(\frac{m^3}{3}\right)$, у методі квадратного коріння $O\left(\frac{m^3}{6}\right)$.

8.4 Ітераційні методи

У наближених або ітераційних методах рішення системи лінійних рівнянь алгебри є межею ітераційної послідовності, одержуваної за допомогою цих методів. До них відносяться: метод простої ітерації, метод Зейделя та ін. $10^3 \dots 10^5$.

Для ітераційних методів характерно те, що вони вимагають початкових наближень значень невідомих, рішення шукається у вигляді послідовності, що наближень поступово поліпшуються, і крім того, ітераційний процес повинен бути схожим. У обчислювальній практиці процес ітерації зазвичай триває до тих пір, поки два послідовні наближення не співпадають у межах заданої точності.

8.4.1 Матричне формулювання ітераційних методів розв'язання систем лінійних рівнянь

При використанні СКМ Maxima цілком обґрунтовано використання матричного формулювання ітераційних методів.

Розглянемо рішення системи $Ax = f$ (A - Квадратна матриця, f вектор правих частин, x - Вектор невідомих). Позначимо $A = L + D + U$, де L - нижня трикутна матриця з нульовими діагональними елементами; D - Діагональна матриця; U - Верхня трикутна матриця з нульовими діагональними елементами.

Для вирішення цієї системи розглянемо ітераційний процес

$$x^{i+1} = x^i - H_i(Ax^i - f),$$

де $\det(H_i) \neq 0$, або

$$x^{i+1} = P_i x^i + d_i, \qquad x(0) = x_0,$$

де $P_i = I - P_i A$ - Оператор i-го кроку ітераційного процесу; $d_i = P_i A$.

Ітераційний процес схожий, якщо послідовність $\{x_i\}$ сходиться до рішення x^* за будь-якого x_0.

Якщо матриця залежить від номера ітерації, ітераційний процес називається стаціонарним:

$$x^{i+1} = Px^i + d. \quad (8.4)$$

Необхідною та достатньою умовою збіжності стаціонарного процесу є виконання умови $\rho(P) < 1$, де $\rho(P)$- Спектральний радіус матриці P (Найбільше за модулем власне число матриці P).

З використанням введених позначень метод простої ітерації (метод Якобі) надається формулою:

$$P = I - D^{-1}A, \qquad D = diag(a_{ii}), \qquad H = D^{-1},$$

а метод Гауса-Зейделя - формулою:

$$P = -(D + L)^{-1}U, \qquad H = (D + L)^{-1}.$$

Розглянемо поелементні розрахункові співвідношення для методів Якобі та Гауса-Зейделя.

Усі елементи головної діагоналі матриці $I = D^{-1}A$ рівні нулю, інші елементи рівні $-\frac{a_{ij}}{a_{ii}}, \ i, j = \overrightarrow{1, n}$. Вільний член рівняння (8.4) дорівнює $\frac{f_i}{a_{ii}}$.

Таким чином, для методу Якобі ітераційний процес записується у вигляді $x^{k+1} = Cx^k +$, де $C_{ij} = -\frac{a_{ij}}{a_{ii}}; \ k = 0, 1, \ldots, \ i, j = \overrightarrow{1, n}; \ E_i = \frac{f_i}{a_{ii}}$.

Для методу Гауса-Зейделя $x^{k+1} = -(D + L)^{-1}Ux^k + (D + L)^{-1}f$, або $(D + L)x^{k+1} = -Ux^k + b$, $x^{k+1} = Bx^{k+1} + Ex^k + e$, де

$$B = \begin{bmatrix} 0 & 0 & \ldots & 0 \\ c_{21} & 0 & \ldots & 0 \\ c_{31} & c_{32} & \ldots & 0 \\ \ldots & \ldots & \ldots & \ldots \\ c_{n-1,1} & c_{n-1,2} & \ldots & 0 \\ c_{n1} & c_{n2} & \ldots & 0 \end{bmatrix}, \text{ и } E = \begin{bmatrix} 0 & c_{21} & \ldots & \ldots & c_{1n} \\ 0 & 0 & \ldots & \ldots & c_{2n} \\ 0 & & \ddots & & \\ \vdots & & & \ddots & c_{n-1,n} \\ 0 & & & & 0 \end{bmatrix}.$$

Розглянемо розв'язання конкретної системи рівнянь $Ax = b$ методом Якобі:

$$\begin{aligned} 8x_1 - x_2 + 2x_3 &= 8, \\ x_1 + 9x_2 + 3x_3 &= 18, \\ 2x_1 - 3x_2 + 10x_3 &= -5. \end{aligned}$$

Обчислюємо елементи матриці B та вектора e:

$$B = \begin{bmatrix} 0 & \frac{1}{8} & -\frac{2}{8} \\ -\frac{1}{9} & 0 & -\frac{3}{9} \\ -0,2 & 0,3 & 0 \end{bmatrix}, \quad e = \begin{bmatrix} 1 \\ 2 \\ 0,5 \end{bmatrix}.$$

Обчислимо значення x за формулою $x^{k+1} = Bx^k + e$. Для вирішення використана наступна послідовність команд Maxima:

перетворення заданих матриць

```
(%i1)    A:matrix([8,-1,2],[1,9,3],[2,-3,10])$
     b:matrix([8],[18],[5])$
     A0: matrix([A[1,1],A[1,1],A[1,1]],
          [A[2,2],A[2,2],A[2,2]],
          [A[3,3],A[3,3],A[3,3]])$
     B:-A/A0+diagmatrix(3,1)$
     e:b/matrix([A[1,1]],[A[2,2]],[A[3,3]])$ x:e$
```

власне обчислення рішення

```
(%i7)    xt:float(B.x+e)$
     xt:float(B.xt+e)$
     xt:float(B.xt+e)$
     xt:float(B.xt+e)$
     xt:float(B.xt+e)$
     xt:float(B.xt+e)$
     xt:float(B.xt+e)$
     x0:xt$
     xt:float(B.xt+e)$
     x1:xt$
     r:x1-x0$
     float(r); / * Оцінка збіжності * /
     float(A.x1-b); /* оцінка нев'язки*/
```

$$(\%o18)\quad \begin{pmatrix} -1.2272477756924971\,10^{-5} \\ -1.3018148195786949\,10^{-4} \\ 6.2047575160040225\,10^{-5} \end{pmatrix}$$

$$(\%o19)\quad \begin{pmatrix} 2.5427663227794994\,10^{-4} \\ 1.7387024772119 73\,10^{-4} \\ 3.6599949035931445\,10^{-4} \end{pmatrix}$$

8.4.2 Метод простої ітерації

Для вирішення системи лінійних рівнянь алгебри (8.1) ітераційним методом її необхідно привести до нормального вигляду:

$\bar{x} = P\bar{x} + \bar{g}$ (8.5)

Стаціонарне ітераційне правило отримуємо, якщо матриця B та вектор $\bar{g}$ не залежать від номера ітерації: $\bar{x}^{k+1} = P\bar{x}^k + \bar{g}$. Нестаціонарне ітераційне правило отримуємо якщо матриця B або вектор $\bar{g}$ змінюються із зростанням номера ітерації: $\bar{x}^{k+1} = P_k\bar{x}^k + \bar{g}^k$.

Стаціонарне ітераційне правило зазвичай називають методом простої ітерації. Межа ітераційної послідовності є точним рішенням системи (8.5) або (8.1).

Для того, щоб метод простої ітерації сходився при будь-якому початковому наближенні, необхідно і достатньо, щоб усі власні значення матриці були по модулю менше одиниці.

З огляду на те, що перевірити сформульовану вище умову, досить складно на практиці застосовують такі достатні ознаки:

- для того щоб метод простої ітерації сходився, достатньо, щоб будь-яка норма матриці P була менше одиниці;
- для того щоб метод простої ітерації сходився, достатньо, щоб виконувалася одна з наступних умов:

$$\sum_{j=1}^{n} |P_{ij}| < 1, \ i = \overline{1,n} \ ;$$

$$\sum_{i=1}^{n} |P_{ij}| < 1, \ j = \overline{1,n} \ ;$$

$$\sum_{i,j=1}^{n} |P_{ij}|^2 < 1 \ .$$

Для визначення швидкості збіжності можна скористатися наступною теоремою: якщо будь-яка норма матриці P, Узгоджена з цією нормою вектора, менше одиниці, то має місце наступна оцінка похибки методу простої ітерації:

$$\left\|\bar{x}^* - \bar{x}^k\right\| < \|P\|^k \cdot \left\|\bar{x}^k\right\| + \frac{\|P\|^k \cdot \|\bar{g}\|}{1 - \|P\|},$$

де $\bar{x}^*$ - Точне рішення системи (8.1).

Іншими словами, умова збіжності виконується, якщо виконується умова домінування діагональних елементів матриці вихідної системи A за рядками $\sum_{i,j=1}^{n} |a_{ij}| < |a_{ii}|$ або стовпцями: $\sum_{i,j=1}^{n} |a_{ij}| \le |a_{jj}|$.

І тут легко можна перейти від системи виду (8.1) до системи (8.5). Для цього розділимо i -е рівняння системи на $a_{i,i}$ і висловимо x_i :

$$x_i = \frac{f_i}{a_{ii}} - \frac{a_{11}}{a_{ii}} x_1 - \ldots - \frac{a_{1n}}{a_{ii}} x_n,$$

тобто. для матриці P буде виконано одну з умов збіжності, де

$$P = \begin{bmatrix} 0 & -\frac{a_{12}}{a_{11}} & \ldots & -\frac{a_{1n}}{a_{11}} \\ -\frac{a_{21}}{a_{22}} & 0 & \ldots & -\frac{a_{2n}}{a_{22}} \\ \ldots & \ldots & \ldots & \ldots \\ -\frac{a_{n1}}{a_{nn}} & -\frac{a_{n2}}{a_{nn}} & \ldots & 0 \end{bmatrix}.$$

Приклад реалізації методу простої ітерації засобами Maxima з друком проміжних результатів наведено в скрипті нижче:

```
(%i1)
    iterpr(a0,b0,x,n,eps):=block([a,b,x0,i,j,s,sum,p],
    a:copymatrix (a0), b:copymatrix(b0),
x0:copymatrix(x),
    sum:1, p:0,
    while sum>eps do (
        sum:0, p:p+1, print("p=",p,"x=",float(x)),
        for i:1 thru n do (
            s:b[i,1],
            for j:1 thru n do (s:sa[i,j]*x0[j,1]),
            s:s/a[i,i], x[i,1]:x0[i,1]+s, sum:sum+abs(s)
        ),
        x0:copymatrix(x)
    ),
    float(x))$
```

8.4.3 Метод Зейделя

У методі Зейделя система (8.1) також наводиться до системи (8.5). Але для обчислення наступної компоненти вектора використовуються вже обчислені компоненти цього вектора.

Ітераційна формула методу в скалярній формі записується так:

$$x_i^{(k+1)} = \sum_{j=1}^{i} p_{ij} x_i^{(k+1)} + \sum_{j=i+1}^{n} p_{ij} x_j^{(k)} + g_i.$$

Встановимо зв'язок між методом Зейделя та методом простої ітерації. Для цього матрицю B представимо у вигляді суми двох матриць: $P = H + F$, де

$$H = \begin{bmatrix} 0 & 0 & 0 & \dots & 0 \\ p_{21} & 0 & 0 & \dots & 0 \\ p_{31} & p_{32} & 0 & \dots & 0 \\ \dots & \dots & \dots & \dots & \dots \\ p_{n1} & p_{n2} & p_{n3} & \dots & 0 \end{bmatrix}; \quad F = \begin{bmatrix} p_{11} & p_{12} & p_{13} & \dots & p_{1n} \\ 0 & p_{22} & p_{23} & \dots & p_{2n} \\ 0 & 0 & p_{33} & \dots & p_{3n} \\ \dots & \dots & \dots & \dots & \dots \\ 0 & 0 & 0 & \dots & p_{nn} \end{bmatrix}.$$

Ітераційна формула методу Зейделя у матричній формі записується у вигляді:

$$\bar{x}^{(k+1)} = H\bar{x}^{(k+1)} + F\bar{x}^{(k)} + \bar{g}, \qquad \text{или}$$

$$(E - H)\,\bar{x}^{(k+1)} = F\bar{x}^{(k)} + \bar{g}, \qquad \text{откуда}$$

$$\bar{x}^{(k+1)} = (E - H)^{-1} F\bar{x}^{(k)} + (E - H)^{-1}\bar{g},$$

тобто. метод Зейделя еквівалентний методу простої ітерації з матрицею $(E - H)^{-1}F$.

Виходячи з отриманої аналогії методів Зейделя та простої ітерації, можна сформулювати наступну ознаку збіжності методу Зейделя: для того, щоб метод Зейделя сходився, необхідно і достатньо, щоб усі власні значення матриці $(E-H)^{-1}F$ за модулем були менше одиниці.

Інакше кажучи, щоб метод Зейделя сходився, потрібно й достатньо, щоб усе коріння рівняння по модулю було менше одиниці, т.к.

$$\left|(E-H)^{-1}F-\lambda E\right| =$$
$$\left|(E-H)^{-1}(E-H)\left[(E-H)^{-1}F-\lambda E\right]\right| =$$
$$\left|(E-H)^{-1}\right|\left|F+\lambda H-\lambda E\right| =$$
$$\left|F+\lambda H-\lambda E\right| = 0.$$

Сформулюємо достатню ознаку збіжності: для того, щоб метод Зейделя сходився, достатньо, щоб виконалася одна з умов:

- $\|P\|_1 = \max\limits_i \sum\limits_{j=1}^{n} |p_{ij}| < 1$
- $\|P\|_2 = \max\limits_j \sum\limits_{i=1}^{n} |p_{ij}| < 1$
- $\|P\|_3 = \sqrt{\sum\limits_{i,j=1}^{n} |p_{ij}|^2} < 1$

При використанні методу Зейделя ітераційний процес сходиться до єдиного рішення швидше методу простих ітерацій.

Приклад реалізації методу Зейделя:

```
(%i1)    seidel(a0,b0,x,n,eps):=block([a,b,i,j,s,sum,p],
    a:copymatrix (a0), b:copymatrix(b0),
    sum:1, p:0,
    while sum>eps do (
        sum:0, p:p+1, print("p= ",p),
        for i:1 thru n do
        (
            s:b[i,1],
            for j:1 thru n do (s:sa[i,j]*x[j,1]),
            s:s/a[i,i], x[i,1]:x[i,1]+s, sum:sum+abs(s)
        )
    ),
    float(x))$
```

Приклад вирішення простої системи методом Зейделя:

```
(%i2)    aa:matrix([3,1,1],[1,3,1],[1,1,3]); bb: matrix
([6], [6], [8]);
```

```
x: matrix ([3], [3], [3]); zz: seidel (aa, bb, x,
3,0.0000001);
```

$$(\%o2)\ \begin{pmatrix}3 & 1 & 1\\ 1 & 3 & 1\\ 1 & 1 & 3\end{pmatrix} \quad (\%o3)\ \begin{pmatrix}6\\ 6\\ 8\end{pmatrix} \quad (\%o4)\ \begin{pmatrix}3\\ 3\\ 3\end{pmatrix}$$

$$p = 1\ p = 2\ p = 3\ p = 4\ p = 5\ p = 6$$

$$p = 7\ p = 8\ p = 9\ p = 10\ p = 11\ p = 12$$

$$(\%o5)\ \begin{pmatrix}1.000000000753427\\ 0.99999999214211\\ 2.000000002368154\end{pmatrix}$$

8.5 Вирішення звичайних диференціальних рівнянь

8.5.1 Методи вирішення задачі Коші

Серед завдань, з якими доводиться мати справу у обчислювальній практиці, значну частину становлять різні завдання, що зводяться до розв'язання звичайних диференціальних рівнянь. Зазвичай доводиться вдаватися до допомоги наближених методів розв'язання таких завдань. У разі звичайних диференціальних рівнянь залежно від того, чи ставляться додаткові умови в одній або кількох точках відрізка зміни незалежної змінної, завдання зазвичай поділяються на одноточкові (завдання з початковими умовами або завдання Коші) та багатоточкові. Серед багатоточкових завдань найчастіше у прикладних питаннях зустрічаються звані граничні завдання, коли додаткові умови ставляться кінцях аналізованого відрізка.

Надалі обмежимося розглядом чисельних методів вирішення завдання Коші. Для простоти викладу методів розв'язання задачі розглядатимемо випадок одного звичайного диференціального рівняння першого порядку.

Нехай на відрізку $x_0 \le x \le L$ потрібно знайти рішення $y(x)$ диференціального рівняння

$$y' = f(x, y), \quad (8.6)$$

задовольняє при $x = x_0$ початковій умові $y(x_0) = y_0$.

Вважатимемо, що умови існування та єдиності вирішення поставленого завдання Коші виконані.

Насправді знайти загальне чи приватне розв'язання завдання Коші вдається для дуже обмеженого кола завдань, тому доводиться вирішувати це наближено.

Відрізок $[x_0, L]$ накривається сіткою (розбивається на інтервали) найчастіше із постійним кроком h ($h = x_{n+1} - x_n$), і за якимось вирішальним правилом знаходиться значення $y_{n+1} = y(x_{n+1})$. Таким чином, результатом розв'язання задачі Коші чисельними методами є таблиця, що

складається з двох векторів: $x = (x_0, x_1, \ldots, x_n)$ — вектора аргументів і відповідного вектора значень шуканої функції $y = (y_0, y_1, \ldots, y_n)$.

Чисельні методи (правила), у яких знаходження значення функції у новій точці використовується інформація лише про одну (попередньої) точці, називаються однокроковыми.

Чисельні методи (правила), у яких знаходження значення функції у новій точці використовується інформація про кілька (попередніх) точках, називаються багатокроковими.

Із загального курсу звичайних диференціальних рівнянь стала вельми поширеною набув аналітичний метод, заснований на ідеї розкладання до ряду розв'язання задачі Коші. Особливо часто з цією метою використовується ряд Тейлора. І тут обчислювальні правила будуються особливо просто.

Наближене рішення $y_m(x)$ вихідного завдання шукають у вигляді

$$y_m(x) = \sum_{i=0}^{m} \frac{(x - x_0)^i}{i!} y^{(i)}(x_0), \qquad (8.7)$$

$$x_0 \leqslant x \leqslant b.$$

Тут $y^{(0)}(x_0) = y(x_0), \quad y^{(1)}(x_0) = y'(x_0) = f(x_0, y_0)$, а значення $y^{(i)}(x_0), i = 2, 3, \ldots, m$ знаходять за формулами, отриманими послідовним диференціюванням заданого рівняння:

$$y^{(2)}(x_0) = y''(x_0) = f_x(x_0, y_0) + f(x_0, y_0) f_y(x_0, y_0);$$

$$y^{(3)}(x_0) = y'''(x_0) = f_{x^2}(x_0, y_0) + 2f(x_0, y_0) f_y(x_0, y_0) +$$

$$+ f^2(x_0, y_0) f_{y^2}(x_0, y_0) + f_y(x_0, y_0)[f_x(x_0, y_0) + f(x_0, y_0) f_y(x_0, y_0)]; \qquad (8.8)$$

$$\ldots$$

$$y^{(m)}(x_0) = F_m(f; f_x; f_{x^2}; f_{xy}; f_{y^2}; \ldots; f_{x^{m-1}}; f_{y^{m-1}})|_{x=x_0, y=y_0}.$$

Для значень x, близьких до x_0, метод рядів (8.7) при досить великому значенні m дає зазвичай гарне наближення до точного рішення $y(x)$ Завдання (8.6). Однак із зростанням відстані $|x - x_0|$ похибка наближення шуканої функції поруч зростає по абсолютній величині (при одному й тому кількості членів ряду), і правило (8.7) стає зовсім неприйнятним, коли х виходить з області збіжності відповідного ряду (8.7) Тейлора.

Якщо у виразі (8.7) обмежитися $m = 1$, то для обчислення нових значень $y(x)$ немає необхідності перераховувати значення похідної, щоправда і точність рішення буде невисока.

При використанні системи комп'ютерної алгебри природнішим виглядає метод послідовних наближень Пікара.

Розглянемо інтегрування одиничного диференціального рівняння $\frac{dy}{dx} = f(x,y)$ на відрізку $[x_0, x]$ з початковою умовою $y(x_0) = y_0$. При формальному інтегруванні отримаємо:

$$\int_{x_0}^{x} \frac{dy}{dx} dx = \int_{x_0}^{x} f(x,y) dx$$

Процедура послідовних наближень методу Пікара реалізується згідно з наступною схемою

$$y_{n+1}(x) - y(x_0) = \int_{x_0}^{x} f(x, y_n(x)) dx$$

Як приклад розглянемо рішення рівняння $y' = -y$, при $y(0) = 1, x_0 = 0$:

```
(%i1)    rp:-y$ y0:1$ x0:0$ /*rp-права частина рівняння
*/
(% i4)   y1:y0+integrate(subst(y0,y,rp),x,x0,x);
     /* y1 - перше наближення */
```

$$(\%o4)\ 1 - x$$

```
(% i5)   y2:y0+integrate(subst(y1,y,rp),x,x0,x);
     /* y2 - друге наближення */
```

$$(\%o5)\ \frac{x^2 - 2x}{2} + 1$$

```
(%i6)    y3:y0+integrate(subst(y2,y,rp),x,x0,x);
```

$$(\%o6)\ 1 - \frac{x^3 - 3x^2 + 6x}{6}$$

```
(%i7)    expand(%); /* Очевидне рішення ОДУ, що
розглядається -
         експонента y=exp(-x).
         В результаті використання методу Пікара
         отримуємо рішення у вигляді ряду Тейлора */
```

$$(\%o7)\ -\frac{x^3}{6} + \frac{x^2}{2} - x + 1$$

8.5.2 Метод рядів, що не вимагає обчислення похідних правої частини рівняння

Природно поставити завдання про таке вдосконалення наведеного вище однокрокового методу, яке б зберігало основні його переваги, але не було б пов'язане зі знаходженням значень похідних правої частини рівняння

$$y_m(x_{n+1}) \approx \sum_{i=0}^{m} \frac{h^i}{i!} y^{(i)}(x_n), \qquad (8.9)$$

де $x_{n+1} = x_n + h$.

Щоб виконати цю умову (останнє), похідні у $y^{(i)}(x), i = 2, 3, \ldots, m$, що входять до правої частини рівняння (8.9), можна замінити за формулами чисельного диференціювання їх наближеними виразами через значення функції y' і врахувати, що $y'(x) = f[x, y(x)]$.

8.5.2.1 Метод Ейлера

У випадку $m = 1$ наближена рівність (8.9) не вимагає обчислення похідних правої частини рівняння та дозволяє з похибкою порядку h^2 знаходити значення $y(x_n + h)$ розв'язання цього рівняння за відомим його значенням $y(x_n)$. Відповідне однокрокове правило можна записати у вигляді

$y_{n+1} = y_n + h f_n$. (8.10)

Це правило (8.10) вперше було збудовано Ейлером і носить його ім'я. Іноді його називають також правилом ламаних чи методом дотичних. Метод Ейлера має відносно низький порядок точності. h^2 на одному кроці. Практична оцінка похибки наближеного рішення може бути отримана за правилом Рунґе.

Приклад реалізації методу Ейлера засобами Maxima наведено в наступному прикладі:

```
(%i1)
    euler1(rp,fun,y0,x0,xend,h):=block([OK,_x,_y,_y1,rez]
,
        _x:x0, _y:y0, rez:[_y], OK:-1, eps:0.1e-7,
        while OK<0 do (
            if ((_x+h>xend) or (abs(_x+h-xend)<eps))
                then (h:xend-_x,_x:xend, OK:1)
                else (_x:_x+h),

    _y1:makelist(float(_y[i]+h*subst([fun[i]=_y[i],x=_x],

    rp[i])),i,1,length(_y)),rez:append(rez,[_y1]),
            _y:_y1
        ),
        rez
    )$
```

Праві частини розв'язуваних диференціальних рівнянь передаються на функцію $euler1$ у списку rp. За умовчанням передбачається, що список імен залежних змінних fun, ім'я незалежної змінної x. Початкові значення незалежної та залежних змінних - список y_0 та скалярна величина x_0, межа інтервалу інтегрування - величина $xend$, крок інтегрування h.

Наступний приклад – звернення до функції $euler1$. Наведено рішення системи із трьох диференціальних рівнянь на інтервалі [0, 1] з кроком $h = 0.1$:

$$\begin{cases} \dfrac{dy}{dx} = -2y, \\ \dfrac{dv}{dx} = -5v, \\ \dfrac{dz}{dx} = 3x. \end{cases}$$

З початковими умовами $y(0) = 1.0; v(0) = 1.0; z(0) = 0$, Рішенням рівнянь даної системи будуть функції:

$$\begin{cases} y(x) = e^{-2x}, \\ v(x) = e^{-5x}, \\ z(x) = \dfrac{3}{2} \cdot x^2. \end{cases}$$

```
(%i2)    euler1([-2*y,-5*v,3*x],[y,v,z],[1,1,0],0,1,0.1);
```

$(\%o2)$ $[[1, 1, 0], [0.8, 0.5, 0.03], [0.64, 0.25, 0.09], [0.512, 0.125, 0.18], [0.4096, 0.0625, 0.3], [0.32768, 0.03125, 0.45], [0.262144, 0.015625, 0.63], [0.2097152, 0.0078125, 0.84], [0.16777216, 0.00390625, 1.08], [0.134217728, 0.001953125, 1.35], [0.1073741824, 9.7656249999999913 * 10^-4, 1.65]]$

Перевірити рішення можна порівнюючи графіки точного рішення та множини обчислених наближених значень. Приклад послідовності команд, що дозволяють виділити окремі компоненти рішення системи ОДУ та побудувати графік точного та наближеного рішення третього рівняння системи, представлений нижче (точні рішення – списки yf, vf, zf ; наближені рішення - списки yr, vr, zr; список значень незалежної змінної xg).

```
(%i3)    rez:euler1([-2*y,-
5*v,3*x],[y,v,z],[1,1,0],0,1.0,0.1)$
     n:length(rez)$
     yr:makelist(rez[k][1], k, 1, n)$
     vr:makelist(rez[k][2], k, 1, n)$
     zr:makelist(rez[k][3], k, 1, n)$
     xg:makelist(0.1*(k-1),k,1,n)$
     yf:makelist(exp(-2*xg[k]), k, 1, n)$
     vf:makelist(exp(-5*xg[k]), k, 1, n)$
     zf:makelist(3*xg[k]^2/2, k, 1, n)$
     plot2d ([[discrete, xg,zr],[discrete, xg,zf]],
              [style, points, lines])$
```

Зменшення кроку h призводить до зменшення похибки рішення (у цьому прикладі - крок 0.1).

8.5.2.2 Метод Рунге-Кутта

Викладемо ідею методу з прикладу завдання Коши:

$y' = f(x, y);$
$x_0 \le x \le b;$
$y(x_0) = y_0.$

Інтегруючи це рівняння в межах від x до $x + h$ $(0 < h < 1)$, отримаємо рівність

$$y(x + h) = y(x) + \int_x^{x+h} f\,[t, y(t)]dt, \tag{8.10}$$

яке за допомогою останнього інтеграла пов'язує значення рішення розглянутого рівняння у двох точках, віддалених один від одного на відстань кроку h.

Для зручності запису виразу (8.11) використовуємо позначення $\Delta y = y(x + h) - y(x)$ та заміну змінної інтегрування $t = x + h$. Остаточно отримаємо:

$$\Delta y = h \int_0^1 f[x + \alpha\, h, y(x + \alpha\, h)]d\alpha \tag{8.12}$$

Залежно від способу обчислення інтеграла у виразі (8.12) набувають різні методи чисельного інтегрування звичайних диференціальних рівнянь.

Розглянемо лінійну комбінацію величин $\phi_i,\ i = 0, 1, \ldots, q$, яка буде аналогом квадратурної суми і дозволить обчислити наближене значення збільшення Δy:

$$\Delta y \approx \sum_{i=0}^{q} a_i \phi_i,$$

де

$\phi_0 = hf(x, y);$
$\phi_1 = hf(x + \alpha_1 h; y + \beta_{10}\phi_0);$
$\phi_2 = hf(x + \alpha_2 h; y + \beta_{20}\phi_0 + \beta_{21}\phi_1);$
$\ldots$

Метод четвертого порядку для $q = 3$, Який є аналогом широко відомої в літературі чотириточкової квадратурної формули "трьох восьмих", має вигляд

$$\Delta y \approx \frac{1}{8}(\phi_0 + 3\phi_1 + 3\phi_2 + \phi_3),$$

де

$$\phi_0 = hf(x_n, y_n);$$
$$\phi_1 = hf(x_n + \frac{h}{3}, y_n + \frac{\phi_0}{3});$$
$$\phi_2 = hf(x_n + \frac{2}{3}h, y_n - \frac{\phi_0}{3} - \phi_1);$$
$$\phi_3 = hf(x_n + h, y_n + \phi_0 - \phi_1 + \phi_2).$$

Особливо відоме інше обчислювальне правило типу Рунге-Кутта четвертого порядку точності:

$$\Delta y = \frac{1}{6}(\phi_0 + 2\phi_1 + 2\phi_2 + \phi_3),$$

де

$$\phi_0 = hf(x_n, y_n),$$
$$\phi_1 = hf(x_n + \frac{h}{2}, y_n + \frac{\phi_0}{2}),$$
$$\phi_2 = hf(x_n + \frac{h}{2}, y_n + \frac{\phi_1}{2}),$$
$$\phi_3 = hf(x_n + h, y_n + \phi_2).$$

Метод Рунге-Кутта має похибку четвертого порядку ($\sim h^4$).

Функція Maxima, що реалізує метод Рунге-Кутта 4-го порядку, наведена в наступному прикладі (з друком проміжних результатів):

```
(%i1)    rk4(rp,fun,y0,x0,xend,h):=block(
         [OK,n,h1,_x,_y,_k1,_k2,_k3,_k4,rez],
         _x:x0, _y:y0, rez:[_y], OK:-1, h1:h,
n:length(_y),
         while OK<0 do (
             if (_x+h1>=xend) then (h1:xend-_x, OK:1),
             _k1:makelist(float(h1*subst([fun[i]=

    float(_y[i]),x=float(_x)],rp[i])),i,1,n),
             _k2:makelist(float(h1*subst([fun[i]=
                 float(_y[i]+_k1[i]/2),x=float(_x+h1/2)],
                     rp[i])),i,1,n),
             _k3:makelist(float(h1*subst([fun[i]=
                 float(_y[i]+_k2[i]/2),x=float(_x+h1/2)],
                     rp[i])),i,1,n),
             _k4:makelist(float(h1*subst([fun[i]=

    float(_y[i]+_k3[i]),x=float(_x+h1)],rp[i])),
                     i,1,n),
             _y1:makelist(float(_y[i]+)

    (_k1[i]+2*_k2[i]+2*_k3[i]+_k4[i])/6),i,1,n),
```

```
            rez:append(rez,[_y1]),
            print("x=",_x, "y=",_y),
            _x:_x+h1,
            _y:_y1
        ), rez
    )$
```

Приклад звернення до функції $rk4$ представлений наступною послідовністю команд (вирішувалась та сама система, що і при тестуванні методу Ейлера):

```
(%i2)    rk4([-2*y,-5*v,3*x],[y,v,z],[1,1,1],0,1,0.1);
x = 0    y= [1,1,1]
x = 0.1  y= [0.81873333333333,0.606770833333333,1.015]
x = 0.2  y= [0.67032427111111,0.36817084418403,1.06]
x = 0.3  y= [0.54881682490104,0.22339532993458,1.135]
x = 0.4  y= [0.44933462844064,0.13554977050718,1.24]
x = 0.5  y= [0.3678852381253,0.082247647208783,1.375]
x = 0.6  y= [0.30119990729446,0.04990547343658,1.54]
x = 0.7  y= [0.24660240409888,0.030281185705008,1.735]
x = 0.8  y= [0.20190160831589,0.018373740284549,1.96]
x = 0.9  y= [0.16530357678183,0.011148649703906,2.215]
x = 1.0  y= [0.13533954843051,0.006764675471380 5,2.5]
```